Practicum: a scalable online system for faded worked examples in CS1

Aaron Bauer
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, Washington

Eleanor O'Rourke
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, Washington

Kyle Thayer
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, Washington

Eric Butler
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, Washington

Whitaker Brand
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, Washington

Stuart Reges
Paul G. Allen School of Computer Science & Engineering
University of Washington
Seattle, Washington

ABSTRACT

Universities are experiencing record enrollments in CS1 courses and are struggling to keep up with demand while also offering a quality experience to an increasingly diverse population. Scalable, online tools will be an important part of meeting this challenge. Current tools available to CS1 students often provide very limited feedback. Faded worked examples are a proven pedagogical technique that can provide richer feedback, but there there are many open questions about how best to deliver them in an online system. Furthermore, few empirical studies explore the design and impact of these types of tools in CS1 courses. In this technical report, we present *Practicum*, a suite of new online tools that provide step-by-step explanations of CS1 concepts and gradually fade these explanations over a series of problems. Our system exploits the procedural nature of these concepts to synthesize explanations from example problems. We studied the impact of these tools by deploying *Practicum* in a large CS1 course with over 1000 students. Though a controlled study in the CS1 lab sections found that *Practicum* did not improve performance compared to existing materials, students who used *Practicum* on their own did perform better on the exams. These students were more likely to be female, have less programming experience, and plan on a non-computer science major. These results demonstrate *Practicum* has the potential to improve learning in a CS1 course, and is accessible to less traditional CS1 students, making it a promising vehicle for for further exploration into the design of interactive tools for CS1.

CCS CONCEPTS

• **Social and professional topics** → **CS1**; • **Applied computing** → *Interactive learning environments*;

KEYWORDS

faded worked examples; automated tools; CS1; learning outcomes

1 INTRODUCTION

Introductory computer science courses at many universities face both strained resources due to increasing enrollments and a continuing need to attract and retain a more diverse population of students. New scalable online tools to support CS1 learning have the potential to help address both of these concerns. Unfortunately, the current tools available to CS1 students often only provide very limited feedback to the student, such as whether their solution to a practice problem is correct.

A large body of research supports using worked examples as an effective pedagogical technique. A worked example consists of an example problem and step-by-step instructions leading to the solution. They have been shown to improve learning in a variety of domains, including LISP programming [20] and statistics [16]. Worked examples have been shown to be even more effective when the transition between viewing demonstrations and independent problem solving is gradually faded [16]. Faded worked examples are gaining interest among computer science education researchers [18], especially in the context of introductory courses (e.g., [6]).

Though researchers have identified faded worked examples as an effective technique, there is much the research community does not know about optimal ways to deliver them in an online system. It is challenging to hand-author worked examples and appropriate fading mechanisms, which makes it difficult for researchers to answer these questions empirically. It would be valuable to have an automated and fully parameterized system in order to facilitate the exploration of a variety of design choices such as choice of explanations, the progression of examples, and how explanations are faded away. As a first step toward this goal, we present *Practicum*, an online system capable of synthesizing step-by-step explanations for example problems, and automatically fading these explanations over a series of problems.

Practicum builds upon O'Rourke et al.'s framework for automatically generating interactive tutorials for procedural knowledge [13]. O'Rourke et al. applied this approach to math problems such as subtraction, but did not extend it to more complex domains or study its use in the wild. We have applied this technique to three types of problems present on our institution's CS1 exams and studied our system's impact when integrated into CS1.

We deployed *Practicum* in a CS1 course at our institution with over 1000 students in order to gain an initial understanding of how this type of tool impacts student performance. The research questions to be addressed were: (1) Does the scaffolded and more interactive practice available in *Practicum* lead to better performance compared to the existing lab materials? and (2) How do students

who use *Practicum* on their own perform compared to those who do not?

We ran a controlled experiment in the official lab sections comparing students who used *Practicum* to students who used the existing, limited feedback materials. We also made the system available to all students, and studied how those who used it compared to those who didn't. While *Practicum* did not improve lab quiz scores relative to existing materials, students who used it on their own performed better on the types of exam questions targeted by our system. Overall, *Practicum* shows promise for improving learning in CS1 in situations where one-on-one human tutoring is not available, and we believe it can be used as a testbed to study questions about the design of interactive tools for CS1 more deeply in the future.

2 RELATED WORK

2.1 Worked Examples

Worked examples have been the subject of substantial study over the past few decades. When it comes to their use in computer science in particular, however, existing work is far less comprehensive. Skudder and Luxton-Reilly's review of the computer science education literature found there has been little research into worked examples in computer science [18], particularly when it comes to formal studies. One exception includes Margulieux et al.'s experiments with worked examples in Android App Inventor, focusing on the effect of sub-goal labeling [10]. Morrison et al. further studied the interaction of subgoal labels and worked examples when applied to a text-based programming language [12], finding that the effectiveness of worked examples could be improved by including subgoal labels.

