A CASE-BASED APPROACH FOR SUPPORTING THE INFORMAL COMPUTING EDUCATION OF END-USER PROGRAMMERS

A Dissertation
Presented to
The Academic Faculty

by

Brian James Dorn

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computer Science

School of Interactive Computing
Georgia Institute of Technology
December 2010

A CASE-BASED APPROACH FOR SUPPORTING THE INFORMAL COMPUTING EDUCATION OF END-USER PROGRAMMERS

ACKNOWLEDGEMENTS

First and foremost, I thank my wife, Christine, for joining me in this crazy adventure. I am not sure she fully realized what she was getting into when she moved to Georgia, but I am grateful for every day we have shared and will share in the years to come. I cannot wait to see where life takes us!

My family has been a significant influence on my commitment to learning. I believe that seeing both my parents go through college instilled in me a fascination with higher education early on. They have humored my desires to earn a doctorate, despite regularly asking when I would get a job during those first couple years. I know of no better way to thank them than to proudly say I have finally finished.

Despite regular jokes about my obsession with being a college student forever, my friends outside of graduate school have been a source of support throughout this, at times seemingly never-ending, journey. With their unwavering faith in my ability to finish my Ph.D. and their regular inquires about whether they could call me "Doctor Dorn" yet, they have regularly humbled me, distracted me, and kept me in good spirits. I am honored to have such amazing lifelong friends, who are far too numerous to mention by name here.

I have been fortunate to have the support and friendship of numerous fellow students at Georgia Tech. Space prevents me from chronicling all the reasons I am grateful to my friends at Tech, but suffice it to say that their intellectual support, encouragement, and laughter have made my time at Georgia Tech enjoyable and memorable. Thank you to those who I have had the pleasure to get to know over the years, and in particular, I offer my gratitude to Tamara Clegg, Brian Landry, Brian O'Neill, Erika Shehan Poole, David Roberts, and Sarita Yardi. I must also thank all current and past members of the Contextualized Support for Learning lab who have offered their opinions on my work and have (at least) tolerated my lengthy comments on theirs. I am especially indebted to Allison Elliott Tew for her camaraderie and assistance with my research, often at the drop of a hat; I couldn't have asked for a better desk neighbor and collaborator.

I would be remiss if I did not acknowledge members of my committee. Mark Guzdial, Amy Bruckman, Christopher Hundhausen, John Stasko, and Janet Kolodner have provided invaluable feedback about my work during my time at Tech, and they have pushed me to investigate interesting problems that matter to real people. I owe an additional debt of gratitude to my advisor, Mark, who asked me some five years ago what it was that I wanted to do for my dissertation. Out of my nebulous, ill-defined ideas he helped me build a research agenda and has served as a tireless advocate for my work ever since.

I also extend my thanks to colleagues in the computing education research community, who have shaped my ideas greatly. In particular, I deeply value the guidance of Josh Tenenberg, Sally Fincher, and Briana Morrison. Their willingness to allow me to pick their brains over the last few years has been personally and professionally rewarding. The CSEd research field is better for having devoted mentors like them.

Georgia Tech is fortunate to have some of the best staff members with whom I have had the pleasure to interact. They make sure that students are well cared for and that the gears of research continue to turn. I am grateful to Monica Ross for handling of all the little things, to Don Schoner for making us all laugh, and to Brian Poole for dealing with technical problems at a moment's notice.

Last, but certainly not least, I am thankful for the National Science Foundation's research funding that supported much of my five years at Georgia Tech and the work detailed in this dissertation. Specifically, NSF grants CISE-0306050, ITR-0613738, and CCLI-0618674 made the research presented here possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Software development is no longer a task limited to professionally trained computer programmers. Increasing support for software customization through scripting, the opening of application programmer interfaces on the Web, and a growing need for domain specific application support have all contributed to an increase in end-user programming. Unfortunately, learning to program remains a challenging task, and the majority of end-user programmers lack any formal education in software development. Instead, these users must piece together their understanding of programming through trial and error, examples found online, and help from peers and colleagues.

While current approaches to address the difficulties facing end-user programmers seek to change the nature of the programming task, I argue that these challenges often mirror those faced by all novice programmers. Thus, pedagogical solutions must also be explored. This dissertation work investigates the challenges that end-user programmers face from a computer science education perspective. I have engaged in a cycle of learner-centered design to answer the high-level questions: What do users know; what might they need to know; how are they learning; and how might we help users discover and learn what they need or want to know? In so doing, I uniquely frame end-user programming challenges as issues related to knowledge and understanding about computer science. Rather than building new languages or programming tools, I address these difficulties through new types of instructional materials and opportunities for felicitous engagement with them.

This work is contextualized within a specific domain of non-traditional programmers: graphic and web designers who write scripts as part of their careers. Through an in-depth, learner-centered investigation of this user population, this dissertation makes five specific contributions:

1. A detailed characterization of graphic and web design end-user programmers and their knowledge of fundamental computing concepts.
2. An analysis of the existing information space that graphic and web designers rely on for help.
3. The implementation of a novel case-based learning aid named ScriptABLE that is explicitly designed to leverage existing user practices while conveying conceptual knowledge about programming.
4. Initial confirmatory evidence supporting case-based learning aids for the informal computing education of web and graphic design end-user programmers.
5. An argument in support of the value of normative computing knowledge among informally trained programmers.

CHAPTER I

INTRODUCTION AND MOTIVATION

Recent advances in the software industry and on the Internet are significantly expanding the programming and software development domain. Software artifacts are now being created by millions of people who have little or no training in computer science, nor consider themselves programmers by conventional definitions. Architects and designers looking to extend the creative abilities of their CAD tools note that through scripting they have “unprecedented accessibility to the generative possibilities and comprehension of equation-based geometry” (Saunders, 2009, p. 133). Professional end users like mathematicians, biologists, and physicists are building software systems to manage and make sense of their knowledge-rich domains (Segal, 2007). Newly opened application programmer interfaces on popular Internet sites like Facebook, Flickr, and Google enable millions of users to create new user-generated web content (Yardi, Dorn, Bruckman, & Guzdial, 2008).

Software development is no longer confined to professionally trained computer programmers. This growing group of informal developers make up a class of computer users known as end-user programmers (EUPers). Broadly defined, EUPers are those individuals who use applications that incorporate features like textual scripting, high-level declarative specification, programming by example, and automation or customization via wizards (Nardi, 1993). Estimates based on projections from the Bureau of Labor Statistics report that over 90 million Americans will use a computer at work by 2012, with 55 million making heavy use of programmable applications like spreadsheets and databases. Of these, it is estimated that 13 million people will describe themselves as non-professional programmers (Scaffidi, Shaw, & Myers, 2005).

Comparing the size of this subset of end-user programmers to the mere 3 million professionally trained programmers in the workforce, it is obvious that informal software development demands the research community's attention.

Computer science educators have long recognized that learning to program is a difficult task, and the barriers end-user programmers face have much in common with issues any novice programmer encounters. They struggle to devise algorithms which solve the problem at hand; they wrestle with particulars of language syntax; and they are at times mystified by program bugs (Ko, Myers, & Aung, 2004). Even in situations where EUPers can easily master a programming language, other challenges arise. Segal notes "depending on the context in which the software is going to be used, issues such as code comprehensibility, software robustness and performance become important" (Segal, 2007, p. 111). Professionally trained programmers take years of coursework in fields like computer science to learn skills and techniques to deal with these issues, but most EUPers are unlikely to ever enroll in formal coursework. Nonetheless, users still depend on the correctness of such software in order to ensure the stability of third-party web servers, to make important business decisions, or to develop scientific theories. In an extreme case, Panko (1995) recounts the story of an oil company that lost millions of dollars due to spreadsheet errors resulting from a lack of rigorous testing practices.

Given the above picture, training opportunities for those engaged in end-user programming activities seem crucial. The challenge for computer science education is that EUPers approach learning in a fundamentally different way from their professional counterparts. EUPers acquire computing knowledge bit-by-bit along the way as they go about solving tasks in the real world (see e.g., Dorn & Guzdial, 2006). That is, they are taking part in informal education. Here, the term informal education refers to:

> The lifelong process whereby every individual acquires attitudes, values,

skills and knowledge from daily experience, educative influences and resources in his/her environment–from family and neighbors, from work and play, from the market place, the library and mass media. (Titmus, 1989, p. 547)

Re-envisioning end-user programmers as informal computer science learners opens numerous research directions. We can move beyond simply building new technologies that enable end-user development and begin asking questions about how the end user comes to understand computing by taking part in software development. Some obvious questions emerge: What do EUPers know about computer science? How do they go about acquiring new knowledge? How might we design learning environments that scaffold their independent learning processes?

The purpose of this dissertation is to explore informal learning among one particular end-user programmer population, in order to:

- understand the programming knowledge currently held by users,
- describe the processes by which such users learned what they know,
- develop an educational resource which is easily integrated into current informal learning practices, and
- show that such a resource can promote measurable learning of computer science concepts.

In particular, I am concerned with supporting end-user programmers' development of "normative" computing knowledge. With the term normative here, I mean to explore standard computing concepts identified by mainstream introductory computing education curricula (e.g., those of CC2001 (The Joint Task Force on Computing Curricula, 2001)). I have chosen to focus on a subset of these topics here for two reasons. First, as a discipline, computing educators have built significant curricula on top of a collection of basic programmatic constructs that are similar across most introductory CS experiences (Tew & Guzdial, 2010). Given that programming fundamentals comprise a large portion of the core computing body of knowledge (The Joint Task Force on Computing Curricula, 2001), I believe these concepts are valuable for all people engaged in scripting or programming activities. Second, and perhaps more important, my formative work suggests there is an opportunity to enhance end-user programmers' productivity by providing them with additional knowledge about basic concepts related to testing and code modularity.

1.1 Graphic Design End-User Programmers

There are many distinct end-user programming domains in which to situate this dissertation work. I have chosen to explore these issues among web and graphic design professionals who write scripts as part of their daily tasks. I have chosen to examine this group of users because they are a large but previously unstudied population, and they already make use of scripting languages closely resembling traditional programming languages (i.e., they do not use highly domain-specific systems like spreadsheet formulae for their scripts).

Graphic/web designers and others involved in media editing make up a relatively new and growing group of end-user programmers. In the realm of image editing, professional software packages like Adobe Photoshop and GIMP implement built-in scripting interfaces via languages like JavaScript, Scheme, and Python. To give a sense for the type of code written in this environment, Figure 1 illustrates a short example written in ExtendScript, Adobe's implementation of JavaScript, which automatically generates 10 frames for an animation that rotates an image 360 degrees in Photoshop CS2.

Experienced programmers immediately recognize many common aspects in this

```
//Setup units and document references
preferences.rulerUnits = Units.PIXELS
var docRef = documents[0]

//Resize canvas to be large enough in both dimensions
var diagSize = Math.sqrt(Math.pow(docRef.height, 2) +
                         Math.pow(docRef.width, 2))
docRef.resizeCanvas(diagSize, diagSize)

//Generate 10 layers for the animation frames
for (var i = 0; i < 10; i++)
{
  var toRotate = docRef.artLayers[0].duplicate()
  toRotate.name = "View␣" + (i + 1)
  toRotate.rotate(36)
}
```

Figure 1: Example ExtendScript for Generating Frames

example: variable declaration and use, mathematical computation, looping constructs, method invocation using the dot-notation, and explanatory comments. Closer examination also reveals that statements like `references.rulerUnits...` and `docRef.resizeCanvas(...)` make use of an extensive API. In fact, Photoshop is accompanied with a 335-page scripting reference manual for ExtendScript (*Adobe Photoshop CS2 JavaScript Scripting Reference*, 2005) that is not unlike documentation found for the Java API.

