Pearson

Technical Research Report

Investigation of student learning in an introductory programming course using a digital learning environment

August 14, 2020

Authors
Derek Fay
Mark Armstrong
Katherine McEldoon
Julia Ridley

Pearson Global Product Organization
Efficacy & Learning Research
Impact Evaluation

Pearson

Table of contents

Executive summary

Introduction

Implementation of Revel in the study

Method

Results

Efficacy statements

Discussion

References

Appendix A. Data cleaning process

Appendix B. Alignment table and summary of research questions

Appendix C. Intake survey

Appendix D. Estimating scores on time factor via the lognormal model for response times

Appendix E. Analyses to assess generalizability of Revel-based metrics

Pearson

Executive summary

Overview of Revel

Revel is an interactive learning environment intended to help students prepare for class by reading a little, then doing a little. Each title within Revel consists of instructional text interspersed with videos, interactives, and scored tasks. The Revel title of interest to this work — *Introduction to Java Programming* by Y. Daniel Liang — leverages principles from learning science to take learners on the journey from reading, to visualizing and interacting with prewritten Java programs, and finally to writing their own programs with immediate feedback. This journey helps learners experience the path from an abstract problem to writing a program in Java that solves that problem.

Key features include programming tasks that provide students with opportunities to practice their coding skills in an authentic environment and receive immediate feedback; multimedia interactives featuring live coding examples; and animations and videos that help learners visualize step-by-step how to build a program.

Intended learner outcomes

The learner outcomes associated with this product, organized by learner outcome category, are:

Learner access and experience

- Learners have access to learning as intended in product strategy.
- Learners have a positive learning experience.

Timeliness and completion

- Learners persist through coursework.
- Learners are actively engaged in the learning experience.

Standard of achievement

- Learners are able to demonstrate skill acquisition.
- Learners express confidence in programming basics.
- Learners achieve competency in subject matter.
- Learners pass the course.

Learner progression

- Learners feel prepared for next level programming or computer science courses.

Pearson

Research aims and research questions

The present study was designed to address three of the above outcomes:

Timeliness and completion

- Learners persist through coursework.

Standard of achievement

- Learners are able to demonstrate skill acquisition.
- Learners achieve competency in subject matter.

The study initially sought to address outcomes related to access and experience, as well as progression. However, a student survey administered at the end of the semester to collect this information did not yield a high enough response rate.

The study design features a set of main research questions (MRQs) and a set of secondary research questions (SRQs):

Timeliness and completion

1. To what extent do students persist on challenging programming tasks in Revel? **(MRQ 1)**
2. To what extent do students complete assigned readings in Revel? **(SRQ)**
3. To what extent do students complete assigned programming tasks in Revel? **(SRQ)**
4. To what extent do students complete assigned readings in Revel on time? **(SRQ)**
5. To what extent do students complete assigned programming tasks in Revel on time? **(SRQ)**

Standard of achievement

1. How do students perform on Revel programming tasks? **(SRQ)**
2. Are students' scores on programming tasks in Revel related to their aggregate scores on programming tests? **(MRQ 2)**
3. Are students' aggregate gain scores on programming tasks (difference in maximum earned points and points earned on first attempts) in Revel related to their course grade? **(MRQ 3)**

Pearson

Key findings

Main research questions

This study investigated data from 114 students enrolled in an introductory computer science course at the University of North Carolina at Greensboro (UNCG) from the Spring 2018 semester to the Fall 2019 semester, who used *Introduction to Java Programming* by Y. Daniel Liang. We used a multiple regression model that controlled for measures of prior achievement, prior experience with computer science, time spent on Revel reading and programming tasks, on-time completion of Revel reading tasks, and demographic attributes.

- 87% of the students in the analytic sample came from the 2019 semesters.
- In analyses involving course grades, we used final course grades with the contribution of scores on Revel assignments removed.

Given the results of this study, we can make the following statements about the efficacy of Revel for *Introduction to Java Programming* by Y. Daniel Liang:

Timeliness and completion

- 90% of students persisted on more than 85% of the tasks on which they initially had an incorrect first attempt.

Standard of achievement

- Higher first attempt scores on Revel programming tasks are associated with higher programming test scores.
    - In particular, a 10-point increase in the student's first-attempt Revel programming task score is associated with nearly a 5-point increase on their programming test scores.
- In general, students who persisted and made higher gains on Revel programming tasks from their first attempt score to their highest score tended to earn higher final course grades.
    - In particular, a 10-point increase in the student's gain score on Revel programming tasks is associated with nearly a 4-point increase in final course grades.

Secondary research questions

Timeliness and completion

- Most students complete most of the assigned tasks in Revel — both programming and reading — but programming tasks are more likely to be completed on time.

Standard of achievement

- The average percentage of points earned on the first attempt was just 32.9%, reflecting the degree of challenge these tasks pose to students. However, return attempts tend to pay off in improved scores, as the typical student (based on the median gain score) sees their performance improve by 73% over repeated attempts.

Recommendations

The results of the study suggest that students with higher first attempt scores on Revel programming tasks tend to do better on programming tests. In addition, students who persist on initially incorrect programming tasks in Revel until earning high scores tend to receive higher final course grades. Given these results, we recommend that instructors using Revel for *Introduction to Java Programming*:

- assign programming tasks and encourage students to complete them by making them account for a portion of students' grades
- encourage students to complete their assignments before they come to class by setting exercise due dates and not accepting late work
- give students unlimited attempts to complete these exercises without deducting points for minor typos or omissions
- let students know the exercises are challenging and that they should keep trying even if they aren't successful on their first attempt

Instructors should also encourage students to:

- complete reading and programming tasks before class — this frees up class time for discussion
- take their time on their first attempts — the more polished the first attempt, the fewer mistakes they will need to correct on subsequent attempts
- keep trying even if they aren't successful on their first attempt (acknowledge that the programming exercises are challenging)
- contact their instructor for help if they don't see improvements in their score despite multiple attempts

Notably, the correlational design used in this study does not allow for claims of causality. However, the results are consistent with the notion that using Revel can help students achieve more in the course, particularly to the extent that they persist on challenging programming tasks.

Next steps

Based in part on the findings of this study — particularly the findings on persistence and relationship between gain scores and course grades — Pearson aims to update the default setting for this title to allow for unlimited attempts at programming tasks, without penalties for repeated attempts.

Pearson

Introduction

Background

Revel is an interactive learning environment intended to help students prepare for class by reading a little, then doing a little. Each title within Revel consists of instructional text interspersed with videos, interactives, and tasks. The Revel title of interest to this work — *Introduction to Java Programming* by Y. Daniel Liang — leverages principles from learning science to take learners on the journey from reading, to visualizing and interacting with prewritten Java programs, and finally to writing their own programs with immediate feedback. This journey helps learners experience the path from an abstract problem to writing a program in Java that solves that problem.

Mark Armstrong, an instructor at the University of North Carolina at Greensboro (UNCG), has used the Liang title to teach an introductory computer science course since 2017. This study examines how the students in this course learn with Revel for *Introduction to Java Programming*, analyzing students' learning behaviors in Revel and their associations with achievement in the course. Analysis is performed using data from courses taught between Spring 2018 and Fall 2019. In addition, we describe how Revel for *Introduction to Java Programming* is implemented in the course using a learner-centered, active learning approach.

Description of Revel

The design of Revel is aligned with a number of learning science principles, ensuring learners get more out of their experience. The Revel experience is centered around the principle of "read a little, do a little". Doing a little as they go encourages students to come to class prepared (in this case, to complete assigned reading and programming tasks before class). When students are prepared for class, this allows the instructor to design the in-class experience to be more learner-centered and active as opposed to just passive lecturing to cover content.

Formative assessment that includes timely, informative feedback is an important learning science principle that has been incorporated into Revel's design. Specifically, formative assessment helps learners monitor progress toward learning goals, identify and correct their mistakes, and think deeply about the correct information (Hattie & Timperley, 2007).

The main types of Revel formative assessment used in the UNCG course were the end-of-section programming tasks and the end-of-chapter programming tasks. First and foremost, the programming tasks in Revel provide opportunities for students to practice their coding skills. Real competence only comes with extensive practice (e.g. Ericsson, Krampe, & Tesch-Römer, 1993; Kotovsky, Hayes, & Simon, 1985). Practice not only strengthens the learner's memory for the information (making sure it sticks in long-term memory), it also weaves it into the tapestry of related knowledge more deeply, and allows the learner to remember information with less and less effort. Being able to call something to mind automatically and without effort is key as the learner takes on increasingly complex tasks (Koedinger, Corbett, & Perfetti, 2012).

In Revel for *Introduction to Java Programming*, students have the opportunity to practice their skills in authentic programming tasks. Authentic activities are complex, ill-defined tasks with real world relevance, often completed collaboratively (Herrington, Oliver, & Reeves, 2003). These open-ended coding tasks require students to create and run their own programs, making them quite challenging. Research shows that active engagement in authentic disciplinary practices results in enhanced learning outcomes (Nathan & Sawyer, 2014).

Learners have multiple attempts on these tasks, and receive immediate feedback on each attempt. Immediate feedback is strongly beneficial for learning (Azevedo & Bernard, 1995; Shute, 2008). In particular, when students are beginning to learn something new and potentially difficult for them, receiving immediate feedback can keep them on track and help them achieve more (Dihoff, Brosvic, Epstein, & Cook, 2004). Feedback on performance increases students' learning and transfer, and supports their ability to monitor their own understanding (Butler & Winne, 1995). The learning from feedback on incorrect attempts shores up the learner's understanding of the problem (Anderson, Reder, & Simon, 1999), and develops the motivation and confidence necessary to take on even more challenges in the future (Duckworth & Gross, 2014). Figure 1 depicts an example programming task from Revel.

Figure 1. Sample programming task from Revel for Introduction to Java Programming

Furthermore, Revel for *Introduction to Java Programming* features a type of multimedia interactive with live code. These interactives are interspersed with the text, giving students more low-stakes opportunities to practice their coding skills. From these interactives, students receive feedback on the correctness of syntax compiling and execution, as well as tips on best coding practices for the problem. Another critical set of assets in this Revel title is the animations and videos. Learning science best practice recommends minimizing distracting content in instructional videos (Guo, Juho, & Rubin, 2014), and says that using videos is particularly helpful for visualizing step-by-step processes that otherwise wouldn't be visible to us because of timescale, size, or non-physical nature. Revel for *Introduction to Java Programming* uses videos and animations to show how a code snippet runs at each line, giving students a glimpse into what is otherwise an invisible process. These videos and animations also act as worked examples, by breaking down a complex problem-solving activity (running code) into smaller, more digestible steps for the student to process. Finally, at key steps of visualizing the code, students are prompted with questions to check their understanding, providing even more opportunities for timely feedback.

Intended learner outcomes

The learner outcomes associated with this product, organized by learner outcome category, are:

Learner access and experience

- Learners have access to learning as intended in product strategy.
- Learners have a positive learning experience.

Timeliness and completion

- Learners persist through coursework.
- Learners are actively engaged in the learning experience.

Standard of achievement

- Learners are able to demonstrate skill acquisition.
- Learners express confidence in programming basics.
- Learners achieve competency in subject matter.
- Learners pass the course.

Learner progression

- Learners feel prepared for next level programming or computer science courses.

The present study

The present study was designed to address three of the above outcomes:

Timeliness and completion

- Learners persist through coursework.

Standard of achievement:

- Learners are able to demonstrate skill acquisition.
- Learners achieve competency in subject matter.

The study initially sought to address outcomes related to access and experience, as well as progression. However, the student survey administered at the end of the semester to collect this information did not yield a high enough response rate.

Pearson

The study design features a set of main research questions (MRQs) and a set of secondary research questions (SRQs). The efficacy statements generated from this research are driven by the results for the MRQs. The SRQs are not being used to generate statements about the efficacy of Revel for *Introduction to Java Programming*, but they target aspects of the product which are of substantive interest in their own right. The MRQs focus on the extent to which:

- students persist in their learning despite obstacles: in this study, whether they make repeated attempts on tasks initially scored incorrect
- students achieve competency in programming: in this study, whether students' initial learning efforts (evidenced by their first attempt performance on Revel programming tasks) are related to their overall competency in programming (evidenced by scores on instructor-created programming tests)
- students acquire programming skills: in this study, whether improvements in programming skills over time (evidenced by gain scores on Revel programming tasks over repeated attempts) are related to their programming skill level at the end of the course (evidenced by their final course grades)

With the main research questions indexed (MRQ 1, MRQ 2, MRQ 3) for reference throughout, the full set of research questions consists of the following:

Timeliness and completion

1. To what extent do students persist on challenging programming tasks in Revel? **(MRQ 1)**
2. To what extent do students complete assigned readings in Revel? **(SRQ)**
3. To what extent do students complete assigned programming tasks in Revel? **(SRQ)**
4. To what extent do students complete assigned readings in Revel on time? **(SRQ)**
5. To what extent do students complete assigned programming tasks in Revel on time? **(SRQ)**

Standard of achievement

1. How do students perform on Revel programming tasks? **(SRQ)**
2. Are students' scores on programming tasks in Revel related to their aggregate scores on programming tests? **(MRQ 2)**
3. Are students' aggregate gain scores on programming tasks (difference in maximum earned points and points earned on first attempts) in Revel related to their course grade? **(MRQ 3)**

Further details about the computation of the measures and metrics are given later. Appendix B gives a complete summary of the research questions, inclusive of alignments to learner outcomes, the requisite measures, and metrics.