We depart from this previous work in that we studied our educational intervention fully integrated into a CS1 course. This is an important perspective, and one that is underrepresented in the literature. This kind of work is necessary to make an effective case for the adoption of these techniques, as it demonstrates their immediate applicability.

A key issue to consider in the design of any instructional materials is the demand placed on a learner's working memory by the learning tasks, known as *cognitive load* [21]. Since working memory is limited, *Cognitive Load Theory* holds that learning suffers when the total working memory requirements exceed a learner's limited capacity [15]. Thus, it is incumbent on instructional systems to direct students' attention toward those aspects of the material important for learning and away from extraneous aspects. Educational research has shown that failing to sufficiently focus attention on appropriate features can interfere with recall and transfer [4].

Hence, we relied upon the known best practices for worked examples in our design of *Practicum*. Based on the principles described by Atkinson et al. for effective worked examples, we avoided the *split attention* effect by keeping the individual steps of our problem-solving procedures small and focused on a single part of the UI [1]. We also incorporated subgoal labeling by including steps in our solving procedures that provide descriptions of the current subgoal, such as, when the problem includes executing a conditional branch, displaying "Let's find the first branch that evaluates to true" before going into the specific steps to determine which branch will execute. Multiple studies on worked examples for computer science have found that providing students with subgoal labels improves learning outcomes [11, 12]. Furthermore, areas for specific tasks are visually separated and labeled, including a *variable bank* for tracking and updating the values of local variables. We did not follow Atkinson et al.'s recommendation to pair examples and practice problems, as later research has shown faded worked examples effectively combine examples and practice [16].

2.2 Tools for CS1

A tremendous number of tools have been developed for computer science education. In the informal taxonomy of such tools used by Pears et al., our system would most naturally fall under *visualization tools* [14]. *Practicum* would not count as a *generic* visualization system like those surveyed by Sorva et al. since it visualizes specific types of problems rather than arbitrary Java programs [19]. Unlike many of the tools discussed in those reviews, or others such as JSVEE [17], our system goes beyond visualization, and provides interactivity and faded step-by-step explanations. A recent generic visualization system, Online Python Tutor, exemplifies this difference [7]. While both it and *Practicum* step through code line-by-line, visualizing the program state at each step, the difference is one of breadth versus depth. Online Python Tutor offers the ability to visualize a much wider variety of programs, whereas *Practicum* offers a more interactive and scaffolded visualization of a narrower set of programs. Kumar has generated step-by-step explanations of code execution and expression evaluation, but the explanations are not faded and focus on building students' mental model of general execution semantics rather than teaching a specific problem-solving procedure [8, 9].

Intelligent tutoring systems are another type of system developed for introductory programming (e.g., [2, 5]). These systems tend to focus on responding intelligently to user mistakes, whereas our system's focus is providing the step-by-step explanations necessary for faded worked examples.

2.3 Automated Instructional Scaffolding

Practicum builds upon O'Rourke et al.'s framework for automated faded worked examples [13]. Under this framework, a designer chooses a problem domain and describes the thought process to solve these problems as a procedure or *thought process algorithm* (TPA). An interpreter steps through this algorithm line-by-line, generating an explanation for each line using a set of mappings between programming language constructs and UI elements provided by the designer. In this way, explanations can be generated for any problem the procedure can solve. User interactions can be similarly generated, prompting a student for input at key points.

With *Practicum*, we make a substantial extension to this previous work. First, we extend this approach to the more complex domain of CS1 concepts. Problem visualization and user interactions are not tied to a grid, as they were for the previous applications, and some of our problem solving procedures must model mentally simulating Java code. Second, the previous evaluation of this framework did not fully integrate its implementations into an ongoing course.

3 PRACTICUM

Practicum currently provides faded worked examples for three types of problems in Java: (1) evaluating pure expressions, (2) determining the results of code snippets containing `if/else` constructs, and (3) determining the results of code snippets containing simple array operations inside a `for` loop. Instructors reported that students struggled with these problems on the CS1 exams at our institution. All three of these problem types can be solved by following an imperative procedure, so they are amenable to automation by O'Rourke et al.'s framework.

All three problem types have a constrained format and are designed to test specific skills. We based our development of the thought process algorithms (TPAs) on the problem-solving strategies our CS1 TAs teach students to use. *Practicum* automatically generates all parts of the worked examples based on the example problem and the TPA, including the visualized problem state, the highlighting of relevant UI elements, and the explanation text. In developing the TPAs, our goal was to help students become familiar with the behavior of certain classes of Java programs. Specifically, we wanted to take the problem-solving strategies CS1 TAs teach students to use on these problems and extract those into procedures. These strategies are intended to be instructive and accessible. Furthermore, we wanted to teach procedures that students could perform on paper to facilitate transfer to the in-class exams. An important feature of the design of *Practicum*, however, is its modularity with respect to a specific TPA. To alter the way the system explains a type of problem, an educational technologist can modify or replace the corresponding TPA. While the UI might require modification to support the new TPA, the underlying architecture is agnostic.

