Implications for the Design of Tutorial Systems for Teaching Program Comprehension: An Empirical User Study

Kathryn Anne Bridson
IMPLICATIONS FOR THE DESIGN OF TUTORIAL SYSTEMS FOR TEACHING PROGRAM COMPREHENSION: AN EMPIRICAL USER STUDY

by

Kathryn Bridson

A Thesis
Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Major: Computer Science

The University of Memphis
May 2020
ABSTRACT

Bridson, Kathryn M.S. Implications for the Design of Tutorial Systems for Teaching Program Comprehension: An Empirical User Study. Major Professor: Dr. Scott D. Fleming

This paper reports the results of an observational user study of a tutorial system for learning to comprehend Java programs. The study involved 9 undergraduate college students enrolled in introductory programming courses. For the study, we built a new system, Coding Companion, with a design generally representative of prior educational systems for learning program comprehension, but that incorporates several novel design decisions for addressing information overload and split-attention effect. Key findings include the following. Participant feedback and usage data suggest that the system was generally helpful for learning and that the lesson interface was engaging and not overly confusing. All nine participants were strongly positive about quizzes provided by the system, and many found answer explanations helpful, even when they had answered the question correctly. Participant comments revealed a tension between information overload caused by long, complex program traces and the importance of completely explaining the traces, especially for beginners.
# TABLE OF CONTENTS

List of Tables v

List of Figures vi

1 Introduction 1

2 Background and Related Work 4

2.1 Systems That Teach Program Comprehension with Program Tracing 4

2.2 Potential Problems with Prior Systems 6

2.2.1 Cognitive Overload 6

2.2.2 Expertise Reversal Effect 7

2.2.3 Split-Attention Effect 7

3 Coding Companion Tutorial System for Program Comprehension 8

3.1 Design Decisions to Address Cognitive Overload and Split Attention 9

3.1.1 Design Decision 1: Line-Level Granularity of Program Traces 9

3.1.2 Design Decision 2: Minimizing Lesson Text 10

3.1.3 Design Decision 3: Affordances for Reading Lesson Text with Coordinating Visualizations 11

3.2 Lesson Interface 12

3.2.1 Interactive Panel 12

3.2.2 Code Snippet Panel 12

3.2.3 Computer Panel 14

3.2.4 Quiz Interface 15

4 Study Method 18

4.1 Participants 18

4.2 Procedure 19

4.3 Data Collection & Analysis 19

5 Results 21

5.1 Lesson Usage Statistics 21

5.2 Quiz Performance and Usage Statistics 22

5.3 Opinion Questionnaire Responses 23

6 Discussion and Implications for Design 26

6.1 General Effectiveness of the Tutorial System and Visualizations 26

6.1.1 Feedback on the Lessons Generally Positive 26

6.1.2 Lesson Usage Behaviors Suggest Engagement 27

6.2 Importance of Quizzes and Answer Explanations 28

6.2.1 Feedback on the Quizzes Strongly Positive 28

6.2.2 Performance on the Quizzes Suggests Engagement 28
# LIST OF TABLES

1. The icons that provide visual cues in the lesson text to direct the user’s attention to relevant visualizations.  
   
2. Participant usage statistics for the Basic Loops lesson. A *step navigation* was counted each time a participant advanced to the next learning step or took a *backward step navigation* to the previous learning step. An *episode of backtracking* was counted each time a participant made a series of consecutive backward step navigations.  
   
3. Participant performance on the Basic Loops quiz. A green ✓ (checkmark) symbol denotes a correct answer, and a red × (ex) symbol denotes an incorrect answer.  
   
4. Time spent on the Basic Loops quiz questions.  
   
5. Participant responses to the opinion questionnaire. The responses to the first four questions were on a 5-point scale from 1 (strong negative) to 5 (strong positive). The four possible responses to the last two questions were *Strong No, No, Yes*, and *Strong Yes*. Positive responses are highlighted in green. Negative responses are highlighted in red. The middle responses of 3 on the 5-point scale are highlighted in yellow. Due to an error, no opinion questionnaire data were collected for P8.
LIST OF FIGURES

1. The Coding Companion lesson interface. The interface includes (A) an interactive panel that contains the lesson text, (B) a source code visualization, (C) a console visualization to depict program output, and (D) a computer memory visualization.

2. The interactive panel for presenting and navigating lesson text. At the top is (A) the title of the current lesson and (B) the current topic within the lesson, with buttons for navigating between topics. The middle section is scrollable and displays (C) previously visited learning steps (faded text) and (D) the current learning step (bold text). In this example, the current learning step includes three icons from Table 1 that direct the user’s attention to the code and memory visualizations. The current step is followed by (E) several steps for which no explanatory text is provided (indicated by the pointing hand icons). At the bottom of the panel are (F) buttons for navigating between learning steps and a progress indicator for the current topic.

3. Example code snippet visualization. The code visualizations may include a lightning-bolt icon denoting the line of code just executed in a program trace (see line 4 above), a stop-sign icon denoting the next line of code to be executed in the trace (see line 6 above), and highlighting to draw the user’s attention to a particular section of code (see “x <= 3” on line 6 above).

4. Example computer memory visualization. For its memory visualization, Coding Companion integrates the one provided by Python Tutor [5]. In the above example, there is one stack frame (for the main method call) that stores three variables (r, divisor, and num), and one of those variables (r) holds a reference to an instance of the Java API’s Random class.

5. Example quiz question for which the user has not yet selected an answer. The question refers to accompanying code and console visualizations (not shown).

6. Example quiz question after a user has selected an incorrect answer. The question refers to accompanying code and console visualizations (not shown). Once the user answers a question, an explanation of the correct answer is displayed. The explanation is provided regardless of whether the user answered correctly or incorrectly.
The amount of time that each participant spent on each learning step in the Basic Loops tutorial. Each box depicts the interquartile range of time values. The horizontal line inside each box depicts the median time value. The whiskers depict the min and max time values. The \( \times \) (ex) symbols depict the mean time values.
Chapter 1

Introduction

For learners of computer programming, gaining skill in program comprehension is critical to success; however, mastering comprehension skills is notoriously difficult. Program comprehension, also sometimes called “program understanding” or “source code comprehension”, is the process of developing mental models of a software system, including its intended architecture, meaning, and behavior [28]. Program comprehension ability is a well-established differentiator between expert and novice programmers (e.g., [1, 17, 21]). Despite the importance of program comprehension, multiple studies have shown that many students coming out of introductory programming courses lack this core skill [11, 10].