Pearson

Implementation of Revel in the study

The course of interest is a semester-length introductory course in computer science at the University of North Carolina at Greensboro (UNCG). The course is described as emphasizing "...problem solving, problem-solving techniques, and software design principles and techniques". These goals for the course are taught via programming.

Although any programming language could accomplish the course goals, the instructor's reason for selecting Java programming was employability trends; job postings for computer scientists and programmers consistently listed Java as an essential requirement. Liang's *Introduction to Java Programming* was in turn selected as the text for the course owing in part to its tight alignment to the curriculum. The title was first used in the Fall 2017 semester and has been used consistently ever since. The programming tasks in Revel (described above) contribute 10% to students' overall course grades.

The present study uses data from four semesters (Spring 2018, Fall 2018, Spring 2019, and Fall 2019). Of the four semesters, three were taught by one instructor (Spring 2018, Fall 2018, and Fall 2019) and one was taught by another (Spring 2019). The instructor for Spring 2019 largely modeled the course and Revel implementation to be like the other instructor, who typically teaches the course. For reasons described later, the majority of students (87%) who agreed to participate in the study were from the Spring and Fall semesters of 2019.

Coverage of Revel content

The Revel title used in the course includes 29 chapters. This being an introductory course, only the first nine chapters were used. Figure 2 shows the progression of the chapter assignments for the Fall 2019 semester without loss of generality to the other three semesters of data in the present study. Chapters from Revel for *Introduction to Java Programming* are shown along the vertical axis, and the number of weeks since the release of the first assignment are shown along the horizontal axis. Looking within each chapter, the amount of time devoted to each topic (in weeks) is represented by the distance between the two endpoints. Moving from left to right along the horizontal axis within panels shows the sequence of topics and the duration of the semester those topics were covered.

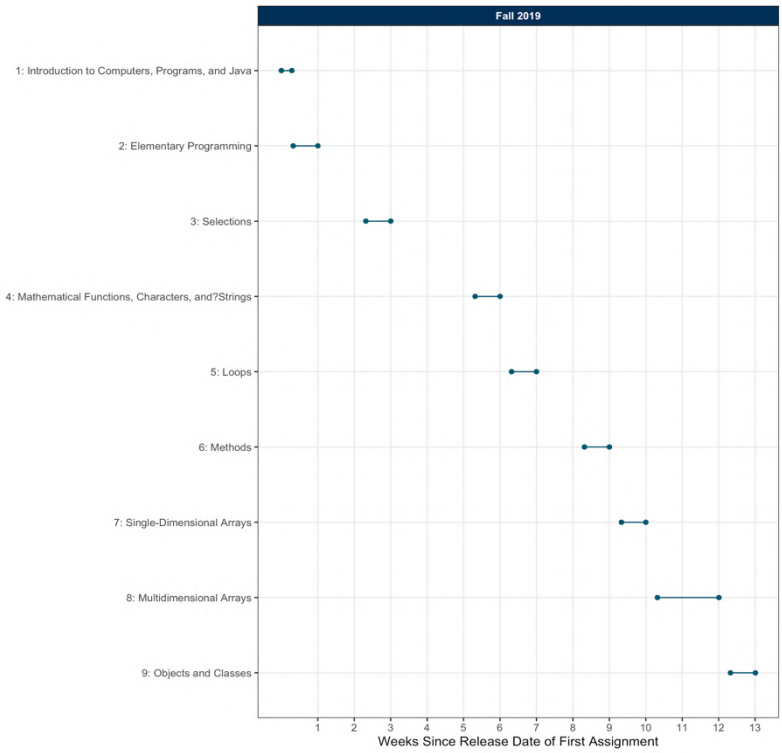

Figure 2. Sequence and duration of coverage of Revel chapters for the Fall 2019 semester

Example topics from the nine chapters used in the course include an overview of operating systems and programming languages in Chapter 1; mathematical functions and data types in Chapter 4; and two-dimensional arrays in Chapter 8. Notably, based on analysis of platform data for all courses using Revel for *Introduction to Java Programming*, the coverage of topics in the UNCG course is highly representative of the majority of courses that have historically used this title. Of all courses that used the title, 70% were found to align to the same content profile as the courses (that is, semesters) in the present study. We revisit this point in the Discussion, when characterizing the degree of generalizability of the results of the study.

Integration of Revel into the course

Across all four semesters, the course was structured with two in-person meetings per week, with the first being a lecture and the second being a structured programming laboratory.

Figure 3 shows the typical schedule used by the instructors for teaching topics in the course. For any given topic, students are first introduced to a topic via the e-text and end-of-section programming tasks in Revel on the Friday prior to a formal lecture on that topic. The instructor's lecture for that topic occurs early in the subsequent week (typically Monday or Tuesday). Following the lecture, students are assigned end-of-chapter programming tasks in Revel in preparation for the instructor-developed programming laboratory. The Revel projects are due the morning of the laboratory, but can be referenced (along with other Revel content) by students during the laboratory. The cycle starts again with a new topic on Friday.

Monday	Tuesday	Wednesday	Thursday	Friday
				Revel reading content & end-of-section programming tasks released for Topic X
	• Revel reading content & end-of-section programming tasks for Topic X due before lecture • Formal lecture for Topic X content and end-of-section programming tasks • Revel end-of-chapter programming tasks released for Topic X		• Revel end-of-chapter programming tasks for Topic X due morning of programming lab • Students complete instructor-created programming lab assignment	Revel reading content & end-of-section programming tasks released for Topic Y

Figure 3. Weekly cadence for integrating Revel into the course as described by the instructor (the schedule shown here was for the Fall 2019 semester, but it generalizes to the other semesters considered in this work)

It is worth noting that instruction in the course was learner-centered (Wright, 2011). Formal lecturing was minimized by:

- devoting the first half of the lecture to discussing students' questions about the assigned reading and end-of-section programming tasks in Revel
- giving students opportunities to practice programming concepts in groups
- performing demonstrations of programming concepts

The programming labs were almost entirely student driven, save for unobtrusive, onsite monitoring by the instructor on a separate computer. The lab assignments were designed to model the strict deadlines software developers are held against in work settings. The strict deadlines are operationalized by deducting points for late assignments, with full credit awarded for on-time completion; 10 points deducted within the first five minutes after the deadline; and no points given at any point thereafter. During an onsite observation, three sections of students were observed to immediately get to work, absent any significant direction from the instructor; the class was filled with dialog amongst students as they worked diligently on the programming assignment.

The actual assignments students engaged in during the programming labs were typically completed in groups using *paired programming*, a common practice among professional software developers (Hanks, Fitzgerald, McCauley, Murphy, & Zander, 2011). Paired programming involved assigning two students to complete the programming lab together. While one student programmed during the first half of the lab, the other student observed and helped identify potential errors; students would simply switch positions for the second half of the lab.

Implementation of Revel features in the course

Table 1 summarizes features of implementation with respect to assessment and reading assignments. Stated generally, the implementation of Revel has been fairly consistent over the four semesters of use. It appears the only substantive difference in implementation across the four semesters was the number of end-of-chapter programming tasks assigned to students. In particular, the instructor teaching the course during the Spring 2019 semester assigned more of these kinds of assignments than the instructor who typically teaches the course.

Table 1. Summary of Revel implementation in the course

	Assessment assignments			Reading assignments	
Semester	**Number of section tasks**	**Number of chapter tasks**	**Total programming tasks**	**Number of chapters**	**Number of chapter sections**
Spring 2018	128	22	150	9	97
Fall 2018	137	23	160	9	98
Spring 2019	148	42	190	9	112
Fall 2019	130	24	154	9	100

Another important aspect of implementation was the number of attempts students were allotted to complete Revel programming tasks. The default setting in Revel allows up to three attempts on programming tasks, with each return attempt resulting in a reduction in the number of points that could be earned. The default setting was changed by the instructors to allow for unlimited attempts — without penalties for return attempts — on all assigned programming tasks in Revel. The choice to allow unlimited attempts was made following the first semester of usage in Fall 2017. The reason for the switch is that when using the default settings, the instructors noticed that students were losing points owing to minor errors (such as typos or omitted semicolons) that had little to do with students' knowledge, skills, and abilities.

Pearson

Method

The research questions in the present study were addressed via descriptive and correlational methods. The correlational methods involved the use of statistical models to connect students' knowledge and skill, with Revel performance used as a proxy measure, to summative measures of achievement collected outside of Revel, whilst controlling for measures of prior achievement, prior programming experience, time spent on reading and programming tasks, on-time completion of reading tasks, and demographic attributes when possible. The results from these models should not be interpreted as reflecting causal relationships between variables.

Participants

The final sample for the present work consists of *N* = 114 students across four semesters (Spring 2018, Fall 2018, Spring 2019, and Fall 2019) and two instructors. Table 2 shows the distribution of students across the four courses.

The majority of the sample comes from the Spring and Fall semesters of 2019 (*n* = 99). This is because of differences in the procedures for recruiting students in the 2019 semesters versus the 2018 semesters. The 99 students from the Spring and Fall semesters in 2019 were recruited in person by either the instructor or a member of the Pearson research team. These students were asked to participate in all aspects of the study (that is, to share their grades in the course, complete surveys, and other activities described below). The remaining 15 students were recruited via an email sent by the instructor. For these students, participation was limited to sharing their course grade and allowing that data to be joined with their data from the Revel platform.

Table 2. Total number of consenting students that completed the course by semester

	Students recruited via email		Students recruited in person		
	Spring 2018	**Fall 2018**	**Spring 2019**	**Fall 2019**	**Total**
Consented *n*	7	8	38	72	**125**
***n* Completed course (analytic sample)**	7	8	33	66	**114**
% of total analytic sample	6.1%	7.0%	28.9%	57.9%	**—**
Total enrollment (% completed of semester enrollment)	103 (6.8%)	96 (8.3%)	93 (35.5%)	95 (69.4%)	**387**

Because of rounding, percentages may not sum to one. The total enrollment was estimated by counting the number of distinct student identifiers within each course in Revel.

Pearson

Table 3 shows the distribution of students' demographic attributes. Notably, demographic data were only available from the 99 students from the Spring and Fall semesters in 2019. Immediately following a question gauging their willingness to participate in the study, these students were taken to an intake survey, part of which requested demographic information. No attempt was made to collect the analogous information from students from prior semesters.

Table 3. Demographic attributes of consenting student sample (*n* = 99, Spring 2019 and Fall 2019 semesters only)

Attribute	Response	Count	%
Age	18–24	92	**92.9%**
	25–29	5	5.1%
	30–34	2	2.0%
Gender	Male	73	**73.7%**
	Female	25	25.3%
	Other	1	1.0%
English first language	Yes	79	**79.8%**
	No	20	20.2%
Full time status	Not full time	27	27.3%
	Full time	72	**72.7%**
High school GPA	Can't remember	14	14.1%
	2.1–2.5	3	3.0%
	2.6–3.0	12	12.1%
	3.1–3.5	26	26.3%
	3.6–4.0	44	**44.4%**
Major	Computer Science	85	**85.9%**
	Other	14	14.1%

Bolded values indicate the most frequently endorsed response.

Pearson

Data collection

Table 4 summarizes the data sources for the measures relevant to the study[1]. Additional details are provided on each measure following the table.

Table 4. Summary of data sources for relevant measures in the study

Source of data	Description of data source and collection of data/relevant measures
Revel usage and performance	Students generate data as they interact with Revel throughout the course. The smallest grain size for this data is a timestamped log of their interactions with the e-text and programming tasks. All other tables used in the current work can be viewed as some level of aggregation of those interactions with Revel. ***Relevant measures:*** *persistence on programming tasks; first attempt performance on programming tasks; gain scores (score on first attempt to highest score) on programming tasks; time spent on programming tasks; time spent on reading tasks; completion of assigned e-text and programming tasks.*
Student intake survey (Spring and Fall semesters of 2019 only)	The intake survey can be viewed as an extension of the consent form for students who chose to participate in the study. The intake survey asked students to self-report demographic attributes (summarized above); various indicators of prior achievement; perceptions about computer science; and expectations for the course. ***Relevant measures:*** *high school grade point average; gender; guardian educational level; number of computer science courses inclusive of high school..*
Institution	The instructor provided information on students' performance in the course. All components contributing to students' grades in the course were provided, with two versions of their course grade. One version included the Revel portion and the other version excluded the Revel portion. ***Relevant measures:*** *programming test scores; course grade with Revel excluded.*

The relevant Revel data and intake survey data were sent to the instructor. The instructor returned the data with all identifying information removed and a new identifier untethered to Pearson data structures. Using the same identifier, the instructor also provided all relevant information about students' performance in the course. All subsequent processing of the data (such as applying filters or joining data sources) was completed by the lead researcher at Pearson.