3.1 Problem Types

In an **expression problem**, the user is asked to evaluate a pure expression. An expression is represented as a collection of operators and operands. *Practicum* currently covers arithmetic (including both integer and floating point division), modulo, and string concatenation. See Figure 1 for an example of an expression problem in progress.

Our expression problem TPA consists of two phases. In the first phase, the student resolves all multiplication, division, and modulo operators. They're asked if there are any such operators in the current expression, and then, if there are, to select the leftmost such operator and its operands. They compute the result of applying the operator, creating a new expression. Once all multiplication, division, and modulo operators have been resolved, a second phase begins where all addition and subtraction operators are resolved in the same manner. Prompts remind users about integer division, string concatenation, as well as the behavior of modulo.

An **if/else problem** consists of a Java method definition, and a corresponding method call. See Figure 1 for an example of an if/else problem in progress. The method definition takes one or more `int`s, contains several `if/else` constructs, as well as lines that modify a local variable, and ends by printing several of the method's local variables. The user is asked to determine what is printed out as a result of a particular method call. Aside from the code itself (with visual cues for the current line, previously executed lines, and lines not executed due to conditional branching), a *variable bank* shows the current values of all local variables.

The TPA we created for if/else problems is based on mentally simulating the Java code line-by-line. All lines, with the exception of the `println` at the end of the problem, are an `if`, an `else`, or modify a local variable. In the case of the latter, the student updates the variable bank to reflect the new value. When a conditional construct is encountered, the student evaluates one or more conditions and determines which branch will execute. If there is a branch (or possibly more than one) that won't be executed as a result, the student crosses out the lines that will be skipped. The crossing out of lines is a technique CS1 TAs at our institution will suggest to students to help them keep track of what has already been done.

Array problems are similar to if/else problems in many ways. They too consist of a method definition and call, and maintain a variable bank displaying local variables. See Figure 1 for an example of an array problem in progress. The method always takes one array of `int`s, and contains a `for` loop that performs some operation over the array. The CS1 TAs at our institution reported students' errors often stemmed from mistaking the array length or forgetting about zero-indexing, so the variable bank displays the length of the array as a separate variable and displays the array indices above the array values. The solution to an array problem is the final array values.

Like if/else, the array problem TPA is based on simulating the code line-by-line. Steps include evaluating the loop condition, updating array elements, and incrementing the loop variable. There are some places where the TPA calls attention to specific pieces of data, like explicitly recording the array indices and the array length. The TPA also goes into detail when resolving array references as resolving array values can be difficult for CS1 students. It highlights the array location being referenced, and then replaces the reference with the value at that location.

3.2 Fading

Practicum supports four levels of fading, designed to provide a smooth transition from a fully explained example to an example the user completes entirely independently. Fading level one is a full demonstration of the example problem with no user input required. Every step in the TPA is fully explained, and the correctly solution shown. Fading level two introduces interactivity. At every appropriate step in the TPA, the user is prompted to provide the correct answer for that step. The prompt explains what input the system is expecting. For example, in an expressions problem, the user will be asked to select the next operator that will be resolved, and then to select its left and right operands. In an if/else problem or an array problem, the user will be asked what line will execute next, as well as the result of a conditional, whether part of an `if` or a `for` loop. If a user enters an incorrect answer three times, the correct answer is displayed and the user is allowed to proceed.

Fading level three removes some of the scaffolding. All the steps of the TPA are still included, but in most cases, instead of telling the user what input is expected, the system instead prompts them to "Try the next step on your own!" If the user gives an incorrect response, the prompt explaining what input is expected is shown. Fading level four further removes scaffolding by skipping all non-interactive TPA steps. At this level, the user proceeds through the