In formative work, I conducted an online survey of graphic and web designers who script in order to gain insight into the scope of their activities (Dorn & Guzdial, 2006). Responses indicate that these users are taking part in substantial programming. They write scripts of significant length and level of sophistication, despite having little to no formal training in computer science. Through scripting, they build software to do things like achieving custom effects not available in the standard tool set and automating batch jobs to cut down on repetitive tasks. They report relying heavily on code examples and documentation like FAQs to learn. They reported a high propensity to reuse existing code (their own and that of others) and to share the scripts they produce with others.

Through this dissertation I will provide a detailed characterization of this user population that reinforces the observations from this formative study. In particular I will present research data about their knowledge of introductory programming concepts, their learning strategies, and the instructional content of resources that they use.

1.2 The Case for Cases

Heavy reliance on example code and frequently asked question style documentation suggest that there is significant potential to use case-based learning aids to scaffold learning about computer science topics for this community. In essence, case-based learning aids distill the experiences of others, presenting them as a library of case examples wherein important lessons are clearly identified, to help novices learn about a skill or task (e.g., Guzdial & Kehoe, 1998; Goel, Kolodner, Pearce, Billington, & Zimring, 1991).[1] Put simply, case-based learning aids give learners a set of worked examples that highlight particular pieces of information relevant to the solution.

With respect to graphic design EUPers, a case might contain an initial problem statement like: "I want to write a program that removes all non-visible layers from a Photoshop image." The case might then include sample input images, a narrative of how the problem was solved, and a code listing of the final solution. The narrative would also introduce necessary computing concepts which are necessary to solve the problem in general (e.g., recursion in this particular case).

Case-based learning aids are a promising means to present computer science content to EUPers for several reasons. The survey showed the most highly rated sources of support for learning were examples of similar tasks from which users could borrow ideas and/or copy code, followed by FAQs and tutorials. Case-based educational materials are centered around specific tasks and source code, making them a close

[1]A more thorough discussion of case-based learning aids is presented in Chapter 2.

fit for reported user practices. Users can continue to use strategies developed while searching other code repositories online, but they will also encounter supplementary information about the solution that goes beyond source code. My analysis of an existing project repository found that many topics were simply not present in user submitted code, and further that most of the projects were, at best, sparsely commented (Dorn, Tew, & Guzdial, 2007). A moderated case library could more directly target which concepts are to be explored and make important concepts salient, rather than rely entirely on community submissions.

In this dissertation I provide substantial evidence in support of case-based learning aids for graphic and web design end-user programmers. Further, I report on the development and evaluation of ScriptABLE, a case-based learning aid designed to scaffold learning of computing concepts among graphic design end-user programmers. Specifically, I will address the following thesis statement with this work.

1.3 Thesis Statement

A case-based learning aid for graphic and web design end-user programmers can leverage current user practices of project and example-driven learning, promote the use and browsing of instructional content, and thereby foster the appropriation of knowledge about normative programming concepts.

1.4 Research Questions

To address this thesis statement, I pose four research questions which will be investigated in three studies. The first two questions provide the necessary foundation to inform the design and content of a new case-based learning aid named ScriptABLE. The final two questions pertain to the evaluation of this resource's effectiveness. I will discuss each of these three studies in turn in the remaining sections of this chapter.

RQ1: What is the nature of graphic/web design end-user programmers' knowledge of normative computing concepts?

RQ2: What learning practices do graphic/web design end-user programmers currently employ, and to what extent do typical resources provide opportunities to learn about normative computing concepts?

RQ3: How does the presentation of conceptual information as a case library influence the way end users interact with resources?

RQ4: To what extent does ScriptABLE as a case-based learning aid enable the appropriation of computing knowledge for users actively engaged in project-oriented programming activities?

1.4.1 Understanding Current and Learning Strategies

The first study in this dissertation focused on aspects of research questions one and two. To examine what users know about a set of introductory programming concepts, I conducted a card sorting activity with practicing web and graphic designers to elicit their knowledge. By comparing the results of individual sorts, I identified a limited collection of concepts that participants rate as difficult, misunderstood, and infrequently used. These findings show natural opportunities for additional conceptual learning. I also used a semi-structured interview to gather qualitative data about web and graphic designers' learning strategies related to scripting and programming. Findings from the interview data point to example and project-driven aids as scaffolds to promote learning while also matching users' current information seeking strategies.

1.4.2 Examining the Conceptual Coverage of a Script Repository

In the second study, I investigated the degree to which an existing popular web-based resource for web and graphic design scripting contained instructional content related to introductory programming concepts. I performed a content analysis of all example scripting projects posted on Adobe's online repository for user-generated scripts. I found a noticeable lack of content for those concepts previously identified as difficult and misunderstood by my target user population. While this repository has visibility and credibility stemming from its corporate host, I argue that it is somewhat deficient as an educational resource—it lacks sufficient example content to support learning that the target audience needs.

1.4.3 Evaluating the Effectiveness of Case-Based Learning Aid

The last study of this dissertation investigated research questions three and four—how presenting conceptual computing content in a case library alters information seeking behaviors and how it can support measurable learning of these concepts. I distilled the findings of the first two studies to a set of concrete design guidelines for educational resources to support graphic and web design end-user programmers. From these guidelines I designed and implemented a novel case library that explicitly targets a limited set of programming concepts. I then performed a laboratory study with users to measure the case library's effectiveness. I found evidence that using a case library encourages browsing navigation behaviors and reduces use of search features. Additionally, I verified that structuring programming content knowledge within a case library can indeed foster the development of normative conceptual understanding for end-user programmers.

1.5 Structure of the Dissertation

This dissertation is made up of six chapters beyond this introduction:

- Chapter 2 discusses related research literature from human-computer interaction, the learning sciences, and computing education.
- Chapter 3 presents the first of the three studies. In it, I outline findings related to web and graphic designers' current knowledge of programming concepts and the strategies they used to learn new information.

- Chapter 4 details the content-analysis study that I conducted to explore the degree to which an existing scripting resource could support learning of introductory computing concepts for web and graphic designers.

- Chapter 5 distills design goals for a new web-based instructional resource for this end-user programmer population. I then present the design and implementation of ScriptABLE, a case-based learning aid built with these goals in mind.

- Chapter 6 describes the evaluation study I conducted for ScriptABLE and details my findings in support of case-based learning aids for web/graphic design end-user programmers.

- Lastly, Chapter 7 provides explicit answers to the research questions based on the findings presented in earlier chapters. I also outline the contributions of this dissertation and briefly comment on potential future research directions.

CHAPTER II

BACKGROUND AND RELATED WORK

This chapter begins with a broad overview of prior end-user development research looking at both the challenges end-user programmers face and several recent attempts to mitigate these difficulties. I then differentiate this dissertation work from existing efforts as a learner-centered approach, rather than a strictly user-centered one. I conclude the chapter by introducing relevant literature from the learning sciences community.

2.1 Supporting End-User Programmers

The underlying challenge in supporting end-user programmers lies in the need to empower them to perform tasks for which they lack knowledge and skill. As Beringer (2004) puts it:

> As an expert design topic, end-user development (EUD) is rather new to human-computer interaction (HCI), although it is implicitly embedded in many design projects. What makes EUD different from other HCI topics is that in traditional HCI terms, users are experts in their tasks, and good tools should match these tasks. Conversely, end-user developers are trying to complete development tasks in which, by definition, they are not experts. Therefore, the dominating design goal of EUD tools is to compensate for a discrepancy between the user's expertise and the development task to be performed. (Beringer, 2004, p. 39)

The prevailing research direction that stems from this view is one that seeks to change the activity of programming to better fit the end user. Thus, there are field studies of EUPers in their natural work environments (e.g., Wiedenbeck, 2005), and there are lab studies exploring feature use patterns among different groups of end-users (e.g., Beckwith et al., 2006). Activities like these ultimately feed into the development of new programming environments and languages designed to minimize user frustration (e.g., Leshed, Haber, Matthews, & Lau, 2008; Burnett et al., 2001). To further depict existing research approaches, I will briefly discuss three representative EUP approaches.[1]

2.1.1 Programming by Demonstration

The first solution is a general class of work classified as programming by example or programming by demonstration (PBD). The motivation behind PBD efforts is that creating programs ought to be as straightforward as "showing" the computer an example of what it is that the user would like to accomplish (Cypher, 1993). That is, they minimize (or entirely eliminate) syntax in favor of recording facilities through which users specify their intended process. Dozens of PBD systems have been developed over the last two decades implementing this basic approach. For thorough reviews of PBD systems, see Cypher (1993) and Lieberman (2001).

One of the more recent systems to subscribe to a PBD approach is CoScripter (Leshed et al., 2008; Little et al., 2007). CoScripter is a collaborative scripting environment for automating web-based processes. Using a web browser plugin, the system records user actions and saves them in a human readable form that can be played back at a later time. The system utilizes the inherent structure of web pages and the limited domain of available actions on the web to infer user intent.

For example, a user might want to create a script that performs a Google search on his or her name. The user would begin recording and navigate to `http://www.google.com`. The user would then enter his or her name as a search term and click

[1]For a more complete depiction of the EUP field, see Lieberman, Paternó, and Wulf (2006).

the search button. Meanwhile, CoScripter watches the user's actions and creates a script that will perform this series of operations. Additionally, the system recognizes that the user's name matches a personal variable called "name" and will automatically substitute the variable's value during playback, rather than relying on the static string input during recording. This way the script can be shared with other people and will work without any editing.

PBD systems suffer from some basic limitations. The most obvious is that script playback often requires that the playback environment matches the original environment exactly. In the case of CoScripter, changes in the structure of the web-pages visited by the script could interfere with correct script operation (Leshed et al., 2008). The inference based approach for determining a user's intent in PBD systems also is problematic because as the context in which the programming task must take place grows, the more difficult it becomes to unambiguously decide what the user means to do every time. Lastly, the flexibility of PBD systems is significantly more limited than general purpose programming environments due to the reliance on user performable actions.

2.1.2 Natural Language Approaches

An alternative to the PBD approach is to reduce the complexity of programming languages, thereby making them easier to use and/or less prone to error. The basic idea is to derive programming syntax that leverages human knowledge of natural language. That is, we should be able to write executable programs that match how we would normally describe the solution in every-day language.

There are numerous projects that seek to develop new syntax that is more human friendly. The HANDS system (Pane, Myers, & Miller, 2002) is a recent example of a programming environment in this category. HANDS is a programming system for children that was designed “to provide a close mapping between the way the programmer envisions a problem solution and the expression of that solution in program code” (Pane et al., 2002, p. 199) Its syntax was derived from non-programmers’ English descriptions of solutions to programming problems (e.g., game design) (Pane, Ratanamahatana, & Myers, 2001). These studies resulted in a syntax that is largely event based (rather than declarative) and makes heavy use of aggregate operations across sets of data (rather than element by element manipulation).

However, design decisions ultimately determine which aspects of natural language can be interpreted by the computer, and the potential for users to incorrectly transfer existing knowledge shifts to other grammatical constructs. For example, Bruckman and Edwards (1999) analyzed programming errors made by children in MOOSECrossing, which features a syntax designed to resemble natural language. They discovered that about 10% of all errors were still attributable to natural-language transfer issues, and some 70% of these were the result of syntax errors or children making guesses about how to do things based on their knowledge of English. Though their results are encouraging, a fundamental challenge for natural language approaches remains—the need to balance support for the ambiguity and multiple forms of expression in natural language with the need for precision and specificity in formal grammars (Little & Miller, 2006).

In light of this, it is fitting that Nardi (1993) challenges the premise that “end users should be shielded from having to use . . . formal languages” (p. 27) and enumerates several examples in which people employ formal languages in daily life with ease (e.g., knitting instructions, baseball play tracking). Instead, she argues for highly domain-specific formal languages. Examples of such targeted domain-specific languages might include the Excel formula language, Wiki markup languages, and the ColdFusion Markup Language.