[1] *Notably, qualitative data were collected to contextualize and/or validate the sources of data, particularly the Revel data. These qualitative data sources included instructor surveys, instructor interviews, and student focus groups. We also attempted to collect survey data from students at the end of the Spring and Fall semesters in 2019. Unfortunately, the combined sample size was too low (N = 33 completed surveys) for inclusion in the study.*

Measures

Revel usage and performance data

Persistence rate on Revel programming tasks (MRQ 1)

A review of published research on the skills associated with self-management has defined persistence as "applying and maintaining appropriate effort to tasks in spite of obstacles or difficulty" (Yarbro & Ventura, 2018, p. 8). The authors argue that evidence of persistence can come from observing a learner who "continues working on a project despite setbacks" (p. 8). Similarly, DiCerbo (2016) defines persistence as "...continuing with a task despite obstacles, difficulty, and/or failure" (p. 778).

Previous studies have measured persistence using students' attempt behavior. For example, Thomas and Pashley (1982) used the number of attempts students made at solving unsolvable puzzles as an index of persistence. Reiher & Dembo (1984) looked at the number of times students attempted problems. More recently, Choi and Bogucki (in press) used a similar definition of persistence in their study modeling student outcomes in computer-based formative tasks, like the programming tasks in Revel.

In this study, we operationalize persistence as follows. After an incorrect initial attempt on an assigned Revel programming task, students were deemed to have persisted on the task if they met either of the following criteria:

- The student made at least two return attempts but did not produce a correct answer.
- The student received partial or full credit on the task on any attempt after the first attempt.

Each student's persistence rate was computed as the percentage of incorrect first attempts on programming tasks for which students were observed to have met either of the above criteria.

First attempt performance on Revel programming tasks (MRQ 2, SRQ)

Students' performance within Revel was computed as the percentage of aggregate points earned across all first attempts on assigned programming tasks. The numerator was the total earned points on first attempts for all attempted programming tasks; no points were given for programming tasks that were not attempted. The denominator was the total points possible across all assigned programming tasks within Revel.

First attempt performance is of interest because it captures learning that has occurred by the time students reach the programming tasks. Before attempting the end-of-section programming tasks, students were expected to have read the assigned section in the text, which includes the interactive tools that demonstrate the conceptual features of a program (such as video tutorials). In addition, before attempting end-of-chapter programming tasks, students were intended to have:

- completed the end-of-section programming tasks
- attended a lecture focused on discussing students' questions about the reading and section programming tasks

Pearson

Thus, students were meant to come into the tasks with a fair bit of preparation. However, given that they are open-ended in nature (rather than multiple-choice questions) and students had multiple attempts to complete them, it is also important to look at how their scores improved across successive attempts.

Gain scores on Revel programming tasks (MRQ 3)

Students' gain scores within Revel were computed as the ratio of realized gain (numerator) in total points earned on programming tasks over possible gain (denominator) in total points as follows:

$$G_i = \frac{Realized\ Gain}{Possible\ Gain} = \frac{\left(\sum_{j=1}^{J} U_{ij}^{Max} - \sum_{j=1}^{J} U_{ij}^{First}\right)}{\left(\sum_{j=1}^{J} U_{ij}^{Possible} - \sum_{j=1}^{J} U_{ij}^{First}\right)}, \quad (1)$$

where for a given student *i* on programming task *j*, U^{Max} is maximum earned points; U^{First} is the points earned on the first attempt; and $U^{Possible}$ is the number of points possible. Summing over programming tasks in the numerator yields the total number of points a student *actually* gained from their first attempt to their highest score; summing over programming tasks in the denominator yields the total number of points a student *could* have gained over their first attempt performance. Notably, summing over programming tasks to compute $U^{Possible}$ yields the total number of points across all Revel programming tasks assigned in the course and, accordingly, will be constant for all students in the same course.

For example, consider a hypothetical course in which an instructor assigns programming tasks totaling 100 points. Further consider two students: Student A who earns 25 points on their first attempt and Student B who earns 70 points on their first attempt. Both students would achieve the same gain score of 0.33 if Student A ultimately earns 50 points [0.33 = (50 - 25) / (100 - 25) = 25 / 75] and Student B earns 80 points [0.33 = (80 - 70) / (100 - 70) = 10 / 30].

The combination of both students' initial attempt scores and their gain scores helps to form a more complete picture of their learning in Revel.

Completion rates for assigned reading content and programming tasks (MRQ 2, MRQ 3, SRQ)

Reading completion rate. A student was deemed to have completed assigned reading content if they were observed to have loaded and unloaded the content at least once for any length of time at any point throughout the course. Each student's reading completion rate was simply the percentage of all assigned readings they loaded and unloaded.

Programming task completion rate. Students had to attempt an assigned programming task at least once in order to be credited with completing it, regardless of the correctness of their attempt. Each student's programming task completion rate was computed as the percentage of assigned programming tasks they attempted.

On-time reading completion rate. The definition for reading completion rates, above, applies to students' on-time completion for assigned reading content, with the exception that students were required to have loaded and unloaded the content within the assignment release and due dates.

On-time programming task completion rate. The definition for programming task completion rates, above, applies to students' on-time completion for assigned programming tasks, with the exception that students were required to have attempted the programming task within the assignment release and due dates.

Time spent on programming and reading tasks (MRQ 2, MRQ 3)

The potential for educational technology to impact students' outcomes depends on the amount of time they spend interacting with it, at least in part. Said differently, we cannot reasonably expect the effects targeted in our main research questions to be impacted by Revel if students spend too little time interacting with programming and reading tasks.

In the form of a factor score (described shortly), we include the time students spent on programming and reading tasks as a covariate in our analyses for addressing MRQ 2 and MRQ 3, which allows us to increase our power to detect effects associated with the main research questions. In what follows, we provide an overview of the model from which the time factors were extracted. Computational details are available in Appendix D.

The present study modeled students' time spent on assigned reading and programming tasks (separately) using a normal model applied to students' log transformed time spent on tasks. The foundations of this approach are tied to modeling task-level response times on educational assessments (van der Linden, 2006) but it has more recently been used in the context of educational technology platforms (e.g., Rushkin, Chuang, & Tingley, 2019). The model assumes a normal density for the log time spent on tasks (denoted by RT for consistency with prior research)[2]:

$$f\left(RT_i;\theta_i;\ \alpha_j;\beta_j\right) = \frac{\alpha_j}{RT_i\sqrt{2\pi}} exp\left\{-\frac{1}{2}\left[\alpha_j(ln\ RT_i - \left(\beta_j - \theta_i\right))\right]^2\right\}, \quad (2)$$

where β_j is the time intensity parameter for task *j*; θ_i is the factor score[3] for the student *i*; and α_j is the discrimination parameter for task *j*. The model was estimated via Markov Chain Monte Carlo (MCMC) as implemented in JAGS (Plummer, 2013); further details and the corresponding JAGS code are available in Appendix D.

[2] *Readers familiar with traditional item response theory (IRT) models may see parallels in the lognormal response model for response times, with the time factor (θ) being akin to the ability parameter; the time intensity (β) being akin to the difficulty or location parameter; and task discrimination parameter (α) retaining the same label.*

[3] *The factor score produced by the model is most commonly characterized as a "speed" or "slowness" parameter.*

The relevant parameter for this work is the time factor score, θ. Relative to more heuristic methods (such as total time, average time, or median time), the primary advantage of the model in the above equation is the accumulation of all evidence available from the time students spend on assigned tasks, which serves as a more reliable measure of students' tendencies. In addition, the model controls for the time intensity (β_j) associated with each task.

The model takes the full set of log-transformed values of time spent as input to concurrently estimate the time intensity for each task and a factor score (θ) for each student. The data for programming and reading consisted of $J = 86$[4] and $J = 112$ tasks, respectively. For the purposes of modeling, the distribution of students' time spent on programming and reading were rescaled to a T-score distribution (that is, $M = 50$, $SD = 10$). Importantly, owing to the scaling in the model, *lower* scores correspond to taking *more* time; *higher* scores correspond to taking *less* time.

Prior research in the context of educational assessments suggests a non-linear relationship between the time students spend on assessment tasks and performance (e.g., Beck, 2005). To adequately control for this relationship, the models for addressing MRQ 2 and MRQ 3 will include a linear and quadratic effect for students' time on assessment tasks. To our knowledge, no research has investigated the nature of the relationship between the time students spend reading and performance. Accordingly, the effect for students' time spent reading is included as a means to separate it from their time on programming tasks.

Student intake survey

The full student intake survey administered at the beginning of each semester is available in Appendix C. For the purposes of the study, the student intake survey was a source of crucial control variables, particularly measures of prior achievement, prior experience with course content, and demographic information. The most important controls include high school grade point average (prior achievement), and guardian education level as a proxy for socioeconomic status. Although these measures are self-reported, Kuncel, Credé, and Thomas (2005) found a correlation of 0.82 between actual and self-reported SAT scores in their meta-analysis, which suggests that self-reported prior achievement scores are a reliable indicator of actual scores.

Institution data

The institution sent the grade information for all consenting students. In addition to final course grades, students' attendance, performance in Revel, grades on laboratory assignments, and grades on written and programming tests were provided. Two versions of the final course grades were provided, one with the Revel portion included and one without. The version with the Revel portion removed was used for modeling in an effort to control endogeneity between students' Revel performance and course grades. Figure 4 shows the distributions for the two outcome variables investigated in MRQ 2 and MRQ 3.

---

[4] *Students typically encounter more than one end-of-section programming task at the end of sections of reading. To reduce the number of parameters, the time spent across end-of-section programming tasks within a section was aggregated up to the corresponding section.*

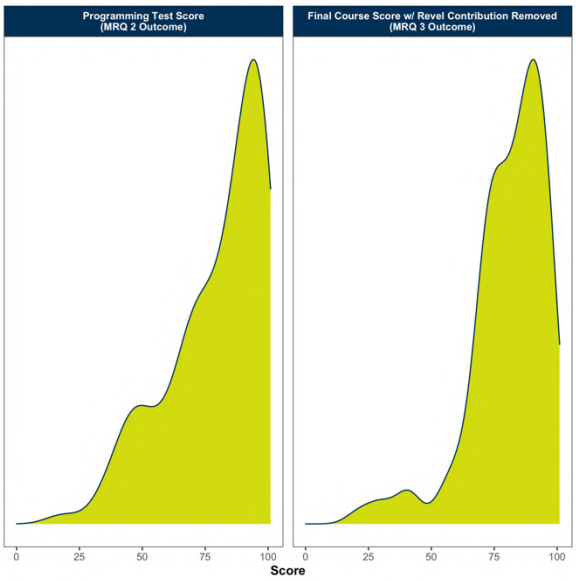

Figure 4. Distributions of outcome variables for MRQ 2 and MRQ 3

Students generally performed quite well on the instructor-created programming tests and in the course overall, with median scores of 86% and 84%, respectively.

Analysis method

The present research pursued descriptive and graphical approaches to address:

- the main research question on persistence rates (MRQ 1)
- all secondary research questions that could be addressed

The interests of MRQ 2 and MRQ 3 were associations between performance in Revel and external measures of student achievement. To address MRQ 2 and MRQ 3, linear regression models were fitted to the data using R 3.6.3 (R Core Team, 2020)[5]. This approach makes it possible to investigate the relationships of interest while accounting for potentially confounding variables.

Controlling for potential confounding variables

An important advantage of regression models is the ability to include variables which are not of substantive interest but, if left unaccounted for, may be unknown, alternative explanations for the effects of interest. In the context of educational research — particularly research targeting student outcomes, such as achievement — potential confounds that are critical to control for include prior achievement and knowledge; direct or proxy indicators of socioeconomic factors; and demographic attributes (What Works Clearinghouse, 2016).

Other variables should also be included as appropriate given features of the data (different semesters, for example). Table 5 summarizes each of the control variables used in the present study.

---

[5] Due to the non-normality in the outcomes, a sandwich estimator was applied to the estimates for the pooled regression models to investigate the impact of heteroskedasticity. Estimates of standard errors were very similar to those assuming homoskedasticity and patterns of statistical significance did not change. Accordingly, we report the results that assume homoskedasticity.