In particular, many students struggle with program tracing [10], a critical skill for program comprehension. Program tracing is “the ability to systematically, manually execute (‘trace’) a piece of code” [10] and mainly involves the construction of an execution trace. Given a computer program and a particular set of program inputs, a student should be able to follow the sequence of program instructions and identify program state changes (e.g., changes to the values of variables) that result from the execution of each instruction. According to Nelson et al.’s theory of program tracing knowledge [15], to correctly construct an execution trace, students must both understand
the execution rule logic of the program interpreter and map the execution rules to the syntax and state that determines what rules are followed and in which situations. However, many students fail to acquire sufficient program tracing skills from introductory programming courses [11, 10].

To help students learn program tracing, researchers are investigating software systems for learning program comprehension. Although these systems cover a variety of programming languages and vary in the details of their designs, they are all based upon the same approach to visualizing program execution traces. In this type of execution trace visualization, a learner can step through each program instruction and see the effect of the instruction on computer memory and console output. Some prior systems provide this type of visualization as a stand-alone aid for students to promote learning [5, 9, 14, 29]. Other prior systems have extended the visualization with natural language explanations of each execution step [15, 22, 27]. Some prior systems have also incorporated challenge questions or exercises involving the visualization to assess gaps in the learner’s understanding [15, 22, 27]. The most recent entry in this design space built upon this foundation by integrating the visualization (including step explanations and challenge questions) into a full tutorial system with natural language lessons on program comprehension [15].

Although evaluations of these systems have generally shown their promise for helping learners gain program tracing knowledge, empirical studies of these system to date have been fairly limited. Several key open questions about the design of such systems remain. One open design question is how best to address the complexity of program execution. An execution trace of even a simple program can involve many execution steps and state changes. On one hand, a learner needs to be able to understand and predict the effects of all these steps. On the other hand, the sheer number of steps can create educational challenges. For example, with so many steps to keep track of, the learner may experience high cognitive load, which is known to hinder learning [24]. Further, the tedium of
reading through so many steps may hurt learner engagement and motivation. Another open design question pertains to prior systems’ use of multiple visualization elements (e.g., the code, the console, and the memory stack visualizations) that are spread across different parts of the screen, potentially causing learners difficulty in making connections between the elements and noticing important changes to an element. Finally, in general, there has been a dearth of studies to understand in detail how students interact with and use these types of systems.

To address this gap in the literature, we took a usability testing approach [16] and performed a detailed user study of learners using a tutorial system for learning program comprehension. We conducted an observational laboratory study involving 9 undergraduate college students enrolled in an introductory programming course. We built our own tutorial system for learning program comprehension, Coding Companion. Although the high-level design of Coding Companion follows that of PLTutor [15] and other prior systems, we made three key design decisions to manage the complexity of program execution traces, which adds a degree of novelty to our design, described in Section 3.

Based on this work, we make the following key contributions:

• Empirically grounded implications for the design of tutorial systems for learning program comprehension based on the results of our user study.

• The design of Coding Companion, a new tutorial system for learning program comprehension that is both representative of prior systems at the high level and introduces several novel detailed design decisions to help manage the complexity of program execution traces.

• An open source implementation of our Coding Companion tutorial system (to be released upon publication of this work).
Chapter 2

Background and Related Work

2.1 Systems That Teach Program Comprehension with Program Tracing

Over the past couple decades, a family of systems has emerged in the literature that aim to help novice programmers gain program tracing knowledge using a common approach to visualizing program traces [5, 9, 14, 15, 22, 27, 29]. In particular, these systems provide three coordinating visualizations to depict program traces: the program code, the program console, and the state of computer memory. They enable the user to navigate through a program trace in a stepwise fashion, and with each step the user takes, the visualizations update to depict what is happening as the program executes. These systems vary in what counts as a step in the trace, with some using token-level granularity (e.g., [15]) and others line-level granularity (e.g., [5]). The program code visualization typically shows the code being traced and highlights which part of the code was just executed in the trace. The console visualization typically displays any output that has been generated so far in the trace. The memory visualization typically shows a table and/or graph representing the current state of the program stack and heap memory in the trace,
including the program variables and their values. These visualizations generally appear in separate panels on different parts of the screen (e.g., as in Fig. 1).

Many of these systems provide only the program trace visualization with little else [5, 9, 14, 29]. The visualization is intended to play a supporting role in instruction on program comprehension. For example, Python Tutor [5] has been embedded in multiple etextbooks [12, 18] to support the books’ instructional text.

Some systems have enhanced the visualizations with natural language explanations of the trace steps [15, 22, 27]. These explanations help direct the user’s attention to particular aspects of the visualization and provide deeper insight into the program semantics. For example, in PLTutor’s lesson on variables, an explanation might say “Look at the namespace and find the variable named box. See that the variable named box still holds the value 1” [15].

Some systems have also integrated interactive questions to test the user’s understanding as they use the system [15, 22, 27]. These questions, typically multiple choice, ask about different aspects of a program trace, such as “What is the output of this program?” or “What is the value on the stack?” Feedback on answer correctness is often provided immediately and may include a textual explanation for how to get the correct answer for missed questions.

Finally, a recent addition to this space, PLTutor [15], provides tutorial lessons in addition to all of the features mentioned above. These lessons go beyond explanations of individual steps to provide comprehensive coverage of programming concepts. Unlike the other systems in this space which act as a support for instruction, PLTutor is a standalone instructional system for learning program comprehension. In this work, we model the design of our Coding Companion tutorial system for learning program comprehension most closely on PLTutor.
2.2 Potential Problems with Prior Systems

Although there has been substantial research activity around the design of educational systems for program comprehension, the empirical studies of these systems have, to date, been limited. For some systems, no empirical evaluations have been reported in the literature (e.g., [5, 27, 29]). The few studies that have been conducted tended either to focus narrowly on a particular system implementation or to evaluate the system as a whole, providing little insight into the individual design decisions that make up the system. Studies of user interviews and log data revealed problems that users faced in interacting with Jeliot 3 [13] and UUhistle [20]; however, it is difficult to see how these finding generalize. Studies of student test performance revealed improved learning gains for users of Jeliot 3 [26] and of PLTutor [15] versus other instructional methods; however, the effects of the various design decisions that went into these systems remains unclear. Although these prior studies begin to scratch the surface on our understanding of this design space, many open questions have been yet to be studied.

Our user study aims to help fill this gap in the literature. It is motivated by three key problems that may exist in these systems that have received little or no previous attention: cognitive overload, the split-attention effect, and the expertise-reversal effect.

2.2.1 Cognitive Overload

The large number of steps involved in program traces—even of simple programs—may induce high cognitive load on learners reading and digesting trace visualizations. Cognitive load refers to the amount of information being stored in the brain’s working memory at one time [23]. Cognitive Load Theory [24] predicts that learning will be reduced when the cognitive load (the strain placed on a person’s working memory while engaged in learning) is too high. A large number of trace steps may significantly burden working memory, as students must keep multiple steps in their mind to understand the visualizations. Moreover, the addition of natural language explanations, while potentially
helpful, may add even more information for students to mentally process on each step—thus, further increasing cognitive load.