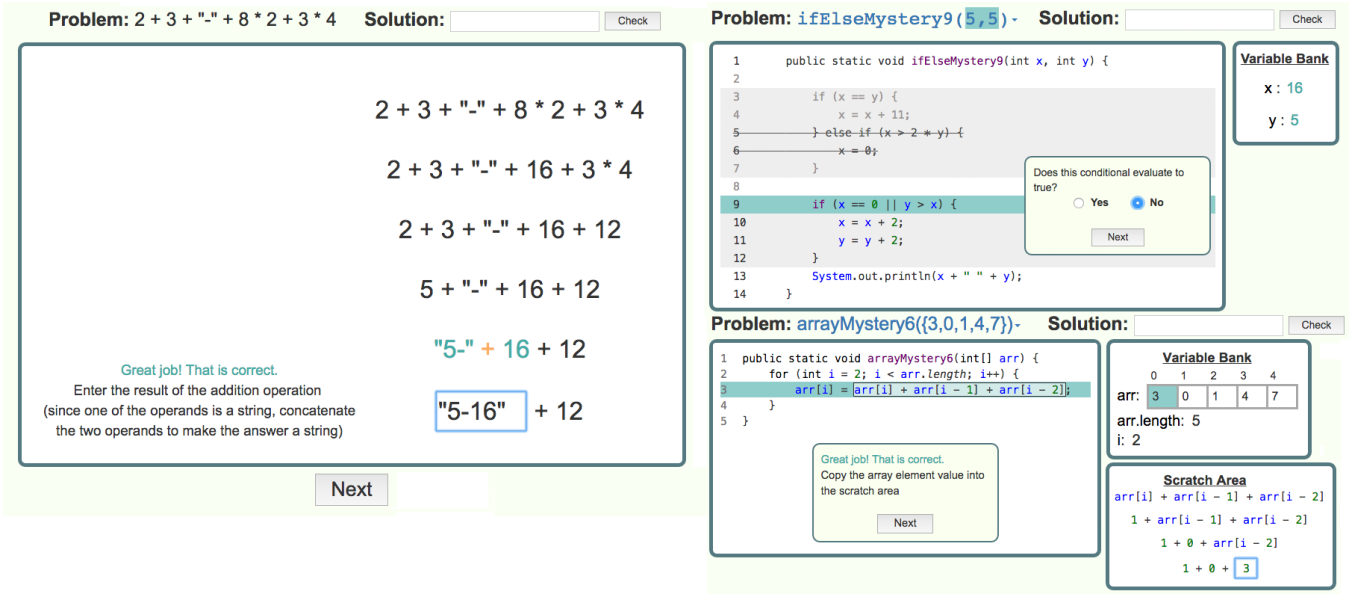

Figure 1: Clockwise from left, the interfaces for an expression problem, an if/else problem, and an array problem. In the expression problem, the task is to evaluate the expression. Each line corresponds to evaluating one operator, starting at the top. The current operator and operands are highlighted and the prompt explaining the current step is shown to the left. In the if/else problem, the task is to determine what is output be the `println`. The *Problem* menu at the top allows the user to select method calls with different arguments. The central box displays the prompt and the method declaration, highlights the current line, and crosses out the lines of conditional branches that didn't execute. The *Variable Bank* displays the values of all local variables. In the array problem, the task is to determine the final values of the array. The primary difference of this interface from if/else is the inclusion of a *Scratch Area*, which is used to diagram the steps involved in resolving references to array elements. In all three problem interfaces, the user can enter a final answer in the *Solution* box at the top at any time.

problem essentially unaided, receiving explanations in a just-in-time fashion (i.e., only in response to an incorrect step). This final level is designed to closely mimic the process of following the solving procedure on paper.

The properties of an optimal fading progression in this context are not well understood, and thus it is important the mechanism for triggering a transition between these fading levels is a parameter of *Practicum*. For the deployment described below, we used a fixed progression based on the number of problems a user had attempted. The first problem would be fully explained (first fading level), and the next three would prompt the user for input at each step (second fading level). The fifth and sixth problems would be at the third fading level with some scaffolding removed and every problem from the seventh onward would be at the final fading level. This progression was tracked separately for each problem type (i.e., regardless of how many expression problems a student has done, their first if/else problem is fully explained).

4 METHODOLOGY

To assess whether it could be successfully integrated into a real-world setting and to investigate its impact on learning outcomes, we deployed *Practicum* in a large CS1 course. Integration into a real-world setting was necessary to demonstrate the viability of the system, particularly with respect to its potential use in future work on the optimal design choices for such a system. We as a community don't have a great understanding of how to optimally design a faded progression worked examples for computer science, so our goal was not to evaluate the impact of our particular design in *Practicum*. Instead, this deployment was intended to capture the impact of *Practicum* holistically, including the ways in which it was used and the effects of that use on the opportunities for transfer occurring as a natural part of the course. Integration into a real-world setting was necessary to demonstrate the viability of the system, particularly with respect to its potential use in future work on exploring design choices for such a system. The instructor of the course was a researcher involved in the development of *Practicum*.

4.1 Design

Our deployment consisted of two concurrent studies, one experimental and one observational. The experimental study took place in three weeks of the official lab sections of the course, which are optional, for-credit opportunities for students enrolled in the main CS1 course. Each section consists of about 40 students and meets in a computer lab for 50 minutes once per week. Students work through an interactive slide deck of review material and practice problems related to the most recent lectures, and TAs are present to offer one-on-one help. Three specific weeks were chosen because each of them focused on one of the three problem types included in *Practicum*. The study was a between-subjects design with two conditions, with half of the lab sections randomly assigned to each

condition. In the control condition, students worked through the interactive slide deck used in previous offerings of the course. The slide deck's interactive feedback is limited to whether a given answer is correct or not. In the experimental condition, the practice problems on the slides were replaced with a link to a special version of *Practicum* that provided a faded progression of example problems with detailed interactive explanations, but otherwise identical to those in the slide deck. In both conditions, students completed a short quiz at the end of the lab.