Pearson

Table 5. Summary of variables used to control statistical confounds

Confound type	Variable	Data source	N	% missing
Prior achievement	High school grade point average (GPA)	Intake survey	99	13.2%
Prior knowledge	Count of computer science courses completed, including high school	Intake survey	99	13.2%
Socioeconomic factors	Guardian education level	Intake survey	99	13.2%
Demographic	Gender	Intake survey	99	13.2%
Study design	Indicator for the four semesters	N/A	114	0%

N/A = not applicable.

Treatment of missing data

Missing data were handled via multiple imputation (MI) as implemented in the MICE R package. This approach allowed all 114 cases to be retained in the analytic sample.

The use of MI rests on the assumption that the missing data are missing completely at random (MCAR) or missing at random (MAR). For the present study, no attempt was made to collect intake survey data from students who completed the course in 2018, and accordingly intake survey data were missing by design for these students. Planned missing data designs (such as the present study) yield missing data that are MAR or MCAR (Enders, 2010) and accordingly support the use of MI.

In the context of the present study, MI was used to create 50 imputed datasets to address the two correlational research questions (MRQ 2 and MRQ 3), which included four variables from the intake survey to control for statistical confounds. The four variables from the intake survey included high school grade point average (HS GPA); the number of computer science courses completed inclusive of high school; guardian education level; and gender.

For HS GPA, missing records were combined with students who reported that they could not remember their HS GPA, yielding a more general "Unknown" group[6]; this produced a complete variable and no imputation was necessary.

[6] *This approach is akin to missing dummy imputation.*

The remaining three variables from the intake survey — all of which are categorical — were imputed. These variables were imputed as single variables with multiple factor levels (as opposed to a set of dummy variables). This approach ensured that imputed datasets did not include impossible combinations (for example, a student who has never completed a computer science course cannot also have completed "2 or more" computer science courses). Using the other analysis variables, the MICE package imputes values using a regression model appropriate to the scale of each imputed variable (such as binary variables imputed via logistic regression, or polytomous logistic regression for variables with > 2 factors).

Summary of the regression models for MRQ 2 and MRQ 3

Figure 5 diagrams a generalized version of the regression models used to address the correlational research questions for this study. The boxes in the diagram collectively show the full set of variables used to address the research questions. The independent variables are shown as three boxes to the left of the diagram, which are in turn enclosed within a larger box representing the full set of independent variables in the model. The box on the right shows the student achievement outcomes. The only *directed* relationships (shown as straight single-headed arrows) into the student achievement outcomes come from the set of independent variables and the error term, ε.

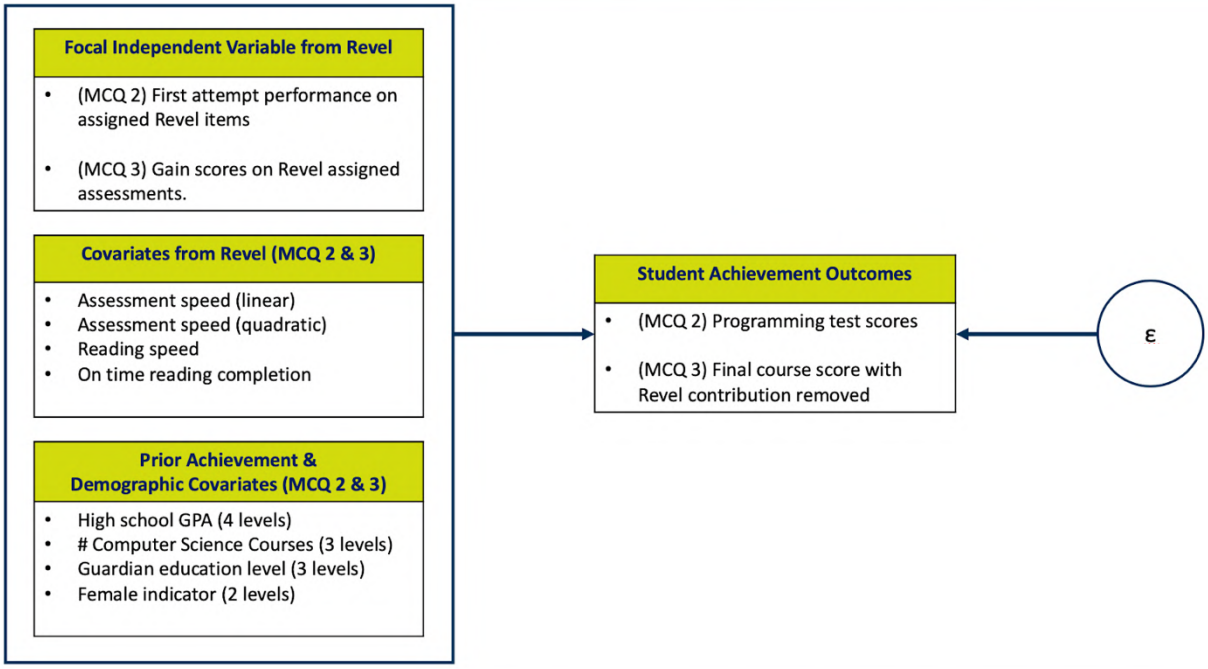

Figure 5. Generalized diagram of regression models used to address correlational research questions

Adjustment for multiple comparisons

Since two achievement-related outcomes were investigated in the present study (programming test performance and course grades), the Bonferroni correction was applied to adjust the familywise Type I error rate; the resulting Type I error rate was adjusted to $\alpha = .025$.

Pearson

Results

[MRQ 1] Do students persist on challenging programming tasks in Revel until obtaining a correct solution?

Figure 6 shows the distribution of students' persistence rates (*N* = 114) on incorrect first attempts on programming tasks. The mass of the distribution is located near the upper boundary (which is 1), indicating that students in the sample were quite persistent on programming tasks. The median persistence rate was 0.96, indicating that 50% of the students in the sample were deemed to have persisted on 96% of the tasks where they initially made an incorrect first attempt. As an indication of the high degree of persistence, 90% of students persisted on 86.8% of the tasks on which they initially made an incorrect first attempt.

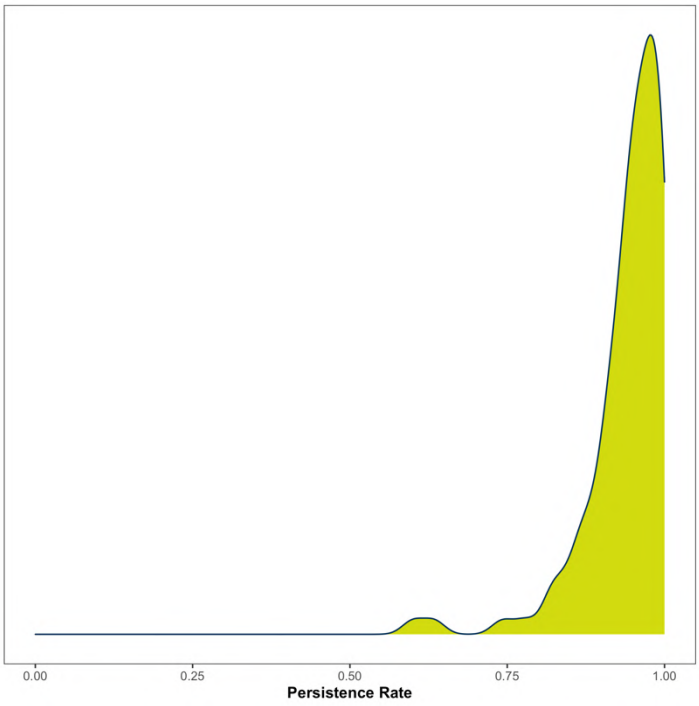

Figure 6. Distribution of persistence rates on incorrect first attempts

Task completion

In this section, we address the four secondary research questions related to students' task completion.

- Do students complete assigned readings in Revel?
- Do students complete assigned programming tasks in Revel?
- Do students complete assigned readings in Revel on time?
- Do students complete assigned programming tasks in Revel on time?

Figure 7 shows the distributions of task completion rates (shown along the horizontal axis) with the task types (Revel programming task, Revel reading task) distinguished by panels. The distributions for task completion rates at any point in the course are represented in blue; the distributions for tasks completed within the assignment window are represented in green.

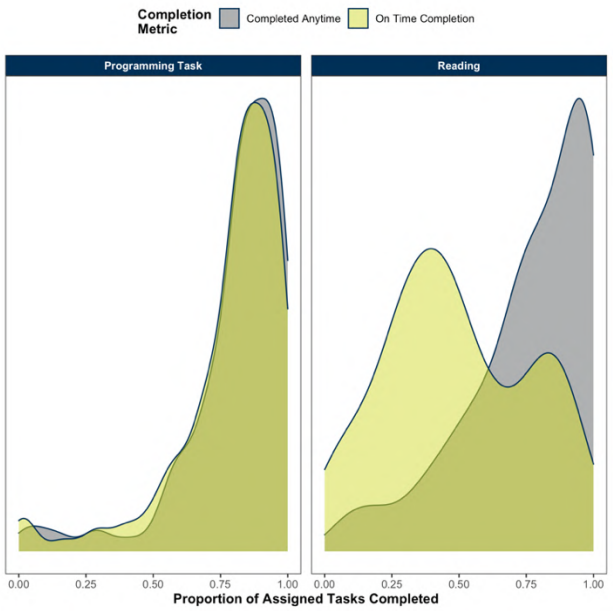

Figure 7. Distributions of programming and reading task completion proportions

For assigned programming tasks (left panel), the distribution for "on-time completion" showed a very slight shift in the negative direction (median = .834) relative to the distribution for completion at any point in the course (median = .851). The distributions were otherwise very similar with dominant modes near the upper boundary of 1.

The high completion rates for programming tasks likely reflect their contribution to students' grades in the course. To complement this perspective, reading tasks in Revel do not directly contribute to students' grades. The resulting on-time completion rates for reading tasks in Revel had an asymmetric bimodal distribution with the highest peak centered over 0.25–0.30; the shortest peak was approximately centered at 0.80.

Taken together, these results suggest that most students do complete most assigned tasks in Revel — both programming and reading — but programming tasks are more likely to be completed on time.

How do students perform on assigned Revel programming tasks?

Figure 8 shows the distribution of students' performance on Revel programming tasks. The average percentage of points earned on the first attempt was just 32.9% (median = 32.8%), reflecting the degree of challenge these tasks pose to students. One of the course instructors indicated that most errors on early attempts are simple typos or accidental omissions (such as brackets, parentheses, or end-of-line semicolons), which motivated them to allow unlimited attempts on programming tasks.

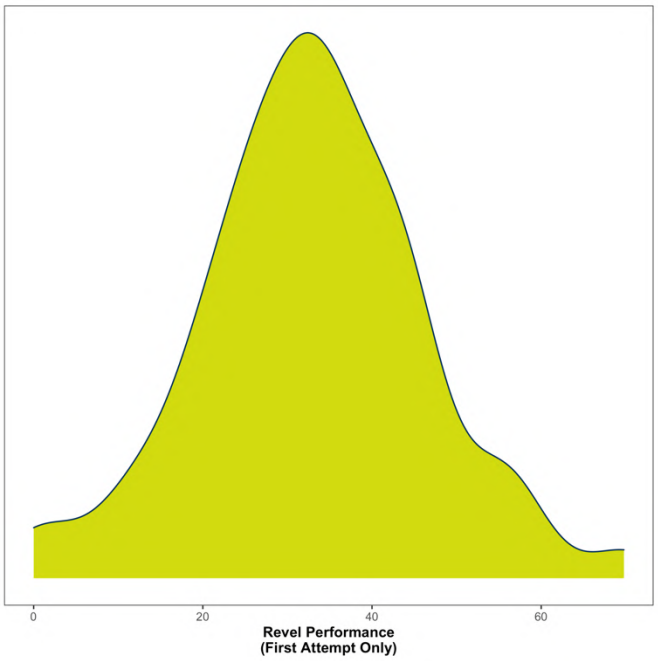

Figure 8. Distribution of first attempt performance on Revel programming tasks
Values along the horizontal axis can be interpreted as the percentage of points earned using only the first attempt on Revel programming tasks.

Based on the distribution of students' persistence rates (see above), students are keen on improving upon incorrect first attempts (the median persistence rate was 0.96). As seen in Figure 9 — which shows the distribution of students' gain scores — return attempts tend to pay off in improved scores, as evidenced by a dominant mode centered over 75 (that is, a 75% gain of possible points earned, relative to first attempts).