2.2.2 Expertise Reversal Effect

As users of the prior system gain more program tracing knowledge, the long program traces and accompanying explanations of trace steps may induce the *expertise-reversal effect* [8]. Expertise-reversal effect refers to instructional techniques that have a reverse effect for students of different levels of prior knowledge [7]. This theory explains how instructional scaffolding that focuses on building mental models can work well for beginners, but learners who have already mastered the mental model often perform worse when given such scaffolding. In program trace visualization systems, the high amount of scaffolding in the program trace visualizations may induce expertise reversal in users who have already mastered the material.

2.2.3 Split-Attention Effect

The multiple coordinating visualizations utilized by these systems may induce the *split-attention effect* [3]. Learners that must divide their visual attention across multiple elements to gather the information necessary for learning have been found to have increased cognitive load, and thus, a reduced ability to learn. For example, researchers have shown that an educational geometry visualization that includes a graph and a formula as separate elements will hinder student learning because their attention is divided between the two elements [25]. The need for users of the prior systems to split their attention between the code, console, and memory visualizations creates a situation ripe for the split-attention effect.
Chapter 3

Coding Companion Tutorial System for Program Comprehension

To investigate the design of tutorial systems for program comprehension, we designed and implemented Coding Companion, a web-based tutorial system that teaches basic Java concepts using lessons to explain concepts, visualizations to present execution traces, and quizzes to assess student knowledge. We designed Coding Companion to be representative of the family of educational systems from the literature discussed in Section 2. In particular, the design was modeled most closely on a recent tutorial system for program comprehension that has shown promise in the literature, PLTutor [15].

Coding Companion employs a lesson interface with multiple coordinating visualizations, as depicted in Fig. 1. The interactive panel (Fig. 1A) displays lesson text that is divided into learning steps. Three accompanying visualizations are used to depict the execution trace: a visualization of the code with annotations to denote the current execution state (Fig. 1B), a visualization of the console to depict the output produced by the trace steps (Fig. 1C), and a visualization of computer memory to depict the current state of memory at each step in the trace (Fig. 1D).
Figure 1: The Coding Companion lesson interface. The interface includes (A) an interactive panel that contains the lesson text, (B) a source code visualization, (C) a console visualization to depict program output, and (D) a computer memory visualization.

3.1 Design Decisions to Address Cognitive Overload and Split Attention

Although our aim was for Coding Companion to be generally representative of prior systems for teaching program tracing, we applied three key design decisions to our system that add a degree of novelty compared with the prior designs. These decisions aimed to address the potential issues with cognitive overload, expertise-reversal, and split-attention effects that we discussed in Section 2.2.

3.1.1 Design Decision 1: Line-Level Granularity of Program Traces

To reduce the number of program-trace steps, and thus help mitigate the potential issues of cognitive overload and expertise-reversal effect, Coding Companion treats a complete line of code as one step in a program trace. This decision is in contrast to representing trace steps at a finer level of granularity, such as the token-level granularity employed by PLTutor [15].
This decision makes a key design trade-off: reduced number of steps versus greater detail and completeness in the explication of traces. Our rationale for favoring fewer steps stems from our observation that, when using a token-based approach, many of the steps become trivial, particularly if the tokens are frequent or commonly used. For example, in the token-based approach, even the processing of a semicolon on the end of a statement receives a step. Thus, one token-level code trace might have hundreds of tokens and therefore hundreds of steps, even if the code is only a dozen lines long. As learners progress to more advanced concepts like arrays and methods, the multitude of trivial trace steps presented by the system seems likely to overload users with information, inducing high cognitive load and the expertise-reversal effect.

### 3.1.2 Design Decision 2: Minimizing Lesson Text

To further reduce the amount of content that learners must read and digest, we made the design decision to minimize the lesson text in Coding Companion. To reduce text without reducing effectiveness, we accomplished the reduction in three ways. First, in writing explanations of trace steps, we followed the principle of using as few words as possible to explain the step without sacrificing clarity. Second, we omitted explanatory text on trace steps for which a similar type of step had already been explained. For example, when executing a loop, the system would provide text to explain each step in the body of the loop for the first iteration, but would omit those explanations for subsequent iterations of the loop. For trace steps with no explanatory text, the system used a pointing hand symbol as a visual cue that the visualizations convey the information for the step (illustrated in Fig. 2E). Third, we further reduced the content by omitting certain trace steps. For example, a Java program has not truly finished executing until the `main` method returns `void`. However, such trace steps are seldom relevant to understanding the primary material being covered in the example. Thus, in Coding Companion, the trace step for returning from the `main` method is omitted from the lessons.
Table 1: The icons that provide visual cues in the lesson text to direct the user’s attention to relevant visualizations.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Line Icon</td>
<td>represented by a lightning bolt symbol, indicates the line of code that has just finished executing.</td>
</tr>
<tr>
<td>Next Line Icon</td>
<td>represented by a stop sign symbol, indicates the line of code that will be executed next.</td>
</tr>
<tr>
<td>Console Icon</td>
<td>represented by a console prompt symbol, indicates that new console output has appeared.</td>
</tr>
<tr>
<td>Memory Icon</td>
<td>represented by a memory chip symbol, indicates that the memory visualization has changed.</td>
</tr>
</tbody>
</table>

3.1.3 Design Decision 3: Affordances for Reading Lesson Text with Coordinating Visualizations

To help students follow concepts as they are visualized on multiple parts of the screen (and thus, to mitigate the split-attention effect), Coding Companion uses two strategies. First, Coding Companion’s content is laid out in a left-to-right reading order. The lesson text anchors the left-hand side of the screen, the code appears in the center, and the console and memory visualizations appear on the right (recall Fig. 1). For each learning step in the lesson, the reader first reads the step text (in the left pane) and then moves their attention right to any relevant visualizations. Second, icons connect the lesson text, the code execution, and the computer visualizations by providing visual cues in the lesson text that direct the user’s attention to a particular part of the screen (see Table 1). The icons aim to help users maintain awareness of all the relevant visualization information associated with the current step and prevent users from missing any important information. These decisions add a degree of visual integration to the separate visualizations with the goal of mitigating the split-attention effect.
3.2 Lesson Interface

3.2.1 Interactive Panel

The leftmost panel shown in Fig. 1A is the interactive panel for a lesson. This panel contains all the text content and clickable navigation links to progress through the current lesson. Fig. 2 details the features of this panel.

In addition to the high-level design decisions, we also made some minor decisions which shaped this panel.

First, we limited the amount of navigating between steps. The explanations of all previous steps of an example remain visible when going to the next step. Users can quickly reference something that happened previously by scrolling up, instead of having to click through the previous steps and then back. To help users focus on the current step, the text content for the current step appears in black, while content from any previous steps is faded but still legible.