2.1.3 Automating Meta-Programming Tasks

The last approach I consider targets aspects of the programming task that are not necessarily tied to syntactic challenges. Along with writing program code, software development involves a number of other tasks. These include, but are not limited to, debugging, testing, and maintenance. Recognizing that end-user programmers also face challenges with these aspects, researchers have also begun exploring ways to facilitate these meta-programming tasks.

In particular, user testing of spreadsheet formulae has received considerable attention. Using the Forms/3 spreadsheet system (Burnett et al., 2001), Burnett et al. have been exploring ways to encourage users to fully test their code (Wilson et al., 2003). The system is designed as a collaboration between the software and the user. As a user creates interconnected formulae in a spreadsheet (i.e., when a formula uses cells that are the output of one or more other formulae as input), the system builds an internal model of the formula execution graph. This model contains all possible paths resulting from branches in the formulae (e.g., if expressions). The system then encourages users to test various input values and verify the output by highlighting cell borders with various colors representing the cell's "testedness" (Wilson et al., 2003). As the user verifies values, the system updates the internal model, eventually leading to a model that is completely verified by the user.

Approaches to automating meta-programming tasks are promising, but they may be hindered by the complexity of the programming task in general. The solution proposed within the spreadsheet domain is tractable within the limited problem space of the formula language, but this may not scale well to end-user programming tasks using more general scripting languages. It is well known that the formal modeling necessary to verify simple programs is cumbersome even for professional programmers. This is not to say, however, that verification systems for end-user programmers should be abandoned—they have demonstrated their effectiveness in some applications. Rather, I argue that extending these techniques to end-users writing code in languages like JavaScript is non-trivial.

2.2 Users as Learners

Stepping back from particular approaches, we might ask more broadly what is it that makes programming daunting for end-users in the first place. In a study of non-programmers learning to use Visual Basic.NET to create GUI applications, Ko et al. (2004) identified six types of learning barriers. These barriers are summarized below:

- **design:** inherent difficulties in conceptualizing a solution to the problem
- **selection:** issues locating and selecting the relevant programming components and interfaces from those made available by the system
- **coordination:** problems properly composing selected components into a workable solution
- **use:** properties of the programming interface that obscure an element's usage or its effect
- **understanding:** difficulties that arise as a result of a mismatch between a program's external behavior (e.g., at runtime) and learner expectations
- **information:** challenges caused by an inability to inspect a program's internal behavior to test learner hypotheses

The approaches presented in the previous section pose *technical* solutions to one or more of the six learning barriers posed by Ko et al. It is not surprising that, as much of this research has taken place within HCI, these solutions are highly influenced by the user-centered design (Norman, 1988) process. In fact, Ko et al.'s discussion of their learning barriers explicitly likens them to Norman's work and states "we can adapt Norman's recommendations on bridging gulfs of execution and evaluation to programming system design" (2004, p. 202). The solutions attempt to abstract away the intricacies of programming languages or automatically complete onerous programming chores, like testing, behind the scenes. The net effect is that the question answered by this stance is: "How can programming be made easier?"

While searching for answers to this question is no doubt useful, we might also ask a different question: "How can acquiring programming expertise be made easier?" If we recognize that Ko's barriers are analogous to the barriers that all novice programmers encounter (see e.g. Spohrer & Soloway, 1985; Du Boulay, 1989; Green & Payne, 1984; Felleisen, Findler, Flatt, & Krishnamurthi, 2004), we might suggest that the necessary solutions are pedagogical, rather than technological. The goal, then, is not to solely focus on creating novel programming interfaces, but to investigate ways to facilitate the informal learning about programming that takes place within end-user contexts. Rather than hide the complexity of the programming task, I seek to enable users to more easily learn about the complex aspects of the task that are relevant (i.e., worthwhile) to their goals.

This distinction between user-centered and learner-centered views is significant. As Soloway, Guzdial, and Hay note, "if addressing the needs of users is the driver, then it is natural to focus on ease of use; if addressing the needs of learners is the driver, then it is natural to focus on the development of understanding, performance, and expertise" (1994, p. 47). Supporting the learner-as-user requires a wholly different approach from traditional user-centered design (Guzdial, 1999). While end-user programmers tend to be experts in their primary domain (e.g., graphic design, accounting), they are simultaneously novices with respect to programming—they are learning about programming and computer science as they work. Central to this task should be resources that help them develop a more expert understanding of programming.

This dissertation focuses on this unexplored space in the end-user programming literature. I take the general-purpose programming language (JavaScript) used by web and graphic designers, for better or worse, as a given. My goal is to explore their challenges through a learner-centered lens. I seek to better understand what graphic/web design EUPers currently know about computing, their current learning strategies, and how to design relevant educational resources that enable users to extend their knowledge of the computing field. The remainder of this chapter explores relevant learning theories and pedagogic strategies.

2.3 Situated Learning

For end-user programmers, the nature of learning is a situated endeavor. They piece together their understanding of programming over time by interacting with tools and programming environments to solve specific problems, seeking information from books and online resources, and asking for assistance from other people (Dorn & Guzdial, 2006, 2010). Knowledge in this environment is distributed across the various components (Hutchins, 1995). Developing an understanding of what is learned and how it can be facilitated is a process of discovering and leveraging the successful activity patterns already in place (Greeno, Collins, & Resnick, 1996).

The situated view of learning has been applied in many learning scenarios. Lave and Wenger propose that learning occurs through legitimate peripheral participation within a community of practice (Lave & Wenger, 1991). The nature of the social contexts in which novices participate motivates and drives all learning. As a learner acquires skill, he or she becomes a more central participant in the community. Apprenticeship environments are the canonical examples of this learning, and Lave and Wenger illustrate the benefits of analyzing learning in this way by applying it to midwives, butchers, and members of Alcoholics Anonymous (Lave & Wenger, 1991).

Other studies using a situated lens have led to a deeper understanding of how skills like mathematics are used in real world environments (see e.g., Rogoff & Lave, 1999; Roth, 2005).

Central to the situated perspective on learning is that knowledge and skills are developed as a result of engagement in tasks that are meaningful. These tasks might relate to one's profession, one's hobbies, or other aspects of one's life. Guzdial and Tew (2006) argue that for instruction to be effective, it necessarily must be perceived as authentic to and closely aligned with such tasks (i.e., with the learner's community of practice (Lave & Wenger, 1991)). Thus, supporting learning among end-user programmers requires not only the development of instruction that matches practices in existing activity patterns, but also the creation of instructive content that relates topics to specific situations they recognize as relevant.

With this in mind, case-based approaches to instruction seem a natural fit. Prior research has suggested the importance of code examples in the informal learning strategies of end-user programmers (Dorn & Guzdial, 2006; Rosson, Ballin, & Rode, 2005), and I will present additional evidence to this point in Chapter 3. Case-based instructional materials would leverage EUPers' tendencies to seek out related examples while also presenting information situated in the context of meaningful tasks. In the following section, I provide an overview of case-based learning aids and present three theoretical perspectives that inform their use.

2.4 Case-Based Learning Aids

Case studies have been employed as a tool to promote learning in many educational settings. In a general sense, a case provides a narrative description of the process by which experts devise a solution to a given problem. Students learn from cases by studying their content, asking questions, making predictions, considering alternatives, and comparing them to other cases.

In computing, case studies have been used as a means to help foster both programming language knowledge acquisition and problem-solving skill development (Linn & Clancy, 1992). Clancy and Linn's *Designing Pascal Solutions: A Case Study Approach* provides a general structure for the presentation of programming cases and outlines a course-length set of examples for teaching computer science (Clancy & Linn, 1995). Each case includes a statement of the problem, a narrative description of an expert's solution, a complete code listing of the solution, a series of questions for the learner to consider about the case, and a number of related assessment questions. Empirical results suggest that the expert commentary about the solution is a key feature of cases that increases the amount one learns (Linn & Clancy, 1992).

Built on the concept of cases, Kolodner, Owensby, and Guzdial define a case-based learning aid as "a support that helps learners interpret, reflect on, and apply experiences . . . in such a way that valuable learning takes place" (2004, p. 829). In practice, these learning aids often take the form of case libraries (i.e., collections of cases) in order to enable learners to compare and contrast interrelated cases. This increases the likelihood that a learner will be able to infer the conditions under which the solution approaches are applicable (Bransford, Brown, & Cocking, 2000).

Such case-based learning aids have been designed and deployed in design domains like architecture and object-oriented programming. Archie (Pearce et al., 1992; Goel et al., 1991) and its descendant Archie-2 (Zimring, Do, Domeshek, & Kolodner, 1995) provided libraries of architecture case studies intended to support graduate students in architecture while creating new designs. For example, they contained cases that explored different ways of providing natural light into a building, helping students to consider alternatives to their design problems. In software design, Guzdial and Kehoe (1998) created STABLE (SmallTalk Apprenticeship-Based Learning Environment) as a resource for students in an object-oriented design course. It contained multiple correct solutions by previous undergraduate students in the same course on similar assignments. Evaluation results showed that students often used STABLE to find code fragments related to their current assignments, but in doing so, they gained new insights into object-oriented design (Guzdial & Kehoe, 1998).

The following three sections introduce case-based reasoning, cognitive load theory, and minimalist instruction. These educational models justify my application of case-based learning aids. In each section I outline the model and how it relates to cases for education.

2.4.1 Case-Based Reasoning

Case-Based Reasoning (CBR) was originally developed as a means to further reasoning abilities of expert systems implemented with computers (Kolodner, 1993). It leverages human tendencies to reference previous experiences when solving new problems. CBR systems allow computers to recall previous similar situations (i.e., cases), compare the prior circumstances to the current situation, and adapt the strategies used in the previous case. Additionally, the computer system stores the new problem's solution and any information learned in the process (e.g., failures in the adaptation, violations of expectations). The expert system learns as it builds a progressively larger and more thoroughly indexed case base by experiencing additional problem scenarios.

The CBR computational model has been appropriated by education and learning sciences researchers as a way to think about human cognition and the design of learning environments (Kolodner, 1997). This cognitive model has several direct implications for promoting effective learning (Kolodner et al., 2004):

1. Failure plays an important role in learning, and learners must receive feedback in order to identify holes in their understanding.

2. Explanation is crucial in case refinement, and learners should be supported to make predictions and to explain outcomes.

3. Case reuse is achieved by thorough indexing; learners need to reflect on their experiences in order to distill what is learned and the conditions under which it applies.

4. Knowledge is enhanced through a process of incremental case refinement, and learning occurs through repeated exposure.

5. Previous experiences provide the basis for reasoning and learners should be encouraged to reuse their own experiences and the experiences of others.

In providing direct access to the experiences of others, case libraries have a number of features that align well with CBR's suggestions (Kolodner et al., 2004). Their explanatory discussion of problem solutions can easily incorporate failure points and guidance for others in similar situations. The library's user interface can explicitly provide indexes of the library and link related cases together. Their structure, if well designed, provides an example for learners in how to organize their own knowledge and experiences.

2.4.2 Cognitive Load Theory and Worked Examples

Introduced by Sweller (1988), Cognitive Load Theory (CLT) seeks to inform and guide instructional design based on underlying cognitive architectures. CLT presumes a human cognitive model comprised of long term memory and working memory. Learning takes place as a result of human interactions with the environment (i.e., senses and actions). As a person makes sense of something, he/she recalls existing knowledge from long term memory and manipulates this knowledge along with new information in working memory. Finally, learning occurs when these knowledge structures, called schemas, are encoded and stored back in long term memory.

The challenge from an instructional design perspective is that short term working memory has a limited capacity. Sweller (1988) argues that when novices are given problem solving tasks, they often either lack or fail to recognize relevant schema in long term memory. In this situation, learners must resort to sub-optimal sense making strategies (e.g., means-ends analysis) that place a high degree of cognitive burden on working memory. When used in this way, less working memory is devoted to schema construction and learning efficiency decreases. Thus, CLT proponents posit that instructional environments and materials must be designed so as to mitigate demands on working memory.