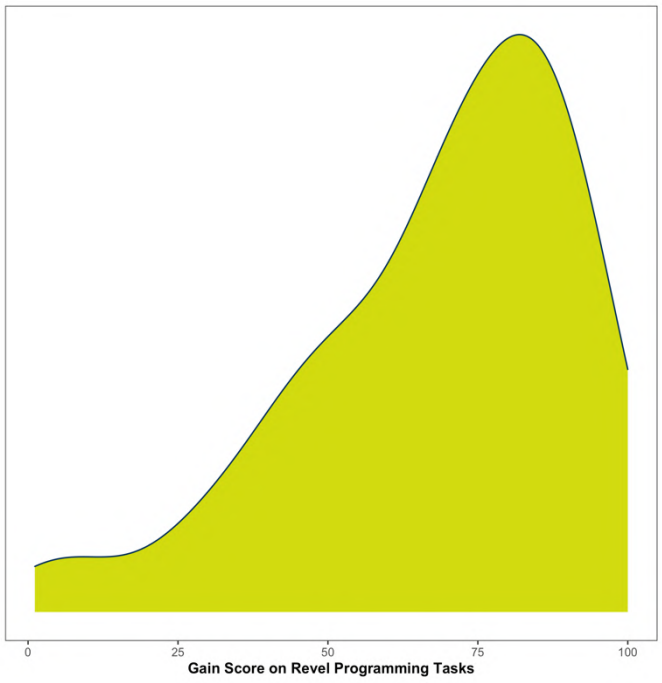

Figure 9. Distribution of gain scores on Revel programming tasks

Pearson

[MRQ 2] Are students' scores in Revel related to aggregate performance on programming tests?

As described previously, a linear regression model was developed to investigate the relationship between students' knowledge and skills in programming, using their first attempt performance in Revel as a proxy, and their performance on summative programming tests created by the instructor. Covariates were included to control for prior achievement and knowledge of programming, gender, and instructor for the course.

Table 6 shows the estimates (after pooling over the 50 imputations) of the regression coefficients along with standard errors, confidence intervals, and *p*-values for each independent variable in the model. Taken together, the full set of independent variables explained 37.4% of the variance in students' programming test scores.

To simplify interpretation, all continuous variables were grand mean centered. With the continuous variables grand mean centered, the expected programming test score is 79.87 for a student who fits the following profile:

- was a student in the Fall semester of 2019 (all other semesters = 0)
- is male (female = 0)
- had a high school GPA between 2.0 and 3.0
- has never taken a computer science course
- has a guardian with a high school education/GED or who did not graduate high school
- is exactly average on all Revel-based variables in the model

Pearson

Table 6. Pooled regression results for addressing MRQ 2 — the relationship between students' performance in Revel and their performance on summative programming tests

	b	SE_b	2.5%	97.5%	*p-value*
Intercept & Revel-based effects					
Intercept	79.87	6.91	66.14	93.59	**< .001**
First attempt performance^{MRQ}	0.49	0.17	0.15	0.82	**.005**
Programming tasks — time spent (linear)	-0.81	0.22	-1.25	-0.37	**< .001**
Programming tasks — time spent (quadratic)	-0.02	0.01	-0.05	0.01	.166
Reading — time spent	0.18	0.21	-0.23	0.60	.384
On-time reading completion	0.16	0.11	-0.06	0.38	.146
Covariates: prior achievement, prior experience, demographic					
Female	-7.25	4.04	-15.29	0.79	.077
High school GPA: Unknown GPA	2.15	6.44	-10.62	14.93	.739
High school GPA: 3.1–3.5	4.30	5.83	-7.28	15.87	.463
High school GPA: 3.6–4.0	1.90	5.06	-8.14	11.95	.708
One computer science course	-0.16	4.31	-8.72	8.40	.970
2+ computer science courses	7.57	4.90	-2.17	17.31	.126
Spring 2018	-16.94	10.01	-36.84	2.95	.094
Fall 2018	-8.09	8.34	-24.66	8.48	.335
Spring 2019	-13.72	5.68	-25.00	-2.45	**.018**
Guardian ed. level: Some college	1.31	4.31	-7.26	9.87	.762
Guardian ed. level: Bachelor's or higher	12.78	4.27	4.29	21.27	**.004**
Variance explained	**Estimate**	**2.5%**	**97.5%**		
Multiple R^2	0.37	0.23	0.51		
Residual variance	223.50				

Owing to the use of multiple imputation, all cases were retained, yielding an analytic sample size of N = 114 students. Bolded values indicate a statistically significant result. MRQ = effect associated with main research question. The significance results are identical after applying the adjustment for multiple comparisons.

Relationship between first attempt performance in Revel and programming test scores

The relationship between students' first attempt performance in Revel and performance on programming tests was of primary interest in the regression model. The results indicate that students with higher scores in Revel tend to achieve higher scores on programming tests. More specifically, students' scores on programming tests tend to increase by 0.49 (±0.17) points for every one-point increase in performance on programming tasks in Revel, on average, holding everything else in the model constant.

Consider the student who is completely average on all independent variables (described above). This student is expected to achieve a programming test score of 79.87. A one point increase in the student's Revel performance is associated with an increase from 79.87 to (79.87 + 0.49) = 80.36 on their programming test performance. A 10-point increase in the student's Revel performance is associated with an increase from 79.87 to [79.87 + (0.49 × 10 = 4.9)] = 84.77 on their programming test performance.

[MRQ 3] Is the aggregate gain from first to maximum attempts on Revel programming tasks positively related to students' course grades?

The statistical model used to address the impact of gain scores on students' overall performance in the course (when Revel performance was removed) was very similar to the model used for MCQ 2 (see above). The model included the same set of independent variables, with the exception that Revel-based gain scores became the focal independent variable instead of students' first attempt performance on Revel programming tasks.

All continuous variables were grand mean centered, rendering the estimated intercept of 78.92 to correspond to a student with the following profile:

- was a student in the Fall semester of 2019 (all other semesters = 0)
- is male (female = 0)
- had a high school GPA between 2.0 and 3.0
- has never taken a computer science course
- has a guardian with a high school education/GED or who did not graduate high school
- is exactly average on all Revel-based variables in the model

Table 7 shows all estimated effects in the regression model, inclusive of standard errors, 95% confidence intervals, and *p*-value. Taken together, the full set of independent variables explained 44.1% of the variance in students' overall course grades (with the Revel component removed).

Pearson

Table 7. Pooled regression results for addressing MRQ 3 — the relationship between Revel gain scores and students' course grades with the Revel portion removed

	b	SE_b	2.5%	97.5%	*p-value*
Intercept & Revel-based effects					
Intercept	78.92	5.18	68.63	89.21	**< .001**
Gain score^{MRQ}	0.37	0.07	0.23	0.51	**< .001**
Programming tasks — time spent (linear)	-0.07	0.16	-0.38	0.24	.662
Programming tasks — time spent (quadratic)	-0.01	0.01	-0.03	0.01	.448
Reading — time spent	0.04	0.16	-0.28	0.36	.816
On-time reading completion	0.06	0.09	-0.12	0.23	.514
Covariates: prior achievement, prior experience, demographic					
Female	-2.50	3.07	-8.60	3.60	.418
High school GPA: Unknown GPA	2.00	4.91	-7.74	11.74	.684
High school GPA: 3.1–3.5	2.74	4.40	-6.00	11.48	.535
High school GPA: 3.6–4.0	-0.60	3.91	-8.37	7.17	.878
One computer science course	-0.28	3.30	-6.85	6.28	.932
2+ computer science courses	3.67	3.74	-3.76	11.11	.329
Spring 2018	-6.63	7.62	-21.77	8.51	.387
Fall 2018	1.08	5.84	-10.51	12.68	.853
Spring 2019	-4.88	4.50	-13.82	4.05	.281
Guardian ed. level: Some college	1.09	3.27	-5.42	7.59	.740
Guardian ed. level: Bachelor's or higher	7.56	3.25	1.10	14.01	**.022**
Variance explained	**Estimate**	**2.5%**	**97.5%**		
Multiple R^2	0.44	0.30	0.57		
Residual variance	131.66				

Owing to the use of multiple imputation, all cases were retained, yielding an analytic sample size of N = 114 students. Bolded values indicate a statistically significant result. MRQ = effect associated with main research question. The significance results are identical after applying the adjustment for multiple comparisons.

Relationship between gain scores on Revel programming tasks and course grades (Revel removed)

The relationship between students' Revel-based gain scores and course grades was of primary interest in the regression model to address MRQ 3. The results indicate that students who demonstrate more improvement in their knowledge and skills in programming, as gauged by higher gain scores, tend to receive higher course grades. A one-point increase in gain scores was associated with a 0.37 (±0.07) increase in course grades, on average, holding everything else in the model constant. A 10-point increase in a student's Revel performance is associated with an increase from 78.92 to [78.92 + (0.37 × 10 = 3.7)] = 82.62 in course grades.

Pearson

Efficacy statements

Given the results of this study, we can make the following statements about the efficacy of Revel for *Introduction to Java Programming* by Y. Daniel Liang:

Timeliness and completion

- 90% of students persisted on more than 85% of the tasks on which they initially had an incorrect first attempt.

Standard of achievement

- Higher first attempt scores on Revel programming tasks are associated with higher programming test scores.
    - In particular, a 10-point increase in the student's first-attempt Revel programming task score is associated with nearly a 5-point increase on their programming test scores.
- In general, students who persisted and made higher gains on Revel programming tasks from their first attempt score to their highest score tended to earn higher final course grades.
    - In particular, a 10-point increase in the student's gain score on Revel programming tasks is associated with nearly a 4-point increase in final course grades.

Pearson

Discussion

This study sought to show the relationship between students' knowledge and skill in programming — using their performance on Revel programming tasks as a proxy — and achievement outcomes in an introductory computer science course. In pursuit of this aim, we combined data from a survey, course grades, and the Revel platform. Descriptive statistics were used to characterize students' usage and performance. Regression models were used to address the relationships of interest, whilst controlling for various usage metrics for Revel, prior achievement, prior knowledge, and demographic attributes.

[MRQ 1] Persistence on Revel programming tasks

Persistence rates on Revel programming tasks were very high in the present study. With a range between 0 and 1, the mass of the distribution was close to 1, with a median persistence rate of 0.96. This means that 50% of the students persisted on 96% of the programming tasks in which their first attempts were incorrect.

Perhaps indicative of students' motivation, the observed persistence rates are incredibly high, bearing in mind that the Revel programming tasks are challenging (the average student earns only 32.9% of the possible points on their first attempt) and demand constructed responses (that is, students are not just selecting from a set of multiple choice options).

Persistence on challenging tasks is an essential behavior to gain proficiency in programming, but persistence alone does not guarantee gains in proficiency. For any given programming task, an indication of proficiency is only evidenced when a student produces a correct response, particularly given the performance-based nature of the programming tasks in Revel. In this way, a student's persistence sets the baseline for what they can gain in proficiency with return attempts.

[MRQ 2] Revel performance and programming test scores

First attempt performance on Revel programming tasks, which we use as a proxy measure for students' knowledge and skills in programming, exhibited a significant positive relationship with performance on external, instructor-developed programming tests. That is, students who did well on Revel programming tasks on their first attempt also tended to do well on the programming tests written by the course instructors. This effect held after controlling for prior achievement (via high school GPA), prior programming experience (count of previous courses in computer science), gender, guardian education level, and variation across semesters.

Notably, the evidence available from a linear regression model is correlational and does not support statements of causality. The *mechanism* for the existence of the relationship cannot be known based on the evidence in the present study. However, given that performance was defined by students' first attempts on Revel programming tasks, possible explanations for this are that:

- Revel provides exposure to programming in Java
- Revel provides scaffolded opportunities to practice reading and writing programs in Java and receive immediate feedback
- students already have strong Java programming ability

While prior achievement (high school GPA) and programming experience (count of previous computer science courses) were controlled for in the model, the measures used in the study were not intended to fully capture the domain of programming ability specifically. As a result, the impact of prior achievement and knowledge is unlikely to have been completely removed in the model in order for us to examine the net relationship between first attempt performance in Revel and programming test scores. However, it is challenging to learn how to program, absent exposure to and practice with programming.

[MRQ 3] Revel gain scores and overall course grades

Gain scores on Revel programming tasks were significantly and positively associated with higher grades in the course. This means that students who worked towards higher (lower) gain scores also tended to achieve higher (lower) course grades. This effect held after controlling for prior achievement (via high school GPA), prior experience (count of previous courses in computer science), guardian education level, gender, and variation across semesters. Moreover, the contribution of Revel to students' grades in the course was removed, effectively removing the built-in weight of Revel.

Again, results from a linear regression model should not be interpreted as evidence of causality. The relationship between gain scores and course grades, however, should not be dismissed. In order to demonstrate performance gains over initial attempts, a student *must* have return attempts (that is, persistence) and *keep trying* until eventually obtaining a correct solution. While some return attempts may involve simple corrections (such as omissions of semicolons to end a line of code), the degree of challenge in the programming tasks (the average student only earns 32.9% of the possible points) suggests some corrections are likely to require more effort from the student. Gains in performance for complex corrections require more than persistence; such corrections require some degree of mental struggle to make the transition from non-proficient to proficient.