Second, we helped users maintain awareness of their progress through a topic. The current step number and total number of steps in the topic appear on the right side of the learning step navigation bar at the bottom of the panel.

Third, we added provisions to stop users spam clicking through the steps. The user must scroll to the bottom of the content before the button to advance the learning step is clickable. Also, the next topic button is disabled until the user has viewed all the learning steps of the current topic.

3.2.2 Code Snippet Panel

The center panel (Fig. 1B) displays a visualization of the program code. The visualization includes icons from Table 1 denoting the current and next lines of code in the program trace. Fig. 3 details the features of this panel. As shown in Figure 3, the code text is displayed with syntax highlighting to emulate the appearance of text in common code
Figure 2: The interactive panel for presenting and navigating lesson text. At the top is (A) the title of the current lesson and (B) the current topic within the lesson, with buttons for navigating between topics. The middle section is scrollable and displays (C) previously visited learning steps (faded text) and (D) the current learning step (bold text). In this example, the current learning step includes three icons from Table 1 that direct the user’s attention to the code and memory visualizations. The current step is followed by (E) several steps for which no explanatory text is provided (indicated by the pointing hand icons). At the bottom of the panel are (F) buttons for navigating between learning steps and a progress indicator for the current topic.
4     int x = 1;
5
6     while (x <= 3) {
7
8         System.out.println("Value of x: " + x);
9
10        x++;
11
12     }

Figure 3: Example code snippet visualization. The code visualizations may include a lightning-bolt icon denoting the line of code just executed in a program trace (see line 4 above), a stop-sign icon denoting the next line of code to be executed in the trace (see line 6 above), and highlighting to draw the user’s attention to a particular section of code (see “x <= 3” on line 6 above).

editors. Some steps have yellow highlighting in the code to show that part of the line is most important to the current step (see Line 6 in the figure).

3.2.3 Computer Panel

The rightmost panel shown in Fig. 1 contains the console (C) and computer memory (D) visualizations. The console displays the output text that has been printed so far in the program trace. For the memory visualization, Coding Companion integrates the visualization engine provided by Python Tutor [5]. This visualization depicts a tabular representation of the program stack, including variables and their state, as well as a graph representation of objects in heap memory. As seen in Figure 4, the graph displays a list of variables in memory and their values on the current execution step. Primitive values are shown as text to the right of the variable name like “divisor” and “num”. Object variables are denoted with an arrow pointing to the class type and a representation of the Object if appropriate. Most of the variables in Coding Companion’s lessons are primitives, except for the example shown in the trace in Figure 4.
Figure 4: Example computer memory visualization. For its memory visualization, Coding Companion integrates the one provided by Python Tutor [5]. In the above example, there is one stack frame (for the main method call) that stores three variables (r, divisor, and num), and one of those variables (r) holds a reference to an instance of the Java API’s Random class.

3.2.4 Quiz Interface

The quiz interface uses the same three-panel layout as the lesson interface (recall Fig. 1), except that the leftmost interactive panel contains quiz questions instead of lesson text. Each quiz question appears in an expandable card, and the user can expand one question card at a time. Expanding a question card updates any code snippet and computer visualizations that go with the question and enables the user to answer the question. All quiz questions are multiple choice and, following best practices [6], offer three possible options to choose from (one correct answer and two distractors). A user can answer questions in any order, but only one question at a time before hitting the “Save” button seen in Figure 5. Unanswered questions are indicated by an open circle to the right of the question text. Clicking “Save” triggers the system to grade the response. After the response has been graded, the system fills in the circle next to the question text to indicate if the response was correct or incorrect, as shown in Figure 6. Once the user answers a
Figure 5: Example quiz question for which the user has not yet selected an answer. The question refers to accompanying code and console visualizations (not shown).

question, the question card displays the correct answer with an explanation of the answer, as depicted in Fig. 6. The explanation is displayed regardless of whether or not the user answered correctly.
4. What number should replace the "???" to generate the final console output shown?

![Quiz Question]

The numbers 0 to 3 have been printed to the console meaning that the last number to make the boolean condition true was 3. Since the less than operator (<) is used, the correct number to replace the "???" is 4, which is one more than the final number printed.

- 3
- i
- 4 ✓

Figure 6: Example quiz question after a user has selected an incorrect answer. The question refers to accompanying code and console visualizations (not shown). Once the user answers a question, an explanation of the correct answer is displayed. The explanation is provided regardless of whether the user answered correctly or incorrectly.
Chapter 4

Study Method

To understand the extent to which the design of Coding Companion is effective and to discover barriers that users of the system encounter, we took a usability testing approach [16], conducting an observational laboratory study of beginning programmers using our implementation of the system. For the study, the system emphasized learning how to comprehend loops (i.e., while, for, and do while loops) in the Java Programming Language.

4.1 Participants

Our participants consisted of 9 beginner programmers (7 males, 2 females) enrolled in an introductory Java programming course (CS0 or CS1) at our university. In responding to a background questionnaire, 3 participants (33%) self-reported Asian or Asian American ethnicity, 4 (44%) self-reported Black or African American ethnicity, 2 (22%) self-reported Native American ethnicity, 3 (33%) self-reported White, Caucasian or European American ethnicity, and 1 (11%) reported Hispanic or Latina/o ethnicity.

In terms of programming background, 8 of the 9 participants reported that they were CS majors or minors. All participants self-reported that they had average or below-average prior programming experience compared to their classmates. All
participants self-reported that they had little or no prior experience with loops at the time of recruitment. We timed our study recruitment based on the lecture schedules of the courses so that students would not yet have been exposed to loops in the courses.

4.2 Procedure

We observed each participant individually as they used Coding Companion. Each study session took approximately 2 hours. To better understand where participants placed their attention, we asked them to “think aloud” as they used the system.

The procedure for each study session involved six main parts. After each part in which the participant used Coding Companion, they participated in a short semi-structured interview about any issues they had and their thoughts on Coding Companion. First, the participant completed a tutorial on Coding Companion to become familiar with the interface. Second, the participant completed a lesson in Coding Companion on Boolean expressions, which was intended to be a review for the participants. Third, the participants took a quiz within Coding Companion consisting of 15 multiple-choice questions on Boolean operators and Boolean expressions. Fourth, the participants completed a lesson in Coding Companion on loops. This lesson was intended to cover material that was new (or mostly new) to the participants and was the focus of our later analyses. Fifth, the participants took a quiz in Coding Companion on loops, consisting of 12 multiple-choice questions. Finally, participants completed a background questionnaire and an opinion questionnaire, consisting of 13 questions that were a mix of close-ended Likert-scale questions and open-ended questions about the participant’s experiences with Coding Companion.