Sweller and Cooper (1985) point to "worked examples" as instructional resources that are well-aligned with CLT. Put simply, a worked example is a description of the process by which a problem is solved focusing on various problem states and steps needed in the solution (Caspersen & Bennedsen, 2007).

Research results provide empirical support for using worked examples. Early experiments with students learning algebra indicated that students learning through interaction with worked examples indeed facilitated knowledge acquisition when compared to a control group using traditional methods (Sweller & Cooper, 1985). In a study of novice recursion instruction, Pirolli (1991) explored the use of instructor-provided examples. In addition to concluding that the examples aided learning, he discovered that examples written to explain how a solution was reached (how-it's-written) were more instructionally efficient than examples that explained how a solution works (how-it-works).

Cases are, by definition, a form of worked example. Further, as described here, case-based learning aids closely resemble collections of Pirolli's "how-it's-written" examples. A case's structure highlights the process by which the problem is solved and introduces the necessary abstract domain knowledge along the way. In a sense, these narratives externalize mental schemas employed by the author for direct inspection by novices. In theory, learning by studying cases reduces the amount of information which must be inferred by the learner about the problem solution, and should therefore reduce the burden on working memory.

2.4.3 Minimalist Instruction

The Minimalist model of instruction is the result of research from the mid 1980s that attempted to better assist users of early computer applications like word processors (Carroll, 1990). At its core, minimalist instruction emphasizes realistic scenarios in which users are learning by doing. Recognizing that people naturally seek information in order achieve a task, tutorials and documentation designed in the minimalist fashion organize and present information around activity. This is in contrast to other instructional design approaches that favor alternative concerns like logical decomposition and conceptual ordering.

Whether applied to simple tasks like word processing or complex tasks like object-oriented programming, minimalist instruction is driven by four design principles (Carroll, 1998):

1. Information should be action-oriented. Users should be given the opportunity to take meaningful action and should be encouraged to try things out for themselves.

2. Information should be anchored in the activity. Instructional activities should incorporate authentic tasks from the user's domain.

3. Instructional materials should support error recognition and recovery. Users should be presented with information about common errors as well as error diagnosis and recovery that pertains to the actions at hand.

4. Information should serve as scaffolding that promotes user independence.

Carroll and Rosson (2005) argue that case-based learning aids, when properly designed, can serve quite naturally as minimalist information sources. As narrative descriptions of solutions to problems, they are inherently oriented towards action and activity. “They provide guidance and encouragement for user action by describing specific activities, events, and problems from real world practice” (Carroll & Rosson, 2005, p. 4). At the same time they present necessary technical information as part of the broader narrative. Cases can be written to highlight common pitfalls and failures associated with the example scenario. Lastly, cases serve as explicit models of expert practice and exposure to multiple cases guides users towards autonomous action.

2.5 Chapter 2 Summary

In this chapter I have motivated the need to support end-user programmers in their software development efforts. I described three current approaches to this endeavor from recent literature: programming by demonstration, natural language systems, and automation of meta-programming tasks. I then argued for the need to view EUPers as novices engaged in activity-driven learning, rather than merely users of software tools. Recognizing that they often obtain their knowledge about programming beyond the confines of formal educational environments, I proposed case-based learning aids as a means to provide EUPers access to instruction about computing concepts.

CHAPTER III

CURRENT KNOWLEDGE AND LEARNING STRATEGIES OF END-USER PROGRAMMERS

Whether designing new programming languages, tools, or educational interventions, a thorough understanding of the target users or learners is required. There is considerable research about what traditional novice programmers do and do not understand (see e.g., Du Boulay, 1989; Green & Payne, 1984; Lewandowski, Gutschow, McCartney, Sanders, & Shinners-Kennedy, 2005; Sanders et al., 2005; Spohrer & Soloway, 1985), but these studies consider students in formal learning environments. End-user programmers and many people engaged in scripting activities often learn about scripting and programming without the aid of a classroom (Dorn & Guzdial, 2006). There is very little empirical data about how such non-traditionally trained programmers grasp conceptual computing knowledge. A detailed characterization of their understanding is necessary in order to appropriately design tools and resources that scaffold end-user development processes.

This chapter presents a study which provides a detailed depiction of what aspects of programming fundamentals one group of non-traditional software developers understand and how they go about learning.[1] This study is contextualized within the domain of professional web design and development, whose members make up a large and diverse group of end-user programmers. They regularly engage in programming activities, making use of textual markup and scripting languages like HTML, CSS, JavaScript, and PHP. In studying practicing web developers here, I am able to explore

[1]This chapter is based on the earlier work (Dorn & Guzdial, 2010): ©ACM, 2010. http://doi.acm.org/10.1145/1753326.1753430

notions of programming among a group of people who program in their careers but may lack a traditional educational background in computing. In this chapter I focus on the first two research questions posed in the introduction to this dissertation.

> **RQ1:** What is the nature of graphic/web design end-user programmers' knowledge of normative computing concepts?
>
> **RQ2:** What learning practices do graphic/web design end-user programmers currently employ, and to what extent do typical resources provide opportunities to learn about normative computing concepts?

More specifically, for this study I have operationalized these questions as follows.[2]

> **RQ1.1:** What programming concepts do web developers recognize, and to what degree do they understand each?
>
> **RQ1.2:** How do web developers think about and associate foundational programming concepts with one another?
>
> **RQ2.1:** What processes do web developers use to learn new programming concepts as they go about their work, and on what resources do they rely?

The remainder of this chapter begins with an outline of the study design and methods used. Then, I briefly discuss participant demographics and background. The primary results are presented in three sections. The first two sections focus on how participants categorized various computing concepts in a card sorting task. This is followed by a discussion of themes about learning derived from interview data. I conclude the chapter by highlighting the findings directly with respect to research questions.

[2]Note, the scope of RQ2 is not exhaustively covered by RQ2.1. Chapter 4 will address the later portion of this question.

3.1 Study Protocol

This study was conducted face to face and consisted of three separate parts. First, participants completed a survey that gathered basic demographic information and details about their professional background. Next, participants engaged in a card sorting activity about various introductory computer science concepts. Finally, I ended each session with a semi-structured interview. The sorting task and interview are discussed in more detail in the following subsections.

3.1.1 Card Sorting Task

Card sorting is a general purpose elicitation technique that can be applied in a wide range of settings (Rugg & McGeorge, 1997). At its most basic, it involves participants grouping items from a set of stimuli (e.g., pictures, words) into categories based on similarity along some dimension. Sorting tasks may be either *closed*, where participants are provided with the sort criteria and fixed categories in which to place the cards, or *open*, with participants developing their own criteria and categories. Through categorizing the physical cards in multiple ways, participants provide indications of their own mental representation of the concepts (Fincher & Tenenberg, 2005).

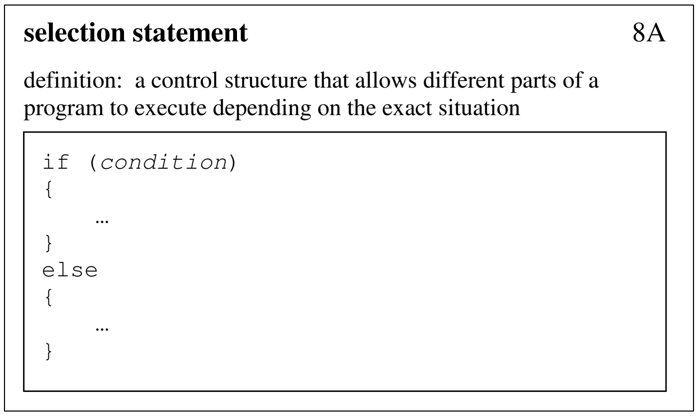
Card sorts are often employed in HCI as a usability tool for gaining an understanding about how users might naturally group certain aspects of a designed artifact (e.g., placement of content on web sites (Katsanos, Tselios, & Avouris, 2008)). However, categorization tasks also allow one to investigate a person's existing knowledge about the stimuli. Fincher and Tenenberg argue that card sorting "can be effective in eliciting our individual, and often semi-tacit, understanding about objects in the world and their relationships to one another" (2005, p. 90). Accordingly, computer science education researchers have successfully used card sorting to elicit novice programmers' knowledge of fundamental computing concepts with cards containing terms about programming (Lewandowski et al., 2005; Sanders et al., 2005).

Building on Sanders et al.'s (2005) work, I used card sorting to explore web designers' and developers' knowledge of introductory computing concepts. I developed a set of 26 cards containing terms from Sanders et al.'s study as well as terms from another study I conducted to explore common introductory constructs found in an online repository of scripting code (this study will be described in detail in Chapter 4). After merging the two lists, I removed any duplicated terms and eliminated terms that lacked concreteness or relevance to the web programming domain (e.g., dependency, thread). That is, I ensured the list of terms had clear syntactic representations in JavaScript. The final cards contained the concepts listed below, with one term per card.[3]

- input
- output
- mathematical operator
- relational operator
- logical operator
- function
- importing code
- functional decomposition
- object
- exporting code
- assignment
- variable
- parameters
- variable scope
- constant
- selection statement
- nesting control structures
- definite loop
- indefinite loop
- recursion
- exception handling
- number
- boolean
- string
- array
- type conversion

The task consisted of a repeated single-criterion card sort with both open and

[3]The formatting of the actual cards used for the study can be found in Appendix A.


selection statement 8A

definition: a control structure that allows different parts of a program to execute depending on the exact situation

```
if (condition)
{
    …
}
else
{
    …
}
```

Figure 2: Example Card with Definitions

closed sorts. The first four sorts were explicitly prompted by me; these closed sorts explicitly explored participants' recognition and understanding of the 26 terms. The first sort is particularly notable in that it asked participants to separate the cards based on whether they recognized the term or not. Because I sought to explore participants' understanding of the underlying concepts and not simply vocabulary recognition, I had them repeat the sort for any cards originally placed in the "don't recognize" category. In this extra sort, I provided cards that contained the unfamiliar term, its definition, and a JavaScript example of the concept in use (see Figure 2; all of the cards with definitions are available in Appendix A). Definitions were drawn from the glossaries of introductory textbooks (Horstmann, 2006; Lewis & Loftus, 2005; Zelle, 2004) and adapted where necessary to fit JavaScript. Any concepts recognized with the aid of this additional information were added to the participant's "recognize" category, and any that remained unknown were eliminated from all subsequent sorts. Following the four closed sorts, participants were invited to openly sort the cards using one criterion at a time in as many ways as they could generate.

3.1.2 Interview

Once participants had exhausted their ideas for additional open sorts, I conducted a semi-structured interview that lasted approximately 30 minutes. The interview elicited information about participants' daily job responsibilities, use of programming or scripting languages, and use of software tools (e.g., Photoshop). I also inquired about typical strategies they employ while developing scripts and resources they rely on to learn new things about programming.

Audio recordings of the interviews were transcribed, and I used a multi-step thematic analysis to analyze the qualitative data. Thematic analysis is a qualitative analytic method that aims to provide a rich and detailed account of the data collected (Braun & Clarke, 2006). The end result of a thematic analysis is a collection of themes based on common patterns observed in the data (e.g., interview transcripts). It is important to note that while themes are necessarily repeated by various participants throughout the interview corpus, it is not the goal of such an analysis to convey the prevalence or relative importance of the themes.

Coding was done in both a top-down and bottom-up fashion. I coded transcripts based on particular questions asked of all participants but also allowed for emergent codes when other themes were mentioned by multiple participants. Additional passes were made through the transcripts to further refine the codes. Lastly, in preparing transcript excerpts for presentation in this dissertation, I have edited them as necessary for anonymity and brevity.