Limitations of the study

The highest level of evidence for the present study is correlational in nature and does not support causal interpretations. Since claims of causality were not the goal of the present research, this is less of a limitation and more of a caution to the reader. Care was taken to estimate the effects of interest, partialling out potential confounds. However, the collection of potential confounds controlled for in the current study are themselves measured imperfectly; subject to missing data; and likely a non-exhaustive accounting of potential confounds.

The participation rate among students in the study was rather low at 32.3%. It is possible that students who participated are systematically different from students who chose not to participate. However, since the data for non-participating students was inaccessible, the extent of the similarity to the participating sample is unknown, both in terms of measured and unmeasured variables.

Pearson

Generalizability of the findings

The results obtained for the present study are based on data collected from a sample of students from one institution with content delivered by two instructors. It would be unreasonable to expect the non-probability sample used in the present study to be representative of all users with respect to both measured attributes (for example, instructor-created exams, survey responses, response rate, participation rate) and unmeasured attributes (for example, regional culture, school culture).

Having said that, it is worth noting that the courses used in the present work are quite representative in terms of content coverage and assignments from Liang's *Introduction to Java Programming*.

A series of analyses were conducted to gauge the degree of similarity between the sample of courses in the present study and other courses that have used the same Revel title (see Appendix E for further details). The courses in the present study covered the first nine chapters (out of the 29 in the title). This coverage of introductory content is highly consistent with 70% of all courses (N = 321)[7] that have used the title. Among the 224 introductory-level courses, the courses in the present study were also quite typical in terms of the counts of reading (chapters and subchapters) and programming tasks (such as end-of-section programming tasks, end-of-chapter programming tasks, and total programming tasks).

Of the 224 introductory-level courses that have used the title, 113 (50.4%) allowed unlimited attempts on programming tasks — as was the case for the sample of courses in this study. The remaining 111 introductory-level courses (49.6%) restricted the number of attempts and imposed point deductions with each return attempt (the default setting in Revel at the time of the present study).

Despite the consistency of the courses used in this work with other courses that have used the title, Revel was tightly woven into the course in a way that scaffolds students from reading to programming and emphasizes active, student-centered learning. Accordingly, different results may also be expected to the extent that characteristics of the course and the *integration* of Revel differ from the courses used in this work.

[7] *The total number of courses is an estimate following the removal of courses that were likely used for the purposes of testing, demonstration, or sampling the title. In addition, we only consider courses with start and end dates in 2018 and 2019, to be consistent with the sample of courses in the present work.*

Conclusion

Revel for *Introduction to Java Programming* is designed to scaffold learners' journeys from reading about programming to writing their own programs in Java. Moreover, immediate feedback on their programs is designed to help learners persist, especially when coupled with the allowance for unlimited attempts. The courses used in the present study also motivated students' usage of Revel by making it a part of students' overall grade, and tightly integrated Revel for *Introduction to Java Programming* into the learner-centered structure of the course. Coupling these contextual features of the courses with the design features of Revel for *Introduction to Java Programming*, the results of the present study suggest:

- students are likely to complete almost all assigned Revel tasks, although programming tasks are more likely to be completed on time than reading tasks
- students with higher first attempt scores on Revel programming tasks tend to do better on programming tests
- the programming tasks in Revel for *Introduction to Java Programming* are indeed challenging, and yet students tend to persist with them after an incorrect first attempt
- students' persistence tends to pay off in the form of higher scores on the programming tasks and, in turn, these gains are associated with higher course grades

Notably, the correlational design used in this study does not allow for claims of causality. However, the results are consistent with the notion that using Revel can help students achieve more in the course, particularly to the extent that they persist on challenging programming tasks.

Implications of findings for product implementation and further research

Given the results of the present study, we recommend that instructors using Revel for *Introduction to Java Programming*:

- assign programming tasks and encourage students to complete them by making them account for a portion of students' grades
- encourage students to complete their assignments before they come to class by setting exercise due dates and not accepting late work
- give students unlimited attempts to complete these exercises without deducting points for minor typos or omissions
- let students know the exercises are challenging and that they should keep trying even if they aren't successful on their first attempt

Pearson

Instructors should also encourage students to:

- complete reading and programming tasks before class — this frees up class time for discussion
- take their time on their first attempts — the more polished the first attempt, the fewer mistakes they will need to correct on subsequent attempts
- keep trying even if they aren't successful on their first attempt (acknowledge that the programming exercises are challenging)
- contact their instructor for help if they don't see improvements in their score despite multiple attempts

Notably, the findings from this study informed an improvement to Revel for *Introduction to Java Programming*. Originally, Revel's default settings were to allow students three attempts at each programming task, with the points awarded diminishing with each unsuccessful attempt. Influenced in part by the study's findings about challenge and persistence, this will be changed to allow unlimited attempts without point deduction for incorrect attempts.

Pearson

References

Anderson, J. R., Reder, L. M., & Simon, H. A. (1999). Applications and misapplications of cognitive psychology to mathematics education. Retrieved from ERIC database. (ED439007)

Azevedo, R., & Bernard, R. M. (1995). A meta-analysis of the effects of feedback in computer-based instruction. *Journal of Educational Computing Research, 13*(2), 111–127.

Beck, J.E. (2005). Engagement tracing: using response times to model student disengagement. *In Proceedings of the 2005 conference on Artificial Intelligence in Education: Supporting learning through intelligent and socially informed technology*. IOS Press, NLD, 88–95.

Butler, D. L., & Winne, P. H. (1995). Feedback and self-regulated learning: A theoretical synthesis. *Review of Educational Research, 65*(3), 245–281.

Choi, J., & Bogucki, M. (in press). Hints, multiple attempts, and learning outcomes in a computer-based formative assessment system. *International Journal of Quantitative Research in Education*.

Clark, R. C., & Mayer, R. E. (2011). E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning (3rd ed.). San Francisco, CA: John Wiley & Sons.

Collins, L. M., and Lanza, S. T. (2010). *Latent class and latent transition analysis: With applications in the social, behavioral and health sciences*. Wiley: New York, NY.

Collins, L. M., Schafer, J. L., & Kam, C-M. (2001). A comparison of inclusive and restrictive strategies in modern missing data procedures. *Psychological Methods*, *6*, 330–351.

DiCerbo, K. E. (2015). Assessment of task persistence. In Y. Rosen, S. Ferrara, & M. Mosharraf (Eds.) *Handbook of Research on Computational Tools for Real-World Skill Development* (pp. 780–806). Hershey, PA: IGI Global.

Dihoff, R. E., Brosvic, G. M., Epstein, M. L., & Cook, M. J. (2004). Provision of feedback during preparation for academic testing: Learning is enhanced by immediate but not delayed feedback. *The Psychological Record, 54*(2), 207–231.

Duckworth, A., & Gross, J. J. (2014). Self-control and grit: Related but separable determinants of success. *Current Directions in Psychological Science, 23*(5), 319–325.

Enders, C. K. (2010). *Applied missing data analysis*. The Guilford Press: New York, NY.

Ericsson, K. A., Krampe, R. T., & Tesch-Römer, C. (1993). The role of deliberate practice in the acquisition of expert performance. *Psychological Review, 100*(3), 363.

Graham, J. W. (2003). Adding missing-data relevant variables to FIML-based structural equation models. *Structural Equation Modeling: A multidisciplinary journal*, *10*, 80–100.

Guo, P. J., Juho, K., & Rubin, R. (2014). How video production affects student engagement: An empirical study of MOOC videos. *Proceedings of the first ACM conference on Learning@ scale conference*. ACM.

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education: A literature review. *Computer Science Education*, 21(2), 135–173.

Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research*, 77, 81–112.

Herrington, J., Oliver, R., & Reeves, T. C. (2003). Patterns of engagement in authentic online learning environments. *Australasian Journal of Educational Technology, 19*(1).

Howard, W. J., Rhemtulla, M., & Little, T. D. (2015). Using principal components as auxiliary variables in missing data estimation. *Multivariate Behavioral Research, 50*(3), 285–299.

Koedinger, K. R., Corbett, A. T., & Perfetti, C. (2012). The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science, 36*(5), 757–798.

Kotovsky, K., Hayes, J. R., & Simon, H. A. (1985). Why are some problems hard? Evidence from Tower of Hanoi. *Cognitive Psychology, 17*(2), 248–294.

Kuncel, N. R., Credé, M., & Thomas, L. L. (2005). The validity of self-reported grade point averages, class ranks, and test scores: A meta-analysis and review of the literature. *Review of Educational Research*, 75(1): 63–82.

Muthén, L. K., & Muthén, B. O. (1998–2017). *Mplus user's guide* (8th ed.). Los Angeles: Muthén & Muthén. Nathan, M. J., & Sawyer, R. K. (2014). Foundations of the learning sciences. In *The Cambridge handbook of the learning sciences* (pp. 21–43).

Nyland, K. L., Asparouhov, T., & Muthén, B. O. (2007). Deciding on the number of classes in latent class analysis and growth mixture modeling: A Monte Carlo simulation study. *Structural Equation Modeling: A multidisciplinary journal*, 14(4), 535–569.

Plummer, M. (2017). *JAGS Version 4.3.0 user manual*. Retrieved from https://web.sgh.waw.pl/~atoroj/ekonometria_bayesowska/jags_user_manual.pdf

Puntambekar, S., & Hubscher, R. (2005). Tools for scaffolding students in a complex learning environment: What have we gained and what have we missed?. *Educational Psychologist*, 40(1), 1–12.

Reiher, R. R, & Dembo, M. H. (1984). Changing academic task persistence through a self-instructional attribution training program. *Contemporary Educational Psychology*, 9, 84–94.

Ruthig, J. C., Perry, R. P., Hall, N. C., & Hladkyj, S. (2004). Optimism and attributional retraining: Longitudinal effects on academic achievement, test anxiety, and voluntary course withdrawal in college students. *Journal of Applied Social Psychology*, 34(4), 709–730.

Rubin, D. B. (1996). Multiple imputation after 18+ years. *Journal of the American Statistical Association*, *91*, 473–489.

Rushkin, I., Chuang, I., & Tingley, D. (2019). Modelling and using response times in online courses. *Journal of Learning Analytics*, 6(3), 76–89.
Shafer, J. L., & Graham, J. W. (2002). Missing data: Our view of the state of the art. *Psychological Methods*, 7, 147–177.

Shute, V. J. (2008). Focus on formative feedback. *Review of Educational Research, 78*(1), 153–189.
Thomas, A., & Pashley, B. (1982). Effects of classroom training on LD students' task persistence and attributions. *Learning Disability Quarterly, 5*(2), 133–144.

van der Linden, W. J. (2006). A lognormal model for response times on test items. *Journal of Education and Behavioral Statistics*, 31(2), 181–204.

What Works Clearinghouse (March, 2016). Applying the What Works Clearinghouse standards to postsecondary research. Washington, DC: Department of Education, Institute of Education Sciences.
Wright, G. B. (2011). Student-centered learning in higher education. *International Journal of Teaching and Learning in Higher Education*, 23(3), 92–97.

Yarbro, J., & Ventura, M. (2018). *Skills for today: What we know about teaching and assessing self-management.* London: Pearson.

Pearson

Appendix A. Data cleaning process

Table A.1 traces the steps taken to arrive at the analytic sample. The exclusionary criteria included:

- the removal of students who could not (owing to age) or did not consent to participation in the study
- absence of data in at least one data source used in the study

The approach used for handling missing data allows for missing covariates and outcome variables, and accordingly, no data were excluded because of missing values.