The observational study took place over the entire course. After the corresponding lab was complete, problems of that type on *Practicum* were made available to the course at large as an optional study tool. Course staff informed students that they could use *Practicum* to study for some of the problems on the course exams.

4.2 Participants

Participants were students enrolled in a CS1 course at a public university in the northwestern United States. We made *Practicum* available to all students in the course. See Table 1 for a breakdown of the different subgroups.

Subgroup	Size	Gender
All enrolled students	1098	34% female
Enrolled in a lab section	392	38% female
Experimental condition	184	35% female
Control condition	208	40% female
Practicum users not enrolled in a lab	293	34% female

Table 1: Participant demographics

4.3 Procedure

We collected the following data as part of our deployment: (1) detailed usage data for *Practicum*, including the problems each student completed and the UI interactions involved, (2) course data including lab attendance, scores on end-of-lab quizzes, and exam scores broken down by question, and (3) responses to a survey sent to all students at the start of the course, which included questions about choice of major and previous programming experience. It is important to note that measuring the number of problems a student completed required some judgment. *Practicum* allows a user to move to a different problem at any point, so students may leave off working through a problem as soon as they feel they understand what's going on rather than continuing to the final step. Hence, for our analysis we consider a problem to have been completed if the student either (1) entered a correct answer in the solution box, or (2) worked through more than half of the steps in the problem.

5 RESULTS

We describe the results from our experimental and observational studies comparing the learning outcomes for different groups of students. Throughout this analysis we use non-parametric Mann-Whitney U tests with $\alpha = .05$ confidence, as our data is not normally distributed. For each test, we report the test statistic U, the two-tailed significance p, and the rank-biserial correlation measure of effect size r.

5.1 Experimental

To address our research question "Does the scaffolded and more interactive practice available in *Practicum* lead to better performance compared to the existing lab materials?", we compared the quiz scores of students in the experimental condition to those of the control condition for each problem type. The quiz questions were the same type of problems students practiced during the lab (either in *Practicum* or in the slide deck). Overall, this analysis shows *Practicum* had little impact on quiz scores relative to the existing lab materials. In the labs covering expression and array problems, we did not find a statistically significant difference in quiz scores between the two conditions. In the lab covering if/else problems, we found a small, but statistically significant difference in quiz scores in favor of the control condition.The relevant data and statistical quantities are given in Table 2.

We also found significant differences in the quiz scores between different labs. We found the expression problem quiz had significantly lower scores than the if/else problem quiz ($U = 53862.5$, $p < .0001$, $r = -.19$), a mean score of 83% and median score of 100% ($n = 367$) compared to a mean score of 93% and median score of 100% ($n = 361$). We also found the expression problem quiz had significantly higher scores than the array problem quiz ($U = 51614.5$, $p = .0010$, $r = .13$), a mean score of 83% and median score of 100% ($n = 367$) compared to a mean score of 71% and median score of 100% ($n = 324$). We believe this indicates the relative difficulty of these problem types for students in our study, with if/else being the easiest and array being the most difficult.

5.2 Observational

We investigated how different groups of students used *Practicum* and how that use related to performance on exam questions demonstrating transfer (i.e., questions of the same type as those available in *Practicum*). An important preliminary result was that students enrolled in the lab sections outperformed their peers on exams. Students enrolled in lab had significantly higher midterm exam scores than those not enrolled in lab ($U = 114046.5$, $p < .0001$, $r = .17$), a mean score of 82% and median score of 84% ($n = 391$) compared to a mean score of 78% and median score of 81% ($n = 701$). Similarly, on the final exam students enrolled in lab had significantly higher scores than those not enrolled in lab ($U = 102525.5$, $p = .0094$, $r = .099$), a mean score of 74% and median score of 78% ($n = 359$) compared to a mean score of 70% and median score of 75.5% ($n = 634$). Since this effect is strong enough to potentially hide any impact of *Practicum*, we excluded students enrolled in a lab from our analysis of *Practicum*'s effect on exam performance.

5.2.1 Impact on Exam Performance. To assess *Practicum*'s effect on a specific problem type, we compared the exam scores of students who had completed a full faded progression (at least seven problems) for that problem type to those of students who did not complete any problems of that type in *Practicum*. Focusing on those students who completed a full progression is intended to capture the potential of *Practicum*.

Lab's problem type	U	p	r	Condition	Mean score	Median score	n
Expression	15820	.27	.06	Experimental	85%	100%	179
				Control	82%	100%	188
If/else	14340.5	.02	−.11	Experimental	91%	100%	166
				Control	95%	100%	195
Array	12855	.78	−.02	Experimental	70%	100%	152
				Control	72%	100%	172

Table 2: The results of comparing quiz scores of the experimental and control conditions in each of the three labs. The mean quiz score, median quiz score, and number of students attending that lab are given for both conditions.