4.3 Data Collection & Analysis

During each study session, we collected a variety of video, audio, system log, and questionnaire data. In particular, we collected the following set of data for each
participant: (1) interaction-log data and screen-capture video of actions taken in Coding Companion during the tutorial, the lessons, and the quizzes, (2) responses to the questions on each quiz, (3) audio recordings of interview responses, and (4) written responses to the background and opinion questionnaires. We also attempted to collect eye-tracking data for each participant; however, we do not report that data, because of system malfunctions and misalignment during the study.

We performed both quantitative and qualitative analyses of the data collected. As quantitative analyses, we calculated participant usage statistics, such as the time spent on each step or quiz question, the number of backtracking steps during lessons, and measurements of quiz performance. As qualitative analyses, we performed qualitative coding [4] on quotes from participants related to their positive and negative experiences using the system.
Chapter 5

Results

In this section, we report the results of our quantitative data analyses, aiming to be as objective and neutral as possible (“just the facts, ma’am”). We reserve interpretation and discussion of the results for Section 6.

5.1 Lesson Usage Statistics

To understand how participants experienced the Basic Loops lesson, we analyzed their navigation of the learning steps and the amount of time they spent on the learning steps.

Table 2 reports statistics regarding participants’ navigation of the learning steps and their total time on the lesson. The lesson contained 60 different learning steps, and all participants visited all 60 steps in the lesson. Participants spent between 9 and 24 minutes on the Basic Loops lesson. Most participants had 3 or fewer episodes of navigating backward in the learning steps. However, one participant (P7) had many more episodes of backtracking than the others, resulting in approximately 1.5 times more step navigations than the other participants.

Fig. 7 reports statistics regarding the time that participants spent on individual learning steps in the lesson. Participants spent roughly between 10 and 20 seconds on each step, and it was uncommon for them to spend longer than 40 seconds on a step.
Table 2: Participant usage statistics for the Basic Loops lesson. A step navigation was counted each time a participant advanced to the next learning step or took a backward step navigation to the previous learning step. An episode of backtracking was counted each time a participant made a series of consecutive backward step navigations.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Total Time on Lesson (Minutes)</th>
<th>Number of Step Navigations</th>
<th>Number of Backward Step Navigations</th>
<th>Number of Episodes of Backtracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>9</td>
<td>60</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>11</td>
<td>66</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>P6</td>
<td>13</td>
<td>62</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P5</td>
<td>14</td>
<td>60</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P9</td>
<td>17</td>
<td>62</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P8</td>
<td>18</td>
<td>60</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>19</td>
<td>62</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>22</td>
<td>63</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P7</td>
<td>24</td>
<td>94</td>
<td>15</td>
<td>9</td>
</tr>
</tbody>
</table>

Mean (SD) 16.3 (4.7) 65.4 (10.3) 2.6 (4.5) 1.9 (2.7)

5.2 Quiz Performance and Usage Statistics

To understand how participants experienced the Basic Loops quiz, we analyzed their responses to the quiz questions and the amount of time they spent on the quiz questions.

Table 3 reports the correctness of participants’ responses to the quiz questions and statistics regarding their overall performance on the quiz. As the table shows, P2 and P4 had the most correct responses to the questions (11 out of 12 correct). In general, participants gave many more incorrect responses during the latter half of the quiz (Q7–Q12) than they did during the first half (Q1–Q6). Moreover, questions Q8 and Q11 stand out as the questions that received the greatest numbers of incorrect responses—only 3 participants answered Q11 correctly, and only one answered Q8 correctly.
Figure 7: The amount of time that each participant spent on each learning step in the Basic Loops tutorial. Each box depicts the interquartile range of time values. The horizontal line inside each box depicts the median time value. The whiskers depict the min and max time values. The $\times$ (ex) symbols depict the mean time values.

Table 4 reports statistics regarding the time participants spent on quiz questions. Participants spent between roughly 6 minutes and 24 minutes on the quiz, with the median time being around 11 minutes. Questions Q9, Q10, and Q12 stood out as the only questions for which the majority of participants spent longer than 1 minute on each question.

5.3 Opinion Questionnaire Responses

To understand participants’ opinions regarding various aspects of the Coding Companion system, we analyzed their responses to the opinion questionnaire.

Table 5 reports each participant’s responses to the Likert-style questions, and statistics regarding aggregate responses. Almost all participant responses to the opinion questions about Coding Companion were positive. The question about Coding Companion’s quizzes stood out as the only question to which all participants responded with the highest
Table 3: Participant performance on the Basic Loops quiz. A green ✓ (checkmark) symbol denotes a correct answer, and a red × (ex) symbol denotes an incorrect answer.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>Q11</th>
<th>Q12</th>
<th>Pct. Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>92</td>
</tr>
<tr>
<td>P4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>92</td>
</tr>
<tr>
<td>P5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>83</td>
</tr>
<tr>
<td>P8</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>83</td>
</tr>
<tr>
<td>P1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>75</td>
</tr>
<tr>
<td>P6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>67</td>
</tr>
<tr>
<td>P9</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>67</td>
</tr>
<tr>
<td>P3</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>58</td>
</tr>
<tr>
<td>P7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>50</td>
</tr>
</tbody>
</table>

The results of our qualitative analysis of the participant responses to the open-ended questions will be reported along with our discussion of the results in the next section (Section 6).
Table 4: Time spent on the Basic Loops quiz questions.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0:30</td>
<td>0:21</td>
<td>6:01</td>
</tr>
<tr>
<td>P4</td>
<td>0:33</td>
<td>0:19</td>
<td>6:38</td>
</tr>
<tr>
<td>P2</td>
<td>0:38</td>
<td>0:20</td>
<td>7:34</td>
</tr>
<tr>
<td>P5</td>
<td>0:49</td>
<td>0:31</td>
<td>9:43</td>
</tr>
<tr>
<td>P8</td>
<td>0:54</td>
<td>0:45</td>
<td>10:43</td>
</tr>
<tr>
<td>P7</td>
<td>0:55</td>
<td>0:40</td>
<td>11:06</td>
</tr>
<tr>
<td>P6</td>
<td>1:05</td>
<td>0:57</td>
<td>13:01</td>
</tr>
<tr>
<td>P9</td>
<td>1:29</td>
<td>0:55</td>
<td>17:44</td>
</tr>
<tr>
<td>P3</td>
<td>2:00</td>
<td>1:48</td>
<td>24:06</td>
</tr>
</tbody>
</table>

Table 5: Participant responses to the opinion questionnaire. The responses to the first four questions were on a 5-point scale from 1 (strong negative) to 5 (strong positive). The four possible responses to the last two questions were Strong No, No, Yes, and Strong Yes. Positive responses are highlighted in green. Negative responses are highlighted in red. The middle responses of 3 on the 5-point scale are highlighted in yellow. Due to an error, no opinion questionnaire data were collected for P8.