Table A.1. Data filtering steps for obtaining the analytic sample

Step	N
Sample if all students consented, as estimated by count of distinct student identifiers in Revel	387
Sample after excluding students younger than 18 or who did not consent to participation in the study	126
Sample after excluding students who could not be found in all data sources containing a course and student identifier (i.e., Revel platform data, institutional data)[8]	125
Sample after excluding students who withdrew from the course	114

[8] *The intake survey was not included as part of this initial join since students in 2018 were never given the opportunity to complete it. After obtaining the list of common students across all Revel platform data and institutional data, we then joined that list to the intake survey data, yielding 110 students.*

Pearson

Appendix B.
Alignment table and summary of research questions

Table B.1. Alignment between learner outcomes and main research questions

Outcome category	Outcome	Research question	Measures/metrics	Type of efficacy statement
Timeliness and completion	Learners persist through coursework	To what extent do students persist on challenging programming tasks in Revel?	% of incorrect first attempts in which the student either (a) eventually obtained partial or maximum credit or (b) made at least two return attempts without producing the correct answer (aggregated over programming tasks for each student).	Descriptive
Standard of achievement	Learners achieve competency in subject matter	Are students' scores on programming tasks in Revel related to their aggregate scores on programming tests?	% of points earned on first attempts for Revel programming tasks. The denominator is the total points possible across all assigned Revel programming tasks. Summative programming test scores from the instructors' records for the course.	Correlational
Standard of achievement	Learners are able to demonstrate skill acquisition	Are students' aggregate gain scores on programming tasks (difference in maximum earned points and points earned on first attempts) in Revel related to their course grade?	$G_i = \frac{Realized\ Gain}{Possible\ Gain} = \frac{\left(\sum_{j=1}^{J} U_{ij}^{Max} - \sum_{j=1}^{J} U_{ij}^{First}\right)}{\left(\sum_{j=1}^{J} U_{ij}^{Possible} - \sum_{j=1}^{J} U_{ij}^{First}\right)}$ where for a given student *i* on programming task *j*, U^{Max} is maximum earned points; U^{First} is the points earned on the first attempt; and $U^{Possible}$ is the number of points possible. Course grades from the instructors' records, computed with the Revel contribution removed.	Correlational

Pearson

Table B.2. Alignment between learner outcomes and secondary research questions

Outcome category	Outcome	Research question	Measures/metrics	Type of result
Timeliness and completion	Learners persist through coursework	To what extent do students complete assigned readings in Revel?	% of assigned reading completed by students	Descriptive
		To what extent do students complete assigned programming tasks in Revel?	% of assigned programming tasks completed by students	Descriptive
	Learners persist through coursework	To what extent do students complete assigned readings in Revel on time?	% of readings completed on or before the assigned due date	Descriptive
		To what extent do students complete assigned programming tasks in Revel on time?	% of programming tasks completed on or before the assigned due date	Descriptive
Standard of achievement	Learners are able to demonstrate skill acquisition	How do students perform on Revel programming tasks?	% of points earned on programming tasks using only the first attempt	Descriptive

Pearson

Appendix C. Intake survey

Thank you for using REVEL! Our goal at Pearson is to help more learners learn more. We do that through our products, and you gave us that opportunity by using REVEL. Feedback from our customers is the engine that drives us forward in pursuit of that goal. With that, we at Pearson would greatly appreciate about 5 minutes of your time to tell us about yourself as a student. Any identifying information will be kept confidential and will be used exclusively for research purposes.

1. Please tell us a little about yourself.

First Name:	
Last Name:	
Email address (the one you used to register for REVEL):	
Age:	☐ 18–24 ☐ 25–29 ☐ 30–34 ☐ 35–39 ☐ 40–44 ☐ 45–49 ☐ 50–54 ☐ 55–59 ☐ 60–64 ☐ 65+
Gender:	☐ Female ☐ Male ☐ Other
Type of student: (Select all that apply)	☐ First generation ☐ Returning ☐ Non-degree obtaining, personal interest ☐ Non-degree obtaining, professional advancement ☐ Transfer ☐ High school ☐ Full time ☐ Part time ☐ Merit-based scholarship ☐ Need-based scholarship ☐ Student loan ☐ Work sponsored

Prior achievement: (Leave blank if not applicable)	High school GPA: ☐ 0–1.5 ☐ 1.6–2.0 ☐ 2.1–2.5 ☐ 2.6–3.0 ☐ 3.1–3.5 ☐ 3.6–4.0 ☐ I do not remember ACT (if applicable): _____ SAT (if applicable): _____
Major(s): (Say 'Undeclared' if you haven't chosen a major)	
Number of computer science courses, including high school (select one):	☐ 0 ☐ 1 ☐ 2 ☐ 3+
Highest education level of primary guardian:	☐ Did not finish high school ☐ High school diploma or GED ☐ Some college ☐ Associate's degree ☐ Bachelor's degree ☐ Master's degree ☐ Doctoral or professional degree ☐ Other, please specify:
English is my first language	☐ Yes ☐ No
If not, what age did you start learning to read English?	

Pearson

2. Please rate how strongly you agree or disagree with the following statements.

	Strongly disagree	Moderately disagree	Slightly disagree	Neither disagree nor agree	Slightly agree	Moderately agree	Strongly agree
I am comfortable with topics in computer science.	1	2	3	4	5	6	7
I am comfortable with using technology.	1	2	3	4	5	6	7
I expect to do well in this class.	1	2	3	4	5	6	7
I think what we are studying in this class is interesting.	1	2	3	4	5	6	7
I want to learn as much as possible in this class.	1	2	3	4	5	6	7
I can apply what we are learning in this class to real life.	1	2	3	4	5	6	7
It is important for me to do well compared to other students in this class.	1	2	3	4	5	6	7

Thank you for completing this survey!

Pearson

Appendix D. Estimating scores on time factor via the lognormal model for response times

Problems with aggregation methods of raw response times

At the event level (a single window of time in which the student loads and unloads the task), the amount of time a student spends on any one task is not particularly reliable. There are a wide range of reasons that contribute to suboptimal reliability, some of which are student driven (for example, the student became distracted after opening a task) while others are the result of factors outside of the student's control (for example, a weak internet connection).

To reduce the impact of poor reliability for any one event, students' response times (RTs) are frequently aggregated in some way. Common aggregations include total time or some first order point estimate (such as mean or median) for each student's RT distribution.

Unfortunately, total time on task accumulates unreliable measures (often leading to distributions that are highly skewed in the positive direction), which in turn biases average RT on programming tasks, typically in the positive direction. While the median is less subject to extreme values, it oversimplifies a student's tendency of response behavior via a point estimate that ignores much of the evidence in available data.

Fitting the lognormal model for response times

The data for fitting the model included an $N \times J$ matrix of log response times. For the present study, students' response times were first aggregated over all tasks (and attempts for assessment tasks) for each assigned assessment and reading before computing the log response times, yielding J = 86 assessments and J = 112 tasks, respectively. The following model was in turn applied separately for assessment and reading tasks.
The model was fit to the data via Markov Chain Monte Carlo with Gibbs sampling as implemented in JAGS (Plummer, 2003). The likelihood for the data was specified as follows:

$$\begin{aligned} \widehat{RT}_{ij} &= \alpha(\beta_j - \theta_i), \\ RT_{ij} &= N(\widehat{RT}_{ij}, \tau) \end{aligned} \qquad \text{(D1)}$$

whereα, β_j, and θ_i retain the same meaning as described above. Owing to the large number of tasks with a small sample, the discrimination parameter was fixed to $\alpha = 1$for all J tasks. The additional parameter appearing in Equation D1 is dispersion, τ, which is on a precision scale in JAGS rather than the more familiar variance (which in turn is the inverse of precision, $1/\tau$).

The following prior distributions (using the precision metric) were assumed for all unknown model parameters (β_j, θ_i, τ):

$$\begin{gathered}\beta_j \sim N(0,.01), \text{ for } j \text{ in } 1{:}J \\ \theta_i \sim N(0,.01), \text{ for } i \text{ in } 1{:}I \\ \tau \sim G(.01,.01)\end{gathered} \tag{D2}$$

The model is not identified in that the probability distributions are the same via the transformations $\beta_j - \epsilon$ and $\theta_i - \epsilon$ (Rushkin, et al., 2019; van der Linden, 2006). The approach recommended in the methodological literature is to restrict the set of $\theta_i$values to sum to zero (see van der Linden, 2006 for further details):

$$\sum_{i=1}^{N} \theta_i = 0, \tag{D3}$$

which was achieved by first drawing values of $\theta_i$from a prior distribution and then grand mean centering those values against the mean of the drawn values, $(\theta_i - \bar{\theta})$.

Three chains were initialized with dispersed starting values and ran for 2,000 iterations with no thinning. Although all model parameters rapidly converged, the first 1,000 iterations were discarded and treated as burn-in iterations, leaving 3,000 = (1,000 iterations ×3 chains) draws from the posterior distribution. Posterior means were computed (separately for assessment and reading) for use as students' time factor score.

JAGS code

```
model {
#- Data likelihood.
for(i in 1:N){
 for(j in 1:J){
  RT_hat[i,j] <- alpha*(beta[j] - theta[i])
  RT[i,j] ~ dnorm(RT_hat[i,j], prec)
 } # close loop over tasks.
} #- close loop over students.

 #- Fix alpha to 1 for all tasks.
 #- Can be tested more easily with larger samples.
 alpha <- 1

 #- Priors for task time parameters.
 for(j in 1:J){
  beta[j] ~ dnorm(0, .01)
 }

 #- Priors for person time parameter.
 for(i in 1:N){
  theta1[i] ~ dnorm(0, .01)
  theta[i] <- theta1[i] - mean(theta1[])
 }

 #- Prior for precision.
 prec ~ dgamma(.01, .01)
 std_res <- sqrt(1/prec)
}
```

Pearson

Appendix E. Analyses to assess generalizability of Revel-based metrics

This section presents a series of results intended to characterize the degree of generalizability of the sample data to the total population of courses that have used Revel for *Introduction to Java Programming* by Y. Daniel Liang. More specifically, this section addresses the following questions:

1. To what extent is the coverage of content from Revel for *Introduction to Java Programming* in the present study similar to that of other courses that have used the same title?
2. In terms of Revel-based metrics of implementation, how similar are the courses in the present study to other courses that have used Revel for *Introduction to Java Programming*?

Data preparation

The data for the analysis of generalizability were obtained from Pearson databases in the Spring 2020 semester, following the completion of data collection for the present study. To be consistent with the courses in the current study, we only retained courses with start and end dates in 2018 and 2019. The following filters were additionally applied to arrive at set of course identifiers that were likely associated with real courses:

- remove "courses" with titles including words indicative of use for testing purposes (such as partial or exact matches to "demo", "trial", "test", or "sample")
- remove courses shorter than five weeks or longer than 36 weeks, based on the start and end dates provided by the instructor/user
- remove courses with only one student (all student identifiers associated with the course had to be found across all tables containing a course and student identifier before obtaining a count of distinct students)
- remove courses that could not be found across tables containing a course identifier

The analytic sample consisted of 321 courses, inclusive of the four courses in the present work.

Content coverage profiles

The initial data used to identify profiles for content coverage consisted of one row for each course (N = 321) and one column for each chapter (J = 29) in Liang's *Introduction to Java Programming*. Each value in the data indicated whether the chapter was assigned in the course (1 = assigned, 0 = not assigned). The analytical goal was to identify profiles of courses that were qualitatively distinct with respect to content coverage, at least as measured by the assignment of chapters from the title. Latent class models (LCA; Collins & Lanza, 2010) were pursued to this end, in addition to obtaining an estimate of the percentage of courses consistent with each profile.

One of the most challenging tasks in applications of LCA is deciding on the number of latent classes to extract. Before fitting any models, descriptive and graphical approaches were pursued to gain some insight into the number of latent classes that could be extracted. Figure E.1 shows the percentage of courses (shown along the horizontal axis) that used each of the 29 chapters (shown along the vertical axis) in Liang's *Introduction to Java Programming*.

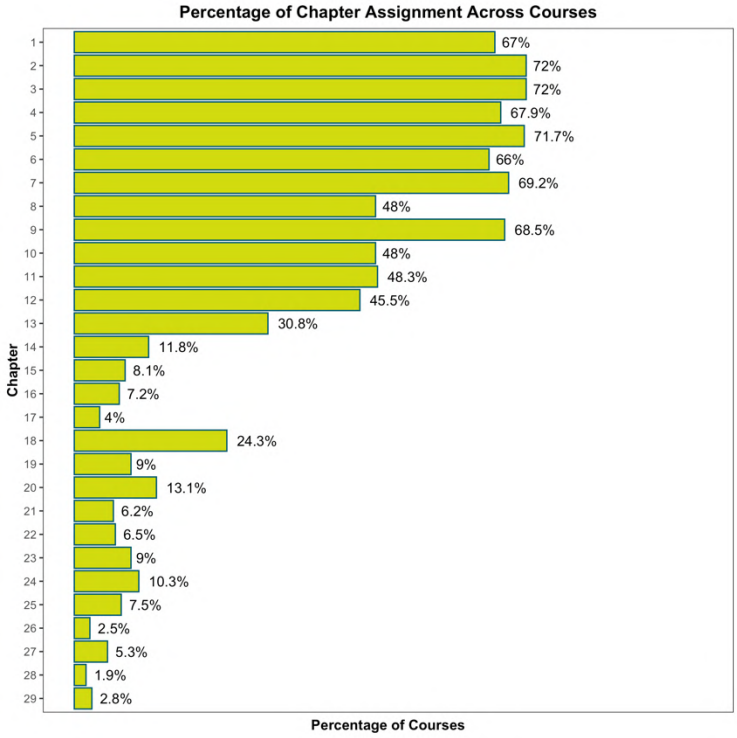

Figure E.1. Percentage of courses that assigned each chapter in Liang's Introduction to Java Programming

The results indicate that the first seven chapters and Chapter 9 were the most likely to be assigned by instructors. Chapter 8 and Chapters 10 through 12 were also assigned in nearly half of all courses. With some exceptions, chapters from the second half of the title are rather unlikely to be assigned. To reduce the potential for estimation issues, chapters used by fewer than 5% of all courses were excluded from subsequent analyses.