3.1.3 Recruitment

Participants were recruited from a large metropolitan area via email. Solicitation messages for volunteers were sent primarily to mailing lists for three large Meetup[4] groups of local web designers, graphic designers, and users of Adobe Photoshop. Volunteers were then pre-screened using a short email survey to ensure that they were actively involved in the web design profession and had prior experience with writing scripts or programs in JavaScript. Face to face interviews were scheduled

[4]`http://www.meetup.com/`

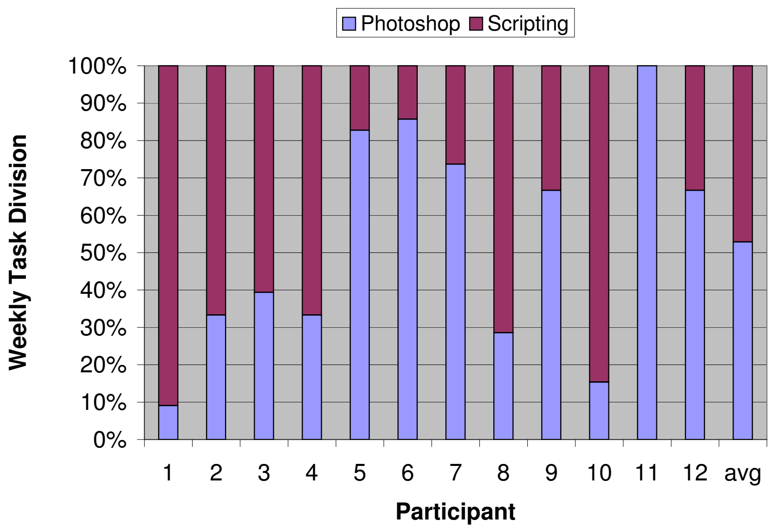


Figure 3: Self-Reported Average Weekly Division of Labor

with participants meeting the study criterion, and participants were compensated $15 for their time.

3.2 Participant Demographics and Background

In total, I interviewed 12 people—seven men and five women. Ten participants indicated on the survey that they actively work in the web design field, and the remaining two were students currently enrolled in web design degree programs at local institutions. Most participants (58%) selected "Web Developer" as their job title with only two people choosing the title "Programmer."

The participants were well educated. All but two participants (one of whom was a current student) held a bachelor's degree, and four participants either had earned or were pursuing a master's degree. However only one person held a degree in computer science. About a third of them held undergraduate degrees in areas related to web or graphic design (e.g., visual communications) with the rest holding degrees in the humanities or other fields (e.g., English, psychology, ministry).

I was successful in recruiting broadly from the web design/development community with respect to reaching those with a wide range of professional experience. The number of years of experience with Photoshop ranged from 2 to 13 years, and participants reported between 2 months and 15 years of scripting or programming language experience. On a scale of 1 (novice) to 5 (expert), the average self reported level of expertise in scripting was 3.21 ($\sigma = 1.16$). Every participant listed exposure to more than one scripting or programming language with JavaScript, ActionScript, and PHP being those most frequently mentioned. Figure 3 illustrates participants' estimated weekly division of labor between scripting and graphics manipulation in Photoshop. On average, my participants' time was split roughly evenly between these two tasks; however most people tended to concentrate more heavily on one or the other.

When elaborating on the nature of their work, most participants noted being involved in front-end web development or design. This job often requires them to build functional web sites from prototypes that have been mocked up with tools like Photoshop that either they or someone else designed. They make decisions about how to slice up visual components in the layout so that they render properly across different browsers, and they write scripts using languages like JavaScript or PHP to enable the intended interactivity features in the design.

3.3 Closed Sort Results

As mentioned earlier, I asked participants to first sort the 26 programming concepts into two piles based on whether or not they recognized the term. The results of this initial sort are presented in Table 1 with concepts ordered by their level of recognition. Despite the lack of what might be considered a traditional computing education, a majority of participants recognized nearly all concepts, and ten of the concepts were universally recognized based on the term alone. The least frequently recognized terms were selection statement, nesting control structures, and functional decomposition, with the last being markedly less familiar than all others.

Once provided with cards containing definitions and examples of the concepts, the

Table 1: Percentage of Participants Recognizing Card Concepts

CS Concept	**Term Only**	**With Def'n**
assignment	100.0%	
input	100.0%	
object	100.0%	
function	100.0%	
parameters	100.0%	
array	100.0%	
string	100.0%	
output	100.0%	
number	100.0%	
variable	100.0%	
mathematical operator	91.7%	100.0%
definite loop	83.3%	100.0%
importing code	83.3%	100.0%
indefinite loop	83.3%	100.0%
boolean	83.3%	91.7%
constant	83.3%	91.7%
exception handling	83.3%	83.3%
type conversion	75.0%	91.7%
exporting code	75.0%	83.3%
logical operator	75.0%	83.3%
relational operator	66.7%	91.7%
variable scope	66.7%	91.7%
recursion	66.7%	83.3%
selection statement	58.3%	91.7%
nesting control structures	58.3%	83.3%
functional decomposition	8.3%	83.3%

rate of recognition increased significantly (paired t-test; $t(25) = -3.696$, $p = 0.001$). Over half of the concepts were then familiar to everyone, and the minimum recognition rate increased to 83.3% with this additional information. Many participants commented on the fact that they did use these concepts often, but they had not initially recognized the terms simply because they had never learned the names. For example:

> P6: Mathematical operator, goodness gracious. [laughter] I've just never heard it called that before. Plus, minus, sure.
>
> P9: Um, some of them I picked up by seeing the code. I just didn't know the name of it, like nesting control structures. You know, putting if statements and when statements inside each other is common practice in code, but I just had never given it a name.

3.3.1 Additional Closed Sorts

After participants identified the subset of concepts they recognized, I prompted them with three additional sorts. Participants sorted the concepts based on their own level of understanding of the concept, based on how often they use the concepts in scripts, and based on how difficult they perceive the concepts are to learn. These closed sorts were intended to provide additional information about conceptual understanding beyond simple concept recognition. Results of these sorts are summarized in Tables 2, 3, and 4, respectively. In each table, concepts are sorted by a "rating" value which is computed as a weighted average of the response frequency across the ordered categories. For example, in Table 2 categories are assigned values between one and four (similar to a Likert-type scale) and the rating corresponds to the average value that participants assigned to this concept. Further, each of these tables divide the upper, middle, and bottom thirds of the rating values with a double line.

Based on their sorting results, participants reported considerable understanding

Table 2: Personal Level of Understanding; Sorted by Decreasing Understanding

CS Concept	Rating (1–4)	1	2	3	4
number	3.92			8.3%	91.7%
boolean	3.91			9.1%	90.9%
variable	3.83			16.7%	83.3%
mathematical operator	3.75		8.3%	8.3%	83.3%
function	3.75		8.3%	8.3%	83.3%
array	3.75		8.3%	8.3%	83.3%
object	3.75			25.0%	75.0%
selection statement	3.73	9.1%			90.9%
nesting control structures	3.70			30.0%	70.0%
string	3.67		8.3%	16.7%	75.0%
parameters	3.67		8.3%	16.7%	75.0%
input	3.67		8.3%	16.7%	75.0%
definite loop	3.67			33.3%	66.7%
relational operator	3.64	9.1%		9.1%	81.8%
constant	3.64		18.2%		81.8%
output	3.58		16.7%	8.3%	75.0%
importing code	3.58		16.7%	8.3%	75.0%
logical operator	3.50	10.0%		20.0%	70.0%
assignment	3.50	8.3%		25.0%	66.7%
exporting code	3.50		10.0%	30.0%	60.0%
variable scope	3.45	9.1%	9.1%	9.1%	72.7%
type conversion	3.45	9.1%		27.3%	63.6%
indefinite loop	3.42		16.7%	25.0%	58.3%
exception handling	3.40	10.0%	10.0%	10.0%	70.0%
functional decomposition	3.40			60.0%	40.0%
recursion	2.90	20.0%	20.0%	10.0%	50.0%

of and comfort with these concepts. Table 2 shows the rating value for each concept as well as a breakdown of participants' self-assessed level of understanding using the categories:

1. I have heard the term but am not comfortable using it in my scripts.
2. I understand the meaning of the term but have problems using it correctly in my scripts.
3. I understand the meaning of the term and am comfortable using it in my scripts.
4. I have a strong understanding of the term and feel I could explain it to someone else.

With the exception of recursion, every concept had a rating of 3 or higher, meaning that participants understood the term's meaning and were comfortable using it. The top two-thirds of topics rated 3.58 or higher, indicating a high degree of knowledge and an ability to explain the concepts to others. Among the lowest ranked terms were variable scope, type conversion, indefinite looping, exception handling, functional decomposition, and recursion. Interestingly, concepts where individual participants indicated trouble were spread throughout the table and were not localized to the bottom third, where one might expect.

I also asked participants to sort the cards into four categories depending on how frequently the concepts were used in code that they wrote. The four categories provided were Never (1), Rarely (2), Occasionally (3), and Frequently (4). Table 3 presents the results of this sort, ordered by decreasing frequency of use. Unlike the sort on understanding, the distribution of responses was fairly uniform across the three tiers—no concept in the top two thirds was placed in the "never" category, and the six lowest ranking terms were all categorized as frequently used by 50% or fewer of the participants. In other words, these results indicate a reasonably strong consensus

Table 3: Frequency of Concept Use; Sorted by Decreasing Frequency

CS Concept	Rating	Never	Rarely	Occasionally	Frequently
number	3.92			8.3%	91.7%
string	3.92			8.3%	91.7%
relational operator	3.91			9.1%	90.9%
selection statement	3.91			9.1%	90.9%
boolean	3.91			9.1%	90.9%
logical operator	3.90			10.0%	90.0%
variable	3.83		8.3%		91.7%
mathematical operator	3.83		8.3%		91.7%
array	3.75		8.3%	8.3%	83.3%
object	3.75		8.3%	8.3%	83.3%
definite loop	3.75		8.3%	8.3%	83.3%
parameters	3.73		9.1%	9.1 %	81.8%
nesting control structures	3.70		10.0%	10.0%	80.0%
assignment	3.67		16.7%		83.3%
input	3.67		8.3%	16.7%	75.0%
output	3.67			33.3%	66.7%
function	3.67			33.3%	66.7%
importing code	3.45		9.1%	36.4%	54.6%
functional decomposition	3.40		20.0%	20.0%	60.0%
variable scope	3.36	9.1%	9.1%	18.2%	63.64%
constant	3.18		27.3%	27.3%	45.5%
exporting code	3.10	20.0%		30.0%	50.0%
exception handling	3.10	20.0%		30.0%	50.0%
type conversion	3.09		18.2%	54.6%	27.3%
indefinite loop	3.08		33.3%	25.0%	41.7%
recursion	2.70	10.0%	20.0%	60.0%	10.0%

about how frequently these programming concepts arise in typical web development work. The topics listed in the top third (number to mathematical operator) are frequently used, and those in the bottom third (importing code to recursion) are used sporadically.

The final closed sort requested of participants was to categorize their perception of how difficult the concepts are to learn to use correctly. They were prompted with three categories for this sort: Easy (1), Intermediate (2), and Advanced (3). Responses on this sort are outlined in Table 4 and are sorted by increasing level of difficulty. This sort exhibited the lowest agreement with 73% of the individual concepts being ranked in all three categories (easy, intermediate, and advanced) by different people. Despite this variation, the final ordering of concepts maps relatively closely to what one might find in an introductory textbook table of contents: basic data types and operators; followed by selection statements and functions; followed by looping, recursion, and exceptions.

3.3.2 Comparison of Closed Sorts

Comparing the results from Tables 2–4, I observed that many concepts appeared to be similarly rated in each of the three sorts. Indeed, a Pearson correlation analysis revealed a statistically significant positive correlation between the ratings for level of understanding and frequency of use ($r = 0.808$, $N = 26$, $p < 0.001$). I also noted statistically significant negative correlations between ratings for frequency of use and learning difficulty ($r = -0.641$, $N = 26$, $p < 0.001$) and between difficulty and understanding ($r = -0.586$, $N = 26$, $p = 0.002$).