<table>
<thead>
<tr>
<th>Question</th>
<th>Participant</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>How helpful for learning did you find the Coding Companion system overall?</td>
<td>P1</td>
<td>5</td>
</tr>
<tr>
<td>How easy or hard did you find it to use and understand Coding Companion’s lesson interface?</td>
<td>P2</td>
<td>5</td>
</tr>
<tr>
<td>How easy or hard did you find it to understand Coding Companion’s code execution graphics?</td>
<td>P3</td>
<td>4</td>
</tr>
<tr>
<td>How helpful did you find Coding Companion’s quizzes?</td>
<td>P6</td>
<td>5</td>
</tr>
<tr>
<td>Would you use the Coding Companion system in the future if it were expanded to cover other Java programming topics?</td>
<td>P9</td>
<td>5</td>
</tr>
<tr>
<td>Would you recommend the Coding Companion system to a friend learning to code?</td>
<td>Strong Yes</td>
<td>Strong Yes</td>
</tr>
</tbody>
</table>
Chapter 6

Discussion and Implications for Design

6.1 General Effectiveness of the Tutorial System and Visualizations

Our data show a number of key indicators that the overall design of the system was effective for most participants. In particular, the opinions they expressed and the behaviors they exhibited while using the system reflected positively on the system.

6.1.1 Feedback on the Lessons Generally Positive

As the data in Table 5 show, the participants’ opinions about Coding Companion were largely positive. A strong majority of the participants expressed that they found the system helpful for learning. All but one participant responded to the question “How helpful for learning did you find the Coding Companion system overall?” with a 4 (2/8 participants) or a 5 (5/8 participants). Most of the participants also expressed that they found the system’s lesson interface understandable. All but two participants responded to the question “How easy or hard did you find it to use and understand Coding Companion’s lesson interface?” with a 4 (2/8 participants) or a 5 (4/8 participants). Overall, the
participants liked Coding Companion well enough that they said they would use the system in the future to learn other programming topics, as shown in Row 5 of the table.

Participants made numerous comments in their interviews that further confirm the effectiveness of the system’s design. For example, Participant P9 thought the system’s lessons were effective:

P9: “In this format and everything, with you teaching a lesson like this, I think that’s effective. I like it.”

Participant P1 liked how the system explained the material:

P1: “It was a good way of explaining it.”

A couple of participants noted how the system improves upon some of the typical ways this material is currently taught. Participant P5 thought that it improves upon classroom lectures:

P5: “I actually am getting more out of [Coding Companion], because we went over those same topics in class today, and I have a stronger grasp of it, because it went into more detail.”

Participant P4 thought that it improves upon textbooks:

P4: “I think this is more like a textbook with the code snippet for you to think over the code and all that. For me, I prefer something more— that’s more visual and more interactive.”

6.1.2 Lesson Usage Behaviors Suggest Engagement

As the data reported in Section 5.1 show, the behaviors that the participants exhibited while going through the lesson in Coding Companion also suggest that they were engaged in the lessons and were not having serious difficulties with the system. No participant gave up before completing the entire lesson, and all the participants viewed all 60 of the learning steps. Furthermore, the participants took time to read each learning step. As Fig. 7 shows, the participants spent between roughly 10 and 20 seconds on average per step. Most participants had a small number of episodes in which they went backward in
the lesson to revisit learning steps. As the rightmost column of Table 2 shows, 5 of the 9 participants had 1–3 episodes of backtracking. Small amounts of backtracking, like we saw here, might indicate that participants were engaged and understanding the lesson: they would occasionally need to go back to a previous learning step for clarification, but were not struggling with misunderstandings to the point that they frequently had to back up in the lesson. Only one participant (P7) exhibited a noticeably higher number of backtracking episodes (9) than the others, and 3 of the 9 participants did not backtrack at all.

### 6.2 Importance of Quizzes and Answer Explanations

#### 6.2.1 Feedback on the Quizzes Strongly Positive

One thing that stood out was how much participants liked the quizzes. All 9 participants took the quiz, answering all 12 questions. As Table 4 shows, they spent roughly between 6 and 24 minutes taking the quiz, with an overall average time of just under 12 minutes. The participants expressed strongly positive opinions of the quiz across the board. As Table 5 shows, all 9 of the participants responded to the opinion question about the quizzes with the most positive rating (5). The participants’ comments also frequently reflected their positive opinion of the quiz—for example, P9 summed it up:

P9: “Good quiz. Definitely good material.”

#### 6.2.2 Performance on the Quizzes Suggests Engagement

The quiz was designed so that earlier questions tended to be easier and later questions tended to be more difficult, and the participants’ performance on the quiz was consistent with the ascending difficulty. As Table 3 shows, the earliest questions (Q1, Q2, Q3, and Q4) were answered correctly by all or all but one participant on their first attempt. In contrast, a couple of the later questions (Q8 and Q11) were answered incorrectly by a strong majority of participants. The time participants spent on questions is also consistent with the escalating difficulty of the questions. As Table 4 shows, the average amount of
time that participants spent on each question from the first half of the quiz tended to be noticeably lower than the time they spent on each question from the latter half of the quiz. One participant, P9, noticed the increasing difficulty of questions and shared their opinion:

P3: “I liked it. It’s like a ramping up difficulty. It’s trying to go over the sense that you know the basic process, then slowly getting harder.”

Another participant, P5, wanted more of the difficult questions:

P5: “Some of [the questions] were good, like, toward the end, where it had the ones that did require a little bit more thought. [I would prefer] more of them that were closer to that length.”

These results suggest that the quiz successfully engaged participants and is consistent with their opinion that the quiz was helpful.

### 6.2.3 Answer Explanations Helpful—Even for Correct Answers

Once a user gave an answer to a quiz question, the system would display the correct answer to the question along with an explanation of the answer. Our data suggest that these answer explanations were a key reason the participants found the quiz so helpful. Missing a question could reveal a gap in a learner’s understanding; however, without an explanation of the correct answer, the learner might have difficulty filling that gap. Thus, we would expect that the answer explanations provide a key benefit to learners, and many of the participants made comments confirming it:

P5: “I would have noticed [the explanation] if I had gotten [a question] wrong.”

P7: “The feedback was very good. For someone like me, if I have a general understanding, then I just do some questions, and I kinda know from the feedback. So, if the questions have feedback, and they kind of explain to you why you got the answer wrong, then it’s very good.”

Interestingly, a couple participants even found the explanations helpful on questions they answered correctly:
P1: “I liked that they gave you—even if you did get them right—they gave you an explanation of why you got it right, instead of just not saying anything about it unless you got them wrong.”

P3: “. . . you might need [an explanation], ’cause towards the end, you might feel like you got [a question] right, but you don’t exactly know for sure that you got it right.”

This comment points to the important design insight that, although a learner may answer a question correctly, they may still lack a full understanding of why that answer is correct. Thus, providing explanations, even for questions answered correctly, is a good idea.