The impact of *Practicum* on exam question scores varied with problem difficulty. For if/else, the easiest problem type, we found no significant difference in scores on the midterm exam if/else question between students completing a full faded progression and students who completed no if/else problems ($U = 11361.5$, $p = .91$, $r = .0069$), a mean score of 98% and median score of 100% ($n = 44$) compared to a mean score of 97% and median score of 100% ($n = 520$). For the problem type of medium difficulty, expression, we found no significant difference in scores on the midterm exam expression question between students completing a full faded progression and students who completed no expression problems, although the effect size was considerably larger than for if/else ($U = 15726$, $p = .10$, $r = .12$), a mean score of 85% and median score of 90% ($n = 66$) compared to a mean score of 81% and median score of 80% ($n = 541$). The increased effect size and significance of *Practicum*'s impact on expression scores compared to if/else scores fits a trend of greater impact on more difficult problems.

Coming to the most difficult problem type, we found a significant difference in scores on the final exam array question. Students completing a full faded progression scored higher than those who completed no array problems ($U = 7795.5$, $p = .041$, $r = .18$), a mean score of 92% and median score of 100% ($n = 38$) compared to a mean score of 80% and median score of 100% ($n = 499$). This variation with problem difficulty suggests a scaffolded progression like *Practicum*'s may be most effective for more complex or difficult CS1 concepts. While further study is needed before this can be concluded with certainty, it is a valuable insight for the future design of *Practicum* and similar systems.

One concern about the result for array problems was that it captured a difference in overall exam performance, rather than *Practicum*'s impact on learning for that specific concept. To assess this, we compared the scores on the remaining final exam questions (i.e., those not involved in the above comparison) for the same populations. If no significant difference were found in overall exam performance, we could feel confident that the difference in performance on the array question could not be entirely explained by a difference in the amount of preparation for the exam, but instead indicated a difference in the effectiveness of that preparation (i.e., the use of *Practicum*). Indeed, we found no significant difference in the scores on the other final exam questions between students who completed a full faded progression of array problems and those who completed no array problems ($U = 8420$, $p = .25$, $r = .11$), a mean score of 73% and a median score of 81% ($n = 38$) compared to a mean score of 70% and a median score of 75% ($n = 499$).

We were very interested in the potential of *Practicum* to help is those students who are struggling and who might benefit from additional scaffolded practice. Unfortunately, no formative assessment was conducted that would have enabled us to identify such students. Instead, we used overall exam score as proxy. Specifically, we looked at the population of students whose total score on the remaining exam questions (i.e., those not corresponding to a problem type in *Practicum*) fell in the bottom third among all students in the course. We found that repeating the above comparisons on this population showed larger positive effect sizes across all three problem types, but failed to reach statistical significance. This is likely because a smaller number of students fell into these groups, reducing the power of our statistical analysis. While we need to confirm with future studies, the increased effect sizes seem to indicate struggling students benefit more strongly from using *Practicum*. The statistical results are given in the Table 3.

5.2.2 Demographics. We were also interested in who was using *Practicum*. In particular, if the students using and potentially benefiting from *Practicum* were disproportionately male, computer science majors, or those with previous formal programming experience, then there would be reason for concern that *Practicum* would fail to be accessible to a diverse student population. To investigate this we integrated the results of a survey sent to all students at the beginning of the course with our usage data for *Practicum*. This survey was designed by the course staff and is used in every offering of the course. The survey included questions about intended major and previous programming experience.

Overall, the students who used *Practicum* were more likely to be female, have fewer years of formal programming experience, and intend to major in something other than computer science. In term of gender, 34% of the students enrolled in the course were female, compared to 36% of the students who completed at least one problem of any type on *Practicum*. Of the 820 students who responded to the survey, 56% reported having no formal programming experience and 47% said they had decided on or were considering majors other than computer science or computer engineering. For survey respondents who completed at least one problem of any type on *Practicum*, a higher percentage reported no formal programming experience (60%) and planned on majors outside computer science (56.0%). These data indicate that *Practicum* was accessible to a population at least as diverse as course overall along the measures we collected.

				users			non-users		
Problem	U	p	r	Mean score	Median score	n	Mean score	Median score	n
Expression	1964	.12	.19	78%	80%	24	71%	80%	203
If/else	1324.5	.23	.14	98%	100%	16	94%	100%	192
Array	689.5	.055	.32	93%	100%	12	71%	90%	168

Table 3: The comparison of exam scores of low-scoring students who completed a full progression for a problem type (users column) to those of low-scoring students who completed no problems of that type (non-users column).

6 DISCUSSION

The results from our observational study demonstrate the exciting potential for *Practicum* to provide effective, scalable support for CS1 concepts in settings where one-on-one human tutoring is not available. The increased learning gains made by *Practicum* users for more difficult problems are of particular interest, as they suggest a potentially fruitful focus for the application and future development of our system. It will likely continue to be difficult to have any impact for concepts like if/else, where the mean exam question scores were above 90%, even for lower-performing students. Our study also took place *in the wild*, fully integrated into an existing CS1 course. While this perhaps led to noisier and less comprehensive data, we believe it lends *Practicum* credibility as a practical and immediately applicable system and presents a more complete picture of how *Practicum* might function in hands of real students.