Figure E.2 shows a heatmap of the bivariate proportions of chapter assignments across courses. Chapters from the title are shown along both axes with chapter numbers increasing moving up along the vertical axis and moving to the right along the horizontal axis. The gradient in the figure corresponds to the proportion of courses, with values becoming more gray as the proportion tends towards zero and more green as the proportion tends towards one.

The key interest for this figure is the *patterns* of joint assignment of chapters rather than the particular values of the proportions. More specifically, the patterns of greatest interest are distinct clusters of joint chapter assignment. These clusters are seen as bright spots in the heatmap, and provide rudimentary insight into the number of profiles that could be extracted with the use of LCA.

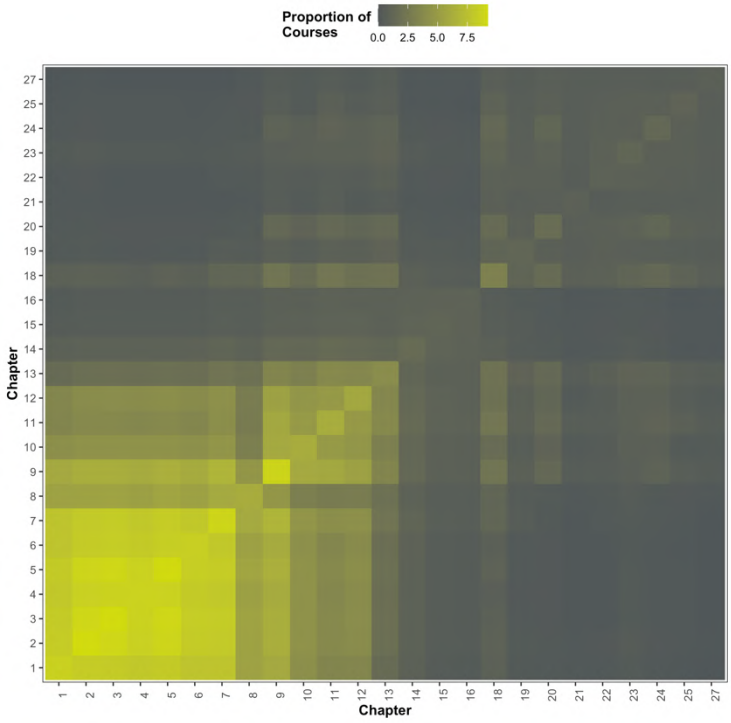

Figure E.2. Bivariate proportions of chapter assignments across courses that have used Liang's Introduction to Java Programming

As evidenced by the bright spot in the bottom left portion of the heatmap, there is one dominant cluster reflecting assignment of the first seven to nine chapters. While an additional cluster was observed for Chapters 9–13, there is no way to determine if this cluster reflects a distinct set of courses from those that assign Chapters 1–7. Although relatively less bright, a third cluster was also observed for Chapters 18–27. Taken together, these results suggest up to three latent classes, with one dominant class reflecting usage of the first seven to nine chapters.

The next step was to fit a series of latent class models, with one additional class extracted with each successive model. The models were fit to the data from 100 random starting values to reduce the risk of selecting a model at a local maximum (Collins & Lanza, 2010). In following this process, the only model to achieve convergence was a model with two latent classes.

The parametric bootstrapped likelihood ratio test (Nylund, Asparouhov, & Muthén, 2007) was in turn conducted with 100 replications to more rigorously test whether the second class was needed. The result was statistically significant ($p < .001$), suggesting that the two-class model fits the data better than a one-class model. Given this, we report on the two-class model, with each latent class representing a content profile.

Figure E.3 shows the profile of content coverage for the two latent classes, with the conditional probability of a chapter being assigned shown along the vertical axis and chapter numbers shown along the horizontal axis. Each line represents one of the two content profiles, with Profile 1 (n = 224, 69.8%) represented by the blue line and Profile 2 (n = 97, 30.2%) represented by the red line.

The largest distinction between the two content profiles was observed for Chapters 1–8, with the conditional probabilities being high for courses in Profile 1 and low for courses in Profile 2. The conditional probabilities for Chapters 9–27 fell precipitously for Profile 1 and generally became modestly higher for Profile 2. These results suggest that there is a dominant profile of introductory-level courses (Profile 1) and a second profile of more advanced courses (Profile 2).

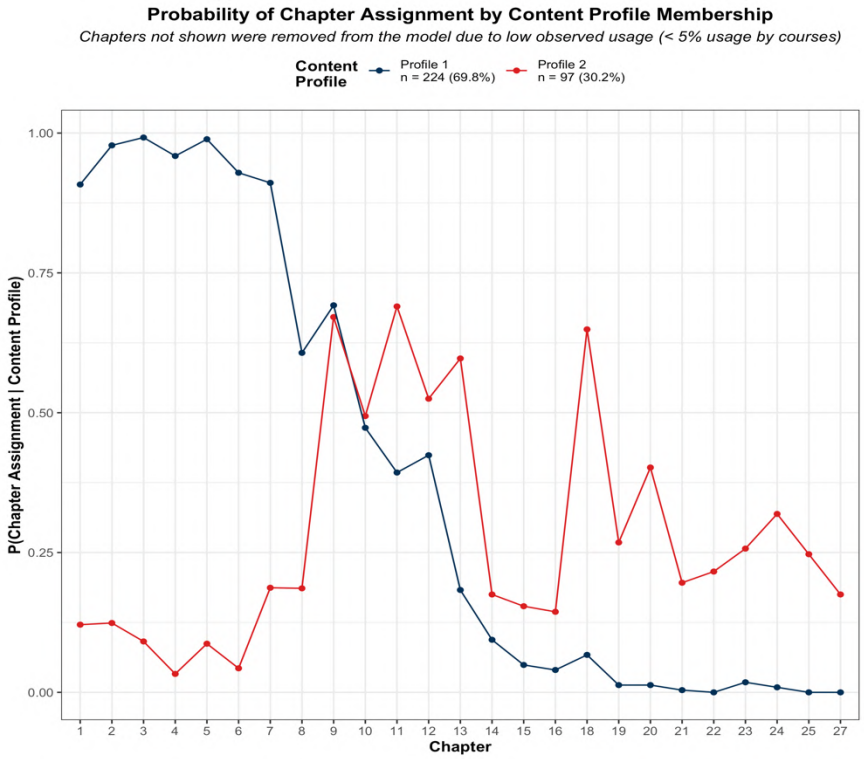

Figure E.3. Latent class profiles of assigned content for courses that have used Liang's Introduction to Java Programming

Profile 1 is consistent with using the title for introductory courses. Profile 2 is consistent with more advanced courses.

Implementation of assignments

The results from the two-class model described above served as the starting point for addressing the degree of similarity between the sample of courses in the present study and other courses that have used the title. All of the courses in the present study were classified into Profile 1, and accordingly were in the majority of courses (*n* = 224, 69.8%) with respect to content coverage. To ensure comparability, the other 220 courses were leveraged to construct reference distributions for each aspect of implementation. The aspects of implementation considered included:

- count of end-of-section programming tasks assigned
- count of end-of-chapter programming tasks assigned
- count of total programming tasks assigned (that is, end-of-section + end-of-chapter programming tasks)
- count of chapters assigned
- count of chapter sections assigned
- length of the course in weeks

The similarity of the courses in the present study to the reference courses was assessed by locating the median value on each of these features within the corresponding reference distribution. Figure E.4 shows these results, with implementation features represented by each panel; reference lines are shown for the courses in the present study (blue dashed line) and reference courses (red dashed line).

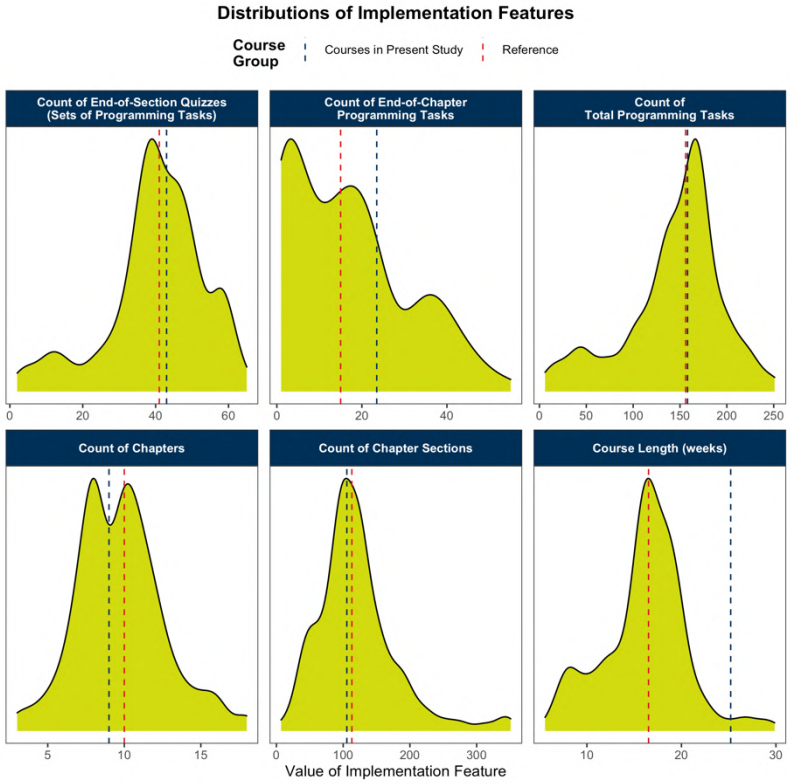

Figure E.4. Distributions of implementation features for introductory courses

The blue dashed reference line in each panel represents the median for the set of courses used in the present study. The red dashed reference line in each panel represents the median for the reference set of courses.

Pearson

Generally stated, the courses in the present study appear very similar to other introductory-level courses that have used the title. For all assigned features (counts of programming tasks and readings from the e-text), the median values for the courses in the current study were situated near the center of the corresponding reference distributions.

The largest difference was observed for the length of the course. The peak of the distribution for course length was located around 16 weeks, which is consistent with courses that are one semester long. The courses used in the present study were also one semester (16 weeks) in length. The aberration was the result of entering an end date in the Revel platform that was beyond the end date of the course.

An additional aspect of the implementation of Revel is the number of allowable attempts at programming tasks. The default setting at the time of the present study allows for three attempts, with penalties assigned for return attempts. The courses in the present study allowed unlimited attempts on the programming tasks without penalties for return attempts. As seen in Figure E.5, introductory-level courses (*n* = 224) are evenly divided in the implementation of attempts, with 111 (49.6%) courses using the default setting and 113 (50.4%) courses allowing unlimited attempts.

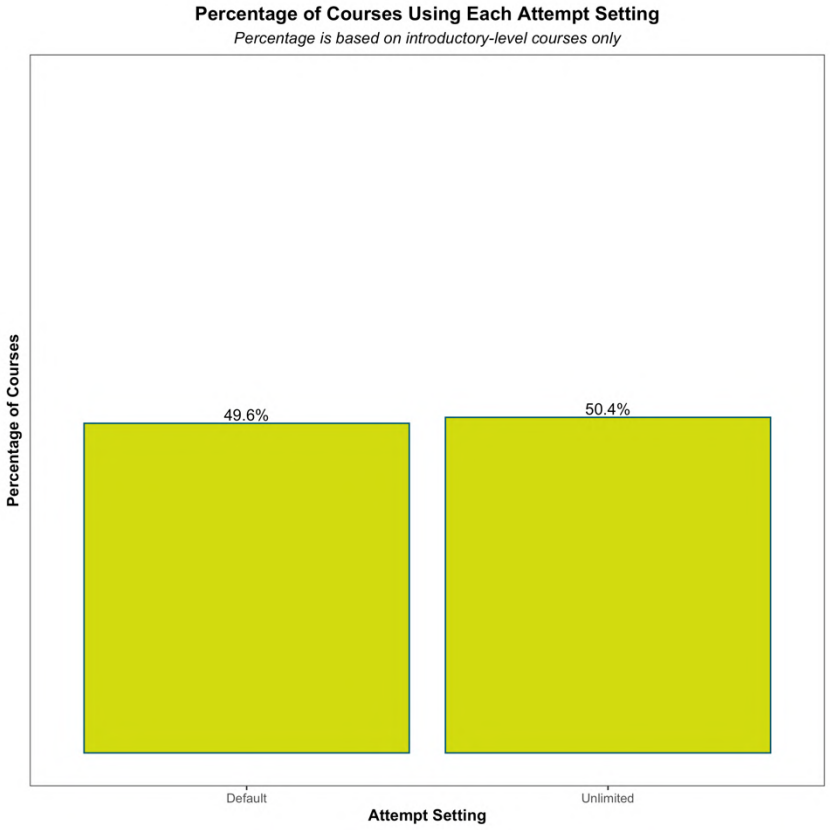

Figure E.5. Distribution of attempt implementation among introductory-level courses that have used Liang's Introduction to Java Programming