6.2.4 Pop-Up Questions Wanted

A final observation that further reinforces the importance of quiz-type questions is the strong demand from participants for “pop-up” questions during the lessons. During the Coding Companion system’s initial tutorial to train users on how to use the system, the system would intermittently pop up questions for the users to answer between steps in the lesson. These pop-up questions were not included in the later lessons on conditionals and loops, and the participants clearly missed them. Seven of the 9 participants expressed their desire for pop-up questions throughout the lessons, for example:

P1: “. . . in each section [of a lesson], maybe a question to answer yourself, after you’re done with the examples.”

P6: “I feel like there should be questions or a hands-on activity, because once you read [a lesson section], you just go on to the next topic, and you don’t really know if you understand it.”

P7: “[Pop-up questions] would make it easier. Just like in the tutorial, it makes you go back and review what you read.”

P9: When asked if they would want a pop-up question after steps in the lesson text, “Yeah, like ‘what do you think the outcome would be?’ . That would be engaging.”
The prevalence of participant demand for pop-up questions suggests that such questions may be a successful strategy for helping learners digest the many steps of a typical execution trace, for example, by breaking up the tedium of reading through all the steps.

6.3 Choice of Visual Cues for Reducing Split-Attention Effect

A key concern in the design of Coding Companion was addressing the potential of issue of split-attention effect caused by having multiple coordinating visualizations in different regions of the screen, and participants generally expressed that they understood the visual cues the system provided to help with the problem. As Table 5 shows, all but one participant responded to the question “How easy or hard did you find it to understand Coding Companion’s code execution graphics?” with a 4 (6/8 participants) or a 5 (1/8 participants). Furthermore, several participants made positive comments about the visual cues:

P5: “It’s definitely a good visual representation.”

P9: “I think the symbols you use are interesting, the stop sign, the lightning bolt, the little ram stick thing. That’s cool. I like how it also checks your reading, too.”

However, one participant, P6, did exhibit issues with understanding the visual cues and with missing information being conveyed in some of the visualizations. This participant gave the lowest response of all the participants, 3 (the middle value on the 1–5 scale), to the question about the understandability of the code execution graphics (Table 5). Furthermore, P6 expressed having difficulty understanding the visual cues during the initial tutorial:

P6: “Like, the stop sign symbol, like, I don’t really know much about it. Like, I know it’s, like, before a line is executed. I was kinda iffy about the stop sign, but everything else was fine.”
Later, during the quiz, P6 also failed to notice information being conveyed by the visualizations on the right-hand side of the screen:

P6: “There was one question, I think you were supposed to use the console output, but I didn’t really notice it until after, like, a minute . . . I didn’t really know to look there.”

Although the other participants generally understood the meaning of the visual cues, they expressed a number of concerns about them and of suggestions for improvement. For example, the intuitiveness of the visual cues was a concern:

P2: “When I see this bolt, it like reminds me of an error . . . And then this stop one makes me feel like an error . . . Stop sign makes me think I’ve crashed.”

P3: “I think you could change the [lightning] button to something else that’s like a green button, you know? Like, red button stop, then you just have the green button that’s a go, so you’re saying you’re executing the code for that line now.”

A couple participants mentioned that they would have preferred to do away with these icons all together and instead use highlighting of lines of code to convey the information:

P4: “The line that is being executed, why don’t you highlight it to get my attention instead of putting the icon next to it?”

P6: “I really feel like it could just highlight it, like, what line you are on, instead of putting a lightning symbol or a stop sign there, ’cause, like, I got kinda confused . . . I was looking at the wrong line.’

In addition to feedback about the visual cues, a couple of participants also expressed wanting a different layout of the visualization panes. In particular, they both wanted a two-column layout with the lesson text in the left column and all the other visualizations (code, output, and memory) in the right column:

P4: “Three columns, that’s too much for me . . . You can work with two screens, but three screens, you have to look back and forth, and that’s pretty tiresome to the eyes.”
P5: “It’s a minor change... where the code would be above where the console would be at. That would just be [my] personal preference.”

The motivation for these suggestions may have been a (possibly unconscious) response to the split-attention effect with the thought that arranging them vertically on screen might somehow reduce the effect. With having four separate coordinating visualizations on screen, efficient usage of screen real estate was a major design concern, and it remains an open question as to whether the visualizations could fit into the layout suggested by the participants.

Overall, the high number of comments that participants provided regarding the visual cues and layout suggests that, although our design of these elements showed a degree of success, there is still room for improvement. Based on the prior work on split-attention effect [2], a design that goes further to integrate the separate coordinating visualizations into a more unified visualization might be effective. However, it remains an open questions as to what form such a more-integrated visualization might take.

6.4 Tension between Information Overload and Completeness

Another key concern in the design of Coding Companion was information overload, and our participant responses suggest that they experienced a degree of information overload; however, they also recognized the importance of fully explicating execution traces, especially for beginners.

6.4.1 Feedback Suggests Some Overload and Expertise Reversal Effect

Although we incorporated design decisions into Coding Companion with the aim of mitigating information overload (recall Section 3.1), the amount of material in the Basic Loops lesson was still arguably high. The Basic Loops lesson had a total of 60 learning
steps divided into four parts: Introduction to Loops (3 steps), While Loop (21 steps), For Loop (16 steps), and Do While Loop (20 steps). The total number of words in the Basic Loops explanations was 1734, with an average number of words per step of 28.9 \( (SD = 27.9) \). On average, a learning step referenced 1.2 of the three visualizations (code, console, and memory; \( SD = 0.8 \)).

Multiple participants made comments that suggest that the amount of information led them to experience at least some degree of information overload:

P4: “[The lessons] should be intuitive, natural, so you don’t need the explanations.”

P6: “I felt like sometimes it just gets too wordy with the simple things… Sometimes I felt like I was reading too much. I got the main idea, and it just kept adding on, which isn’t bad, but its not my thing.”

P9: “It is wordy, though. There is a lot of wording in there. I don’t know if it would be worth maybe snipping that a little. It all depends on the student, I guess.”

Additionally, the participants’ desire for pop-up questions (recall Section 6.2.4) may also have been a response to cognitive overload, because such questions would create breaks in the lesson.

One participant, P4, stood out as especially disliking the amount of information being presented. They made numerous comments in this regard, such as:

P4: “This would only be useful for beginners, [not] for experienced coders or people as you go higher.”

P4: “I think the introduction is very thorough, but there is a lot of text… When I want to start learning code, I just want some sort of visual instruction, very short, very direct, and easy to understand.”

His aversion to textual explanations also applied to the quizzes:

P4: “If you provide explanations for every question, then someone who got those questions right, they will skip over them… You should provide explanations, but only for difficult questions.”
P4: “I think there should be an explanation for every question, but it will be automatically hidden if you answer it correctly, and it will only show if it’s a particularly difficult question that the teacher wants to emphasize.”