Further I compared the terms with respect to their relative grouping in the tiers of the three sorts. This provided an indication for the concepts that were uniformly ranked in terms of the participants' level of understanding, the frequency with which

Table 4: Concept Difficulty to Learn; Sorted by Increasing Difficulty

CS Concept	Rating	Easy	Intermediate	Advanced
number	1.08	91.7%	8.3%	
boolean	1.09	90.9%	9.1%	
relational operator	1.09	90.9%	9.1%	
variable	1.17	91.7%		8.3%
constant	1.18	81.8%	18.2%	
logical operator	1.20	80.0%	20.0%	
string	1.25	83.3%	8.3%	8.3%
mathematical operator	1.33	75.0%	16.7%	8.3%
input	1.33	75.0%	16.7%	8.3%
assignment	1.42	66.7%	25.0%	8.3%
parameters	1.50	58.3%	33.3%	8.3%
selection statement	1.55	45.5%	54.6%	
output	1.58	58.3%	25.0%	16.7%
importing code	1.67	50.0%	33.3%	16.7%
function	1.75	33.3%	58.3%	8.3%
type conversion	1.82	27.3%	63.6%	9.1%
nesting control structures	1.90	40.0%	30.0%	30.0%
array	1.92	25.0%	58.3%	16.7%
exporting code	2.00	30.0%	40.0%	30.0%
object	2.00	33.3%	33.3%	33.3%
definite loop	2.08	16.7%	58.3%	25.0%
indefinite loop	2.08	16.7%	58.3%	25.0%
variable scope	2.09	27.3%	36.4%	36.4%
recursion	2.20	20.0%	40.0%	40.0%
functional decomposition	2.30	20.0%	30.0%	50.0%
exception handling	2.50	10.0%	30.0%	60.0%

they are used, and the perceived conceptual difficulty. I noted four terms that consistently appeared in the first tier, two in the second, and six in the bottom tier. The concepts that were rated the most highly understood, most frequently used, and easiest to learn were number, boolean, variable, and mathematical operator. Inversely, those which ranked least understood, least used, and most difficult were exporting code, indefinite loop, variable scope, recursion, functional decomposition, and exception handling. The concepts parameters and output were consistently in the middle tier.

3.4 Open Sorting

Once participants had completed the final closed sort, I provided them with the opportunity to sort the cards into groups using criteria of their own choosing. Through open sorting, I aimed to gather additional insight about web developers' knowledge of these 26 concepts and their associations between concepts. Participants were encouraged to generate as many sorts as they could and I recorded the participant's sort criterion, category names, and placement of the cards within the groups. Altogether, the participants generated 28 sorts. With an average of 2.3 sorts per participant, they generated noticeably fewer sorts than introductory computing students or educators engaged in a similar task (4.5 and 5.2, respectively) (Sanders et al., 2005). However, given that these participants completed a number of closed sorts prior to open sorting, this value may be artificially low. I also noted that the participants used fewer categories per sort on average (2.6) than the students (4.0) or educators (3.7).

To further explore the data gathered from the open sort activity I employed superordinate analysis to classify similar sorts into thematic groups (Rugg & Petre, 2007). These groups bring together sorts that relate to a common theme, regardless of differences in the wording that participants used to describe them. The purpose of such an analysis is to determine commonalities in sorts across the participants, indicating the typical ways people think about this particular set of stimuli.

Two independent raters grouped the 28 sorts into mutually exclusive categories based on the similarity of their criterion. To aid in making decisions about whether two sorts were similar, raters had access to the criteria and category names given for a sort by the participant as well as the excerpt of the interview transcript relevant to each open sort. Transcripts enabled raters to make an informed decision about a sort's meaning, particularly in the case where participants had difficulty in succinctly naming their sort criterion but were able to talk generally about what they were trying to accomplish with the sort. Raters achieved 79% agreement on the thematic grouping of the 28 sorts on their first pass. They then collaboratively negotiated the group definitions relevant to the six sorts where there was initial disagreement. In the end, seven thematic groups which each contained more than one sort were derived. These themes were (the number of sorts related to each theme appear in parenthesis):

Conceptual Ordering (4) Sorts which classify concepts by the order in which they should be learned or the order in which concepts build on one another.

Quality Metrics (4) Sorts which separate concepts by various software quality metrics like readability, maintainability, and efficiency of code. For example, concepts believed to slow down code execution might be placed in a category called "inefficient" while others are placed in an "efficient" category.

Terminology (3) Sorts which classify cards by terminology considerations. For example, a sort whose categories are labeled "terms you need to know to communicate with others" and "terms that are academic."

Language Decomposition (3) Sorts which attempt to separate concepts into functional groups based on their semantics (e.g., "related to functions" or "related to numbers").

Expertise of Others (3) Sorts expressing beliefs about the expertise or understanding of others. This often appeared when participants believed their peers might sort the cards differently if asked to rate their expertise.

Relevance to Scripting (2) Sorts that distinguish concepts based on whether they are generally applicable to the typical code or scripts that web developers write.

Desire to Know More (2) Sorts that prioritize concepts by an interest in learning more about them.

The results of the card sorting task provide a detailed picture about what computing concepts web developers understand and how they relate the concepts to one another, but they provide little information about how professional web developers learn as they go about their work. For this, we must turn to the qualitative data presented in the next section.

3.5 Learning and Resources

The primary focus of this semi-structured interview with participants was to elicit their strategies for learning new information. My analysis of interview transcripts resulted in four themes related to learning: motivation to learn new things, learning processes, resources used for learning, and heuristics for judging information quality. Each of these themes is discussed in the following subsections.

3.5.1 Impetus for Learning

While some participants indicated that they enjoyed learning new languages or details about scripting for curiosity's sake, most expressed that their decision to learn something was a matter of necessity. The computing concepts that they chose to learn needed to contribute in some way to the completion of their current project (in a similar fashion to Blackwell's (2002) attention investment model). Incorporation of new web features like login-based access or embedded streaming video (and learning the necessary underlying programming skills) were driven by project needs. Learning of new features was also motivated by a need to remain up-to-date in order to write standards-compliant code. Participants two and five discuss their reasons for learning new things below.

> P2: I don't care where technology is going. It's like, does my check get cashed on Friday? Ok. And if they have a new something that comes out that will impede my check being cashed on Friday, then I will learn it.
>
> P5: Like when CSS was officially considered a standard, and I went, oh crap, now I have to learn it.

Even among those who discussed learning new languages or language features for fun, they often did so by choosing to use the unfamiliar concepts in an upcoming project. In these cases the participants were willing to tolerate some inefficiency in completing the project because they recognized they were learning something new.

3.5.2 Learning Processes

Participant nine succinctly conveys his learning process, and that expressed by most participants, by stating, "generally, the best way I learn is to just jump in headfirst." Several participants used the phrase "trial-and-error" to characterize their script development. When asked to elaborate they described a process akin to bricolage programming (Turkle & Papert, 1991), iteratively writing code, examining the results, and seeking out information as necessary. In this way, the participants exemplified the opportunistic approach to programming described by Brandt, Guo, Lewenstein, Dontcheva, and Klemmer (2009). One participant explains:

> P1: I start off actually trying to do something that I need to complete as my first step even not knowing anything about it. And I guess the first thing that I'll do is I'll Google the subject and see what I can pull out on

> the Web. What information I can get out of it. And then I just hit the floor running, or at least I try. And then of course I come to points where I stumble, and I can't go forward cause it's too complex there's just some stuff that I don't know. So at that point I have a couple of choices.

He goes on to describe his decision making process for what to do when web resources are not enough—whether to consult with a colleague for help or search for a book.

However, while going to the Web to look for an answer was almost universally the first line of defense, it may not always be the most fruitful activity. One participant realized that this strategy was suboptimal while reflecting on the sources of information she used and which were the most useful in answering her questions.

> P2: The Internet, of course you can Google anything, that's my number one place. And it's fairly useful. Wow, that's a good question. The order in which I tap my resources are from least useful to most useful. So my colleagues are my second level, cause you know, different companies you work at have different systems. So something might work good in practice or I might find it on Google, but it just doesn't work well with servers and the software we use. So it would be Internet, colleagues, books as far as the order that I tap my resources. The most useful would be books, colleagues, Internet.

3.5.3 Resources Used

Over the course of the interviews, participants mentioned relying on over a dozen different resources for learning something new. In addition to generic occurrences of "the Internet" or "Google", seven different online resources were discussed by different participants. I also noted seven offline resources. Table 5 summarizes these 14 unique sources.

Table 5: Resources for Learning

Online	**Offline**
• code samples or example demos • walkthroughs and tutorials (e.g., `www.w3schools.com`, `www.smashingmagazine.com`) • language or library references (e.g., `www.ruby-doc.org`) • subscription-based online training sites (e.g., `www.lynda.com`) • forums or user groups • blogs, both as authors and as readers • podcasts	• books • code samples • tutorials or other help files provided with software • manuals • colleagues, friends, or instructors • strangers with similar job descriptions (e.g., other webmasters) • classes

To provide greater detail about these web developers' use of resources, I included an extra question at the end of the demographic survey. This sequence of prompts was replicated from Rosson et al.'s (2005) study of web developers. It asked participants to rate how likely they would be to consult various resources when attempting something new on a scale of 1 (very unlikely) to 5 (very likely). Specifically, I inquired about interactive wizards, example code, classes/seminars, books, FAQs/tutorials/online documentation, friend/coworker, and technical support. Figure 4 depicts the percentage of participants rating each resource as likely or very likely to consult. Similar to Rosson et al.'s findings, the participants indicated a strong preference for online documentation, books, examples, and personal communication.

3.5.4 Judging Relevance

The final theme related to learning deals with how web developers judge the quality and relevance of content they find online. The breadth of web content can be a

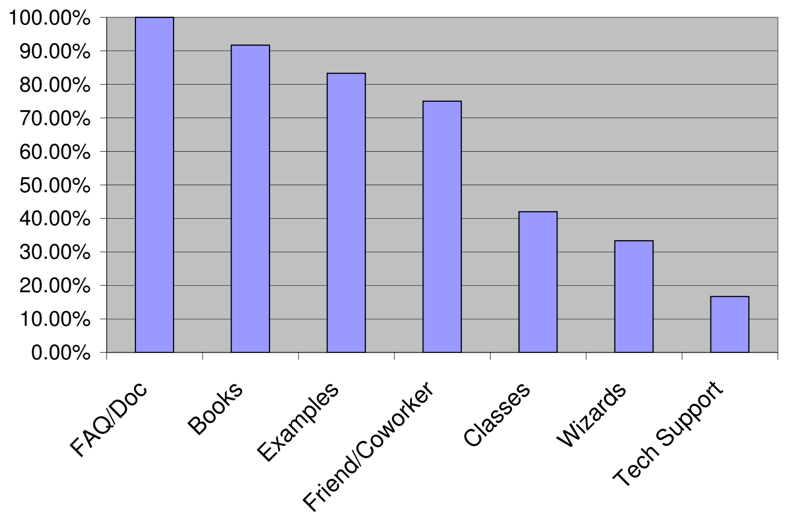


Figure 4: Percent of Participants Rating Resource as Likely or Very Likely to Use

double-edged sword; on the one hand, chances are good that an answer to one's question exists online, but on the other hand, locating that information can be time consuming. One participant reflects on her ability to find relevant information:

> P11: I'm starting to learn where those online resources are, but early on here it's been kinda daunting to figure out. I've done a lot of online that just takes me nowhere. It's, you know, spend an hour just clicking around trying to figure out where to find the answer at.