The increased effect sizes for lower-performing students are encouraging. Along with the user demographics, this gives us confidence that *Practicum* is able to serve a broad range of students rather than potentially reinforcing existing inequality in CS1 courses.

While the results of our observational study were promising in terms of *Practicum*'s effect on learning outcomes, the results of our experimental study were not. Though we do not have any data that would allow us to establish this with any certainty, the presence of human tutors (i.e., TAs) may have overwhelmed any benefit resulting from using *Practicum* instead of the existing materials. This would not be surprising as one-on-one tutoring has been shown to be among the most effective educational interventions [3]. Lab students' higher scores on exams are suggestive of this. We also have some anecdotal evidence to support this explanation. Lab TAs reported that when a student would get stuck on a problem, instead of relying on the explanations provided by *Practicum*, the student would raise their hand and have a TA walk them through the problem. In this way, the experimental condition may have resembled the control condition more closely than intended. We are not attempting to compare *Practicum* to human tutoring, but rather we are asking what impact *Practicum* has in the absence of human tutoring (i.e., students using the tool voluntarily outside a structured environment like a lab section). This is especially relevant as rapidly increasing CS1 enrollments mean a resource intensive intervention like one-on-one tutoring may not always be feasible.

Human tutors would not explain, however, the statistically significant deficit of the experimental condition for if/else problems. It possible that a lack of time on task contributed to this. In the if/else lab, students spent significantly less time completing problems in *Practicum* than in the expression lab ($U = 7854$, $p < .0001$, $r = -.47$), a mean of 429 seconds and median of 215 seconds ($n = 166$) compared to a mean of 590 seconds and a median of 548 seconds ($n = 179$). A comparison between the if/else lab and the array found a similar difference ($U = 6598.5$, $p < .0001$, $r = -.48$), with students in the array spending a mean of 779 seconds and median of 677 seconds completing problems ($n = 152$). Perhaps students found the problems too easy and quickly moved on, as the experimental condition mean quiz score was above 90%. Another possibility is the deficit of the experimental condition reveals a flaw in our design of the thought process algorithm for if/else problems. In either case, both of our studies indicate that for systems like *Practicum*, targeting concepts that students already tend to master is counterproductive or at least a poor investment in terms of impact.

The fading progression used by all three problem types may have limited the impact of *Practicum* in both the experimental and observational study. In general, different types of problems took different amounts of time to work through. This meant that completing the full faded progression might take a student much longer to do for the array problem type than the expression problem type, and may have resulted in fewer students completing this progression for longer problems. Indeed, while 152 students in the expression lab completed a full progression, only 16 students and 10 students completed a full progression in the if/else and array labs, respectively. Different progressions for each problem type may have been more suitable, or an adaptive progression like those in intelligent tutoring systems that responds to student progress. Furthermore, *Practicum* did not give students explicit feedback on their current fading level, nor given them direct control over it. The effect of fading progressions and related mechanisms is among the many remaining questions about the optimal design of instructional systems like *Practicum*.

The mechanism of transfer within *Practicum* also merits rigorous investigation. Our design of the system and our analysis in this work both posit the completion of a full fading progression as the primary mechanism, but controlled experimentation is needed to empirically verify this. We also know very little about the motivation or attitude of students using *Practicum* and their behavior outside the system. Surveys and interviews could fill this gap, and perhaps enable us to better address flaws in *Practicum*'s design and adapt it to suit how students want to use it.

We have identified the following limitations of our studies that may impair their generalizability. First, our positive results stem from an observational rather than experimental study. This inevitably makes the results vulnerable to selection bias since using *Practicum* was optional. We have no way of knowing if the students who appeared to benefit from using *Practicum* would have achieved similar scores without it, but the relatively large population and

lack of significant difference on exam questions not targeted by *Practicum* suggest we found a real impact attributable to our system.

A second limitation is that our data comes from a single CS1 course. Replicating these studies in future offerings of the course, as well as at other institutions is needed to ensure the validity of this work. Furthermore, we had very limited opportunities to observe learning transfer. With only a single exam question for each concept, our results are potentially highly susceptible to variations in the properties of that question and related factors. Furthermore, the exam questions, as well as the lab quiz questions, were relatively easy for the average student as shown by the high median scores. Incorporating some kind of formative assessment into *Practicum* may provide more robust measurement without disrupting the course. Despite these limitations, we believe this work makes a valuable contribution to the empirical computing education literature, and will help facilitate future research on the design of CS1 educational technology.

Finally, we lack data on the use of alternative methods of practicing the concepts targeted by *Practicum*. It is possible there is some other activity highly correlated with using *Practicum* that is responsible for the results we found. While this seems unlikely, detailed surveys or interviews would be necessary to definitively discount it.