The reason for P4’s negative reaction may have been the result of aptitude–treatment interaction—in particular, the expertise reversal effect [8]. Although his answers on the background questionnaire did not indicate that he had more experience than the other participants, his performance on the lessons and quizzes suggested that his expertise was greater than most of the others. He received the highest score on the quiz (see Table 3) and completed the quiz with the second fastest time (see Table 4). He also had the second fastest time to complete the Basic Loops lesson (see Table 2) and fastest median time per step on the lesson (see Fig. 7).

If his expertise was indeed advanced, then many of the instructional steps and text may have been redundant and represented extraneous effort, leading P4 to experience the expertise reversal effect. For example, this effect may explain why, P4 gave the lowest response of all participants (3) to the question about how helpful the system was for learning (see Table 5).

6.4.1.1 Feedback Acknowledges Beginner Benefits

Interestingly, despite the fact that many participants appear to have experienced information overload during the lesson, several of them also indicated that fully explicating the execution traces could be beneficial—especially for beginners:

P5: “[The lesson] was really informative. It was able to fit a lot of information into a short amount of time, and still be clear enough to hang onto it and have a good grasp…”

P6: “Sometimes it’s, like, too wordy.” But, when asked about users with little programming knowledge, “Yeah, I’m kinda in that situation, so it helps a lot.”

P9: “If I was teaching somebody, I’d probably end up saying this much stuff, too, honestly.”
Thus, coping with the tension between information overload and full explication of execution traces remains a significant design challenge for tutorial systems on program comprehension. Our results suggest that full explanations of execution steps may be appropriate only for beginners. Adaptive fading of educational scaffolding has been successful in prior work to mitigate the expertise reversal effect, and thus, to design systems that are effective for learners with varying degrees of expertise [19]. A promising direction for future work might be to incorporate adaptive fading into the design of tutorial systems for program comprehension.
Chapter 7

Threats to Validity

No empirical study is perfect, and our user study has several key threats to validity. First, the sample size was small, creating a threat to the generalizability of our results; however, our sample of 9 participants is considered appropriate for our user testing methodology [16]. Second, the window of time for participants to have enough background knowledge of Java but still be unfamiliar with the topics covered was very short. Therefore, although we asked that all participants have little to no experience with the topics covered, they may have gained some knowledge between the time they were recruited and the time they sat for the study. Third, participants may have experienced some reactivity effects, meaning they may have acted differently than they normally would, because they knew they were being observed or guessed that the researchers had created Coding Companion. The questionnaire responses and comments were generally favorable which indicates this may have occurred. Also, since the participants were asked to view the lessons and quizzes, their usage patterns may have been different than if they were given the system to use on their own.
Chapter 8

Conclusion

In this paper, we reported on an empirical user study of Coding Companion, a tutorial system for learning to comprehend Java programs that is generally representative of prior educational systems for program comprehension from the literature (esp. [15]). Key findings of the study include the following:

- **Overall Effectiveness**: Participant feedback and usage data suggest that the system was generally helpful for learning and that its interface was engaging and not overly confusing.

- **Quizzes**: All nine participants were strongly positive about quizzes provided by the system, and many found answer explanations helpful, even when they had answered the question correctly. Furthermore, many participants wished for the addition of pop-up questions during the lesson.

- **Icons and Layout**: Although participants generally understood our icon-based approach to aid reading the coordinating visualizations, their many constructive comments about the icons and panel layout suggest that there is room for improvement in mitigating split-attention effect.
• **Information Overload versus Completeness**: Participant comments revealed a tension between information overload caused by long, complex program traces and the importance of completely explaining the traces, especially for beginners.

In conclusion, our findings suggest several promising directions for improving the design of educational systems for program comprehension. Pop-up style questions may be an effective means for breaking up the reading of long program traces and for mitigating information overload effects. Adaptive fading of trace steps based on a learner model may be an effective approach for eliminating redundant learning steps and for mitigating the expertise-reversal effect. New visualizations that even more tightly integrate and unify the lesson, code, console, and computer visualizations may go further toward eliminating the split-attention effect. Such innovations could make strides toward helping learners gain the program tracing knowledge they need to master computer programming.
References


Appendix A

Background & Opinion Questionnaire
How helpful for learning did you find the Coding Companion system overall?

Very Unhelpful ➔ Very Helpful

How easy or hard did you find it to use and understand Coding Companion’s lesson interface?

Very Hard ➔ Very Easy

How easy or hard did you find it to understand Coding Companion’s code execution graphics?

Very Hard ➔ Very Easy

How helpful did you find Coding Companion’s quizzes?

Very Unhelpful ➔ Very Helpful
Was there anything that you particularly liked about Coding Companion? If so, please tell us about it.
Was there anything that you particularly disliked or thought could be improved about Coding Companion? If so, please tell us about it. If you have any specific recommendations for improvement, please tell us those as well.
Would you use the Coding Companion system in the future if it were expanded to cover other Java programming topics?

- Strong Yes
- Yes
- No
- Strong No

Would you recommend the Coding Companion system to a friend learning to code?

- Strong Yes
- Yes
- No
- Strong No
What is your gender?

- Man
- Woman
- Non-binary category or something else; please specify:

What is your racial/ethnic identity? Please check all that apply.

- Asian or Asian American
- Black or African American
- Native American
- Native Hawaiian or other Pacific Islander
- White, Caucasian, or European American
- Hispanic or Latina/o
- Arab, Middle Eastern, or Persian
- Something else:
What are your current college major(s)/minor(s)? Check all that apply.

<table>
<thead>
<tr>
<th>Major</th>
<th>Minor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Science (CS)</td>
<td></td>
</tr>
<tr>
<td>Electrical &amp; Computer Engineering (ECE)</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
</tbody>
</table>

If you checked a major/minor for “Other”, please tell us what the major/minor is in.

_________________________________________________________________________________

_________________________________________________________________________________

_________________________________________________________________________________

How do you estimate your programming experience compared to your classmates?

<table>
<thead>
<tr>
<th>Very Inexperienced</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Very Experienced</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How do you estimate your experience with the following programming languages? If there are any programming languages not listed with which you have significant experience, please write them in at the bottom.

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Very Inexperienced</th>
<th>Very Experienced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>C</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>Python</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>C++</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>Visual Basic .NET</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>JavaScript</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>C#</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>PHP</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>SQL</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>Objective-C</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>MATLAB</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>R</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>Assembly</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>Swift</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>Go</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
<tr>
<td>Ruby</td>
<td>○ ○ ○ ○ ○</td>
<td>○ ○ ○ ○ ○ ○</td>
</tr>
</tbody>
</table>

Other: __________________
Other: __________________
Other: __________________
Other: __________________