Though some participants were unable to articulate specific strategies they use to evaluate information, relying on their "gut reaction", many outlined informal heuristics that they employ while searching the Web. I noted 10 such rules of thumb in the interview corpus, ranging from some that are rather specific to others which could be highly subjective:

1. Legitimacy of sources, with a preference for content hosted by established or official publishers

2. Author credentials, preferring people recognized within the web development community

3. Google's page rank algorithm as a predictor of utility
4. Conformity of provided code to W3C standards
5. Availability of working code demos
6. Similarity of language features used in code examples to those used in the participant's code
7. Positive and negative comments of others posted in reply to tutorial, blog, or forum entries
8. Opinion of peers about a particular source of information
9. "Digestibility" of the information, with a preference for more easily consumable things (this notion was a combination of how well written and succinct the content was, while preserving the necessary details for understanding)
10. Overall aesthetic feel of the hosting web site

If not always sophisticated, these heuristics provide evidence that web developers do develop meta-cognitive strategies for evaluating information. They also have ramifications for how educational content might be delivered to end-user programmers via the Web. In the next section I discuss this and other implications of these findings.

3.6 Discussion

The discussion here is divided into two parts. First I interpret my findings relative to each of the three operationalized questions in turn. Then I consider high-level implications that play a role in shaping the informal instruction environment (i.e., ScriptABLE) that will be introduced later in Chapter 5.

3.6.1 Findings from the Research Questions

3.6.1.1 What programming concepts do web developers recognize, and to what degree do they understand each? (RQ1.1)

On the whole, the participants recognized nearly all of the concepts *with the aid of a definition and example.* The terms used in this study were standard terminology from introductory materials, but several participants lacked knowledge of the formal names for these concepts. Thus, these results suggest the importance of multiple indexes in reference materials which target informal learners. I also found that participants expressed remarkably normative judgments about concept difficulty; in many ways the average ratings matched what we might expect from a computer scientist.

3.6.1.2 How do web developers think about and associate foundational programming concepts with one another? (RQ1.2)

The open sorting data provides some insight into how web developers' associations may differ from other populations. When compared to Sanders et al.'s (2005) card sort study with novice computer science students, web developers generated fewer sorts per person with fewer categories per sort. This suggests that introductory CS students may have a more sophisticated understanding of these concepts than web developers.

The common open sort criteria noted in this study also support this interpretation. While not necessarily organized by natural language groupings (as noted in a previous study of typical novice programmers (McKeithen, Reitman, Rueter, & Hirtle, 1981)), only one of the seven sort themes I identified involved grouping cards based on programming language semantics. Contrastingly, the most frequently occurring category groupings generated by introductory CS students all appear to make use of programming syntax or semantic concerns (Sanders et al., 2005). The participants in this study appeared to associate concepts much more frequently based on pragmatic dimensions related to their day to day use of the concepts.

3.6.1.3 What processes do web developers use to learn new programming concepts as they go about their work, and on what resources do they rely? (RQ2.1)

I found that learning in this context is often motivated by project demands, whether that be a need to learn a specific new technique or to update one's skill set to continue to write standards-compliant code. Participants expressed a trial and error approach to programming where writing code is interleaved with information foraging. I found that participants learned from a wide variety of online and offline resources, with a preference for FAQ-style documentation, books, and related code examples.

3.6.2 Implications Moving Forward

The closed sorting data exhibited a strong correlation between frequency of use and concept difficulty. Further I noted that these web developers choose to learn concepts they perceive to be directly related to their tasks. Taken together they seem to suggest that web developers are learning those concepts that are either the easiest or the most useful. While perhaps not surprising, web developers and other end-user programmers may be missing out on more advanced concepts that could be quite useful but are not entirely obvious to them. For example, concepts like indefinite loop, exception handling, and program decomposition were uniformly ranked at the bottom of the sorts, but use of these constructs could easily aid these programmers in developing more robust, reusable software.

The results presented in this chapter confirm a reliance on resources like tutorials and example code, often found through web searches. As I argued in Chapter 2, these practices closely match the affordances of case-based learning aids.

3.7 Chapter 3 Summary

In this chapter I have presented the results of a study of 12 web designers and developers. Through this analysis of card sorting data I have contributed the first detailed depiction of this group of non-traditional programmers' understanding of foundational programming concepts. My qualitative results provided additional evidence in support of models of opportunistic programming, and I further elaborated on the common resources web developers seek out in order to learn something new.

CHAPTER IV

EXAMINING THE CONCEPTUAL COVERAGE OF A SCRIPT REPOSITORY

The previous chapter highlights a number of normative computing concepts that were consistently rated as difficult, not well understood, and not frequently used by graphic and web designers who program. Furthermore, Chapter 3 illustrated the important role that example code found online plays in these designers learning processes. Thus it is natural to wonder about the nature of the code examples that these end-user programmers come across. For example, does the code found online illustrate examples of how the difficult or misunderstood concepts could be used or does it only reinforce the concepts which are already frequently used?

This chapter provides a detailed analysis of the JavaScript code contained in one repository of scripts for Photoshop. In particular, this is a study of the conceptual coverage of introductory computing concepts within a corpus of scripting projects.[1] Such an analysis is important in fully understanding the strengths and weaknesses of this popular form of support.

This analysis seeks to address the latter portion of the research question two through the more specific operationalization of this question given in RQ2.2.

> **RQ2:** What learning practices do graphic/web design end-user programmers currently employ, and to what extent do typical resources provide opportunities to learn about normative computing concepts?
>
> **RQ2.2:** To what extent does code found online provide relevant examples

[1]This chapter is based on the earlier work (Dorn et al., 2007): ©IEEE, 2007. `http://dx.doi.org/10.1109/VLHCC.2007.35`

of introductory programming constructs?

The balance of this chapter proceeds as follows. I outline details of the method used for this study in Section 4.1. I then provide an overview of the results in Section 4.2, with a detailed discussion of patterns in construct use following in Section 4.3. I conclude the chapter by revisiting research question 2.2 and discussing implications of the results for the design of future example-driven resources.

4.1 Method

I conducted an artifact analysis of all scripts publicly available for download in the Photoshop scripting section of the Adobe Exchange repository[2]. As a community-driven site officially hosted by the company responsible for the product, it has immediate credibility and is a natural source of support for users seeking information. Also, I illustrated in Chapter 3 that these qualities are important characteristics in determining whether a resource is considered relevant or not for end-user programmers. To focus this analysis, I only considered scripts that were hosted in the Adobe forum directly and did not include forum contributions that referenced scripts hosted on other sites.

4.1.1 Development of Coding Scheme

To analyze the contents of these scripts, I developed a coding scheme that considered both general introductory computing constructs as well as EUP domain specific constructs. The computing constructs were informed by the computing education literature, while the domain specific constructs were suggested by end-user programming studies and derived in a data-driven manner by the scripts themselves.

The first set of constructs draws on Tew and Guzdial's (2010) effort to develop a language independent assessment of introductory computing concepts.[3] They have

[2]All files retrieved November 30, 2006 from `http://share.studio.adobe.com`

[3]While the analysis described in this chapter chronologically precedes the publication by Tew and

variable	selection (`if`)
mathematical operators	definite loop (`for`)
relational operators	indefinite loop (`while`)
logical operators	nested loops
assignment	recursion
number	user defined functions
boolean	user defined objects
string	user input
array	output to user

Figure 5: Textbook-Based Coding Elements

identified a set of computing constructs that are common across introductory courses which avoids bias from any particular language or pedagogical approach. They conducted an analysis of the table of contents of the top two CS1 textbooks as identified by each of the major publishers—12 books in total. This list of concepts was revised using the framework of the Computer Science volume of Computing Curricula 2001 (The Joint Task Force on Computing Curricula, 2001) as an organizing principle. It was further refined by analyzing the content of canonical texts representing each of the common introductory approaches (objects-first (Deitel & Deitel, 2005; Lewis & Loftus, 2005), functional-first (Felleisen, Findler, Flatt, & Krishnamurthi, 2001), and imperative-first (Zelle, 2004)). A construct was included in their list if it was covered by all of the texts or excluded by only one of the texts. Their analysis yielded 27 total constructs, and I used this set as a starting point for my coding scheme.

It was necessary to modify and extend the original set of constructs considering the EUP domain in this study. Some concepts were not relevant or practical in the domain (e.g., class-based objects and inheritance), others needed slight modification due to the particulars of JavaScript. The resulting computing constructs included in the coding scheme are listed in Figure 5.

Guzdial (2010), the work was informed by Tew's ongoing efforts at the time (personal communication, November, 2006). I have chosen to cite their 2010 paper here as it provides the best reference for this particular work.

Most of these constructs in JavaScript are similar to their counterparts in general-purpose computing languages. However, a few warrant additional explanation. The "number" coding element included use of any kind of numeric literal as JavaScript does not distinguish between types of numerics (e.g., integer, floating point). In the realm of web and graphic design scripting, user input and output is inherently graphical in nature. As such, the I/O constructs in the coding scheme included input dialogs and message boxes. Lastly, since the nature of Photoshop scripting considered here almost always requires calling of functions and using objects from the API, I limited my scope to instances of user-defined functions and objects. User-defined functions had to be explicitly defined and named, and user-defined objects had to include a constructor and be instantiable.

To fully analyze EUP scripts, it was important to supplement the general introductory computing concepts with a few domain specific ones. Previous studies of end-user programming practices (Dorn & Guzdial, 2006; Rosson et al., 2005; Scaffidi, Ko, Myers, & Shaw, 2006) suggested that intellectual property and code modularity could be important considerations in this domain. I added three items to the coding scheme (copyright notice, end-user license agreement, and credits external sources) to address the issue of intellectual property. ExtendScript allows for importing and exporting of code to aid in modularity and code reuse, so these items were also added to the coding scheme.

A few data-driven constructs were included as well. I noted that some users had attempted to make their scripts unreadable by humans; others employed built-in functionality in Photoshop to record their script via the user interface rather than typing code; and others still incorporated rather sophisticated exception handling mechanisms. I wondered how common these practices were and added these to the coding scheme. The resulting EUP constructs are listed in Figure 6.

copyright notice	exception paths (`try/catch`)
end user license agreement	use of recorded code
credits external sources	includes external code
code obfuscation	externalizes code to client

Figure 6: EUP Coding Elements

4.1.2 Coding Process Details

I began the coding process by establishing the reliability of the coding scheme. Two independent raters coded a random sample consisting of 13 scripts ($\approx$ 20% of the total data set) according to the coding scheme. I computed the kappa statistic (Cohen, 1960) as a measure of inter-rater reliability, and while most of my coding elements exceeded the κ=0.80 threshold expected in the social sciences (Landis & Koch, 1977), some revisions were necessary on the constructs user defined objects (κ=0.51) and end user license agreement κ=0.56). The operational definitions of these coding elements were revised, and raters again coded another 20% of the scripts, randomly selected, on the two elements whose inter-rater reliability was not yet established. After updating the criteria and recoding another sample, raters achieved a κ=1.00 on all remaining coding elements. These high κ values may be partially attributed to binary coding categories and the lack of rater judgment required for some constructs. Once inter-rater reliability was confirmed, the two raters each coded half of the scripts according to the revised coding scheme.