7 CONCLUSIONS & FUTURE WORK

We make several important contributions in this work. We present *Practicum*, a new suite of online tools for faded worked examples in CS1. This system can automatically synthesize explanations from example problems, and automatically fade these explanations away over a series of problems. In developing *Practicum*, we extended the procedural knowledge framework of O'Rourke et al. to a new, more complex domain. Finally, we conducted a large-scale evaluation of *Practicum*'s impact on learning in a CS1 course.

This evaluation is promising, and indicates *Practicum* has the potential to improve learning in CS1 in situations where one-on-one tutoring is not available, particularly for lower-performing students and on more difficult concepts. The demographics of the users of *Practicum* suggest it can support a diverse population of CS1 students. *Practicum* will be available online for anyone to use.

Practicum represents a unique combination of pedagogical approach and technical infrastructure. Its fully parameterized automated generation of faded progressions of worked examples offers an opportunity to explore a little-understood design space, and investigate optimal ways to deliver this kinds of educational content at scale.

There are many avenues for future work. Empirical studies are needed to investigate how systems like *Practicum* can optimally structure faded progressions of worked examples. This includes questions of how this structure should adapt in response to student behavior or problem domain. More broadly, there are many parts of the design of systems like *Practicum* that merit further study, including choice of explanations and feedback to student mistakes.

It would also be valuable to gather data on the effects of *Practicum* on student motivation as well as other data about students' response to using the system. A more complete picture of how tools like *Practicum* fit into students' practice in CS1 courses is needed to better tailor its design and understand its impact. Finally, extending *Practicum* to other types of problems in CS1 would broaden its applicability as well as help refine the findings presented in this work about its interaction with problem difficulty.

REFERENCES

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

[2] PL Brusilovsky. 1992. Intelligent tutor, environment and manual for introductory programming. *Educational and Training Technology International* 29, 1 (1992), 26–34.

[3] Michelene TH Chi, Stephanie A Siler, Heisawn Jeong, Takashi Yamauchi, and Robert G Hausmann. 2001. Learning from human tutoring. *Cognitive Science* 25, 4 (2001), 471–533.

[4] National Research Council et al. 2000. *How people learn: Brain, mind, experience, and school: Expanded edition.* National Academies Press.

[5] Davide Fossati, Barbara Di Eugenio, Christopher W Brown, Stellan Ohlsson, David G Cosejo, and Lin Chen. 2009. Supporting computer science curriculum: Exploring and learning linked lists with iList. *Transactions on Learning Technologies* (2009).

[6] Simon Gray, Caroline St Clair, Richard James, and Jerry Mead. 2007. Suggestions for graduated exposure to programming concepts using fading worked examples. In *Proceedings of the third international workshop on Computing education research.* ACM, 99–110.

[7] Philip J Guo. 2013. Online python tutor: embeddable web-based program visualization for cs education. In *ACM technical symposium on Computer science education.* ACM.

[8] Amruth N Kumar. 2006. Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors. *Technology, Instruction, Cognition and Learning* (2006).

[9] Amruth N Kumar. 2015. The effectiveness of visualization for learning expression evaluation. In *ACM Technical Symposium on Computer Science Education.* ACM, 362–367.

[10] Lauren E Margulieux, Mark Guzdial, and Richard Catrambone. 2012. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *international computing education research conference.* ACM, 71–78.

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

[12] Briana B. Morrison, Lauren E. Margulieux, and Mark Guzdial. 2015. Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. In *International Computing Education Research Conference (ICER '15).* ACM, 9. https://doi.org/10.1145/2787622.2787733

[13] Eleanor O'Rourke, Erik Andersen, Sumit Gulwani, and Zoran Popović. 2015. A Framework for Automatically Generating Interactive Instructional Scaffolding. In *ACM Conference on Human Factors in Computing Systems.* ACM.

[14] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin* 39, 4 (2007), 204–223.

[15] Jan L Plass, Roxana Moreno, and Roland Brünken. 2010. *Cognitive load theory.* Cambridge University Press.

[16] Alexander Renkl, Robert K Atkinson, Uwe H Maier, and Richard Staley. 2002. From example study to problem solving: Smooth transitions help learning. *The Journal of Experimental Education* 70, 4 (2002), 293–315.

[17] Teemu Sirkiä. 2013. A JavaScript library for visualizing program execution. In *Koli Calling.* ACM.

[18] Ben Skudder and Andrew Luxton-Reilly. 2014. Worked examples in computer science. In *Australasian Computing Education Conference.* Australian Computer Society.

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

[20] J.G. Trafton and J. Reiser. 1993. *The Contributions of Studying Examples and Solving Problems to Skill Acquisition.* Technical Report. Naval HCI Research Lab.

[21] Tamara van Gog and Fred Paas. 2012. Cognitive load measurement. In *Encyclopedia of the Sciences of Learning.* Springer, 599–601.