4.2 Results

The initial set of Photoshop scripts was collected from the Adobe Exchange community and then cleansed of any entries that were corrupt or incorrectly categorized. After removing the improper entries, the final data set contained a total of 62 individual scripts making up 48 distinct projects contributed by 27 unique users. I use the term *project* to refer to one downloadable entry in the online community. For example, a project could consist of a single script posted as a text file, or it could

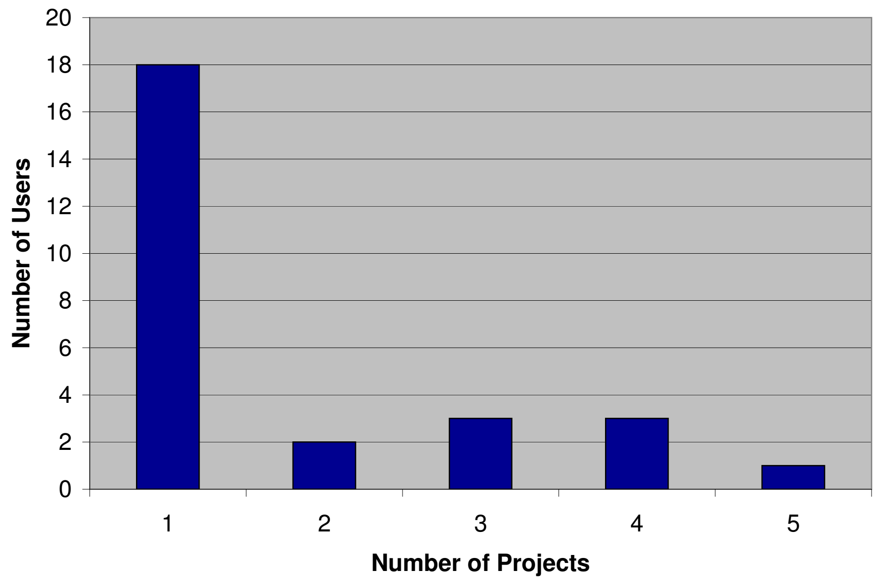


Figure 7: Distribution of Project Submissions

be an archive file containing multiple, related scripts and associated data files. Figure 7 illustrates the distribution of project submissions. Most users posted only one project, though one-third of users made multiple contributions to the community.

The bulk of the results presented here use a per-project unit of analysis, rather than a per-user or per-script approach. Focusing on individual projects mitigates skewing effects that might be introduced by single projects that contain multiple scripts (as in a per-script analysis). I also avoid a per-user analysis as it would be somewhat precarious to infer knowledge of programming based solely on constructs used in a minimal set of examples, particularly given that most users only contributed one project. Although previous studies indicate that many end-users lack formal training in computer science (e.g., Dorn & Guzdial, 2006; Rosson et al., 2005, as well as the previous chapter), I do not know the particular training background of the users who posted to this forum, nor can I infer whether these scripts are the result of personal or work projects. In a sense, what I present here is an analysis of the computing content embodied in projects that might serve as case examples from which another end-user could learn.

Table 6: Project Line Length Breakdown

	Mean	StDev	Median	Min	Max
Code	555.56	674.89	246.5	9	3224
Comment[4]	63.54	65.18	26.5	0	237
Whitespace	65.46	158.47	20.5	0	1057
Total	676.96	760.61	403.5	11	3300

4.2.1 Project Size

While there were as many as eight scripts in a single project, most (87.5%) contained only one script. In order to gain insight into the size and complexity of the projects being created, I calculated the total number of lines used for code statements, whitespace, and comments for each. I report based on the sum of the individual script line lengths for projects containing multiple scripts. Table 6 summarizes basic statistics for project size. There was a large amount of variation in each of the line types computed, as noted by the standard deviations. However, the median lengths indicated moderately sized scripts that included a fair amount of commenting, though the nature of the comments was not closely examined.

Looking at the distribution of these lengths provided a more detailed picture of project size. Figure 8 depicts the range of project sizes in terms of the number of source code lines. There were two noticeable peaks in this distribution, the first of which occurred at 200 or fewer lines of code. This might be predicted if users are expected to implement short programs that accomplish relatively simple tasks. More surprisingly, there was a clear second peak occurring in projects with greater than 1000 lines of code.

Project sizes provided an initial feel for the size and complexity of code, but a more detailed analysis of each project's content was needed to understand the types

[4]Lines counted under "Comment" include both comment-only lines and code lines which have terminal comments.

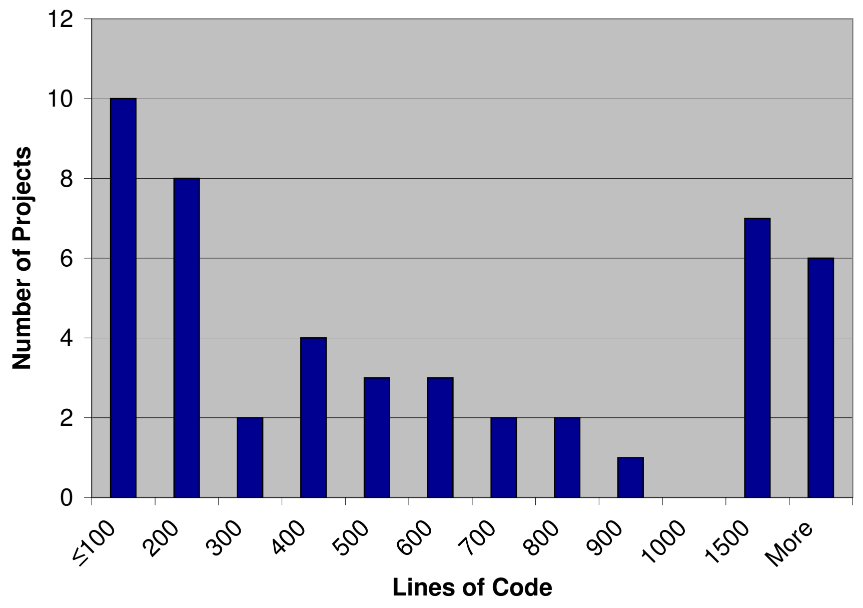


Figure 8: Distribution of Project Lengths

of computing knowledge evidenced in the code base.

4.2.2 Project Content

Application of the coding scheme to all of the individual scripts resulted in an overview of construct use. These results were then aggregated to form a per-project summary. For those projects containing multiple scripts, a construct was indicated as being used if one or more of the constituent scripts used the construct. The aggregate use amounts for each construct, grouped by higher-order concern, are presented in Table 7.

The most commonly used programming constructs were: variable, assignment, relational operators, selection, number, and string. These results were largely expected given that tasks like assigning a numeric value to a variable are fundamental to most coding activities. Excluding those related to intellectual property and recorded code (as these are not programmatic constructs per se), the least frequent constructs were: indefinite loop, nested loops, recursion, type conversion, user-defined objects, and

Table 7: Construct Use by Project

Construct		Use %
Variable		100.00%
Use of Recorded Code		33.33%
Expressions	Assignment	100.00%
	Relational Operators	97.92%
	Mathematical Operators	83.33%
	Logical Operators	54.17%
Control Structures	Selection (`if`)	97.92%
	Definite Loop (`for`)	60.42%
	Exception Paths (`try/catch`)	60.42%
	Indefinite Loop (`while`)	37.50%
	Nested Loops	29.17%
	Recursion	2.08%
Data Types & Structures	Number	100.00%
	String	95.83%
	Array	83.33%
	Boolean	79.17%
I/O	Output to User	83.33%
	User Input	60.42%
Modularity	User-defined Functions	70.83%
	User-defined Objects	18.75%
	Import or Include External Code	0.00%
	Export Code to External Client	0.00%
Intellectual Property	Copyright Notice	62.50%
	End User License Agreement	47.92%
	Credit Given to External Sources	22.92%
	Human Unreadable Code Obfuscation	8.33%

exporting/importing code. Some of these observations could be tied to tool and language influences (e.g., prototype-based rather than class-based definition of objects in JavaScript). Others, like the decrease in use from definite loops to nested loops to recursion, seem indicative of conceptual difficulties noted in previous research (e.g., Soloway, Bonar, & Ehrlich, 1989; Wiedenbeck, 1988). I present a detailed discussion of these issues in the section that follows.

4.3 Discussion of Construct Prevalence

I observed that within the higher-order concerns Expressions, Control Structures, and Modularity some constructs are heavily adopted while others are used at much lower levels (see Table 7). End-user programmers can be viewed in many ways as novices because they traditionally lack formal training in computer science and learn content just-in-time as it relates to their specific tasks (Dorn & Guzdial, 2006, 2010; Rosson et al., 2005). Therefore, previous studies of novice programmer behavior provide useful information for interpreting these results.

4.3.1 Operators

Almost all (97.92%) of the projects used a relational operator (e.g., `<`, `>=`, `!=`), most often inside the condition of a selection (`if`) statement. A clear majority (83.33%) also used mathematical operators. While some projects did include numeric calculations to resize images or parts of images, many of the mathematical operators noted were uses of the unary increment operator as part of a definite loop (`for`) construct. However, markedly fewer projects (54.14%) used a logical operator (`&&`, `||`, `!`).

Previous work with introductory students (Tew, McCracken, & Guzdial, 2005) indicates that beginners tend to struggle with boolean logic in conditional statements. Pane et al. (2001) found that boolean operators are particularly difficult for beginners because they require statements to be expressed in ways that are unfamiliar. Since many end-user programmers are self-taught and have learned to program to support their own goals, it is perhaps not surprising that the code found here seems to use the operators which are familiar to novices.

4.3.2 Control Structures

Control structures of some kind were included in most of the projects I analyzed. Almost all (97.92%) of them included a selection (`if`) statement, and most (60.42%) used a definite loop (`for`) construct. However, only a third of the projects used the indefinite loop (`while`) or nested loop constructs.

Soloway et al. (1989) identified the inherent complexity of the `while` loop because it conflicts with the preferred cognitive strategy that students employ when solving iterative problems. The definite loop (`for`) construct more closely matches the *read, then process* strategy, thus possibly explaining the higher rate of use observed here. Additionally, the infrequent use of nested loops could be another sign of cognitive complexity. Boundary condition errors are a frequent mistake when beginning students write loops, and loop nesting only exacerbates these boundary concerns (Ginat, 2004).

Only one project included recursion, a topic with which many novices struggle (Wiedenbeck, 1988). A common pedagogical technique to address this difficulty is to introduce recursion by way of analogy, but Photoshop lacks readily apparent concrete examples. However, there are several tasks in this context that do lend themselves to recursive solutions. For example, the one use of recursion in this analysis, in effect, traversed a tree made of Photoshop image layers (leaf nodes) and layer sets (internal nodes) removing all non-visible layers along the way.

4.3.3 Abstraction and Modular Coding

A large portion (70.83%) of the projects contained user-defined functions, while significantly fewer (18.75%) implemented objects. Despite the fact that ExtendScript documentation highlights the ability to create reusable code modules using external files, I noted that no project incorporated either the import or export construct.

Fleury (Fleury, 1997) observed that students consistently preferred programs containing duplicated code rather than programs that used abstracted functions, claiming that it was more easily read and debugged. While I note that functions were highly used in this domain, more advanced abstractions for modularity were largely ignored. Hoadley, et al. suggest an explanation that "abstract understanding of a function and belief in the benefits of reusing code" (Hoadley, Linn, Mann, & Clancy, 1996, p. 109) impact whether or not a user is likely to invest time in programming for abstraction.

4.4 Discussion

Additional discussion of these results is separated into two parts. First, I will explicitly address the research question under investigation in this study. Then I will use the results here to draw out implications for the design of an example-driven educational resource (i.e., ScriptABLE).

4.4.1 Findings for the Research Question

At the outset of this chapter I introduced the primary research question for this study as follows:

> **RQ2.2** To what extent does code found online provide relevant examples of introductory programming constructs?

The repository studied here was intentionally domain specific with respect to Photoshop scripting. This guaranteed a high degree relevance (or similarity of context) for the scripting domain studied in this dissertation. Within the collection of projects I noted considerable variation whether or not a particular concept was included (as shown in Table 7). However, as I argued in the previous section, the patterns observed when considering groups of concepts match empirical evidence about conceptual difficulty. That is, the easiest concepts had many examples in the repository, while examples of difficult concepts were few and far between. Thus, the code found in this repository serves to illustrate basic introductory concepts well, but more advanced concepts—though still well within the scope of introductory computing curricula—are largely absent.

This observation is problematic if a learner were to attempt to use this repository as a means to gain additional knowledge. Chances are that the concepts for which there are multiple examples to compare and contrast are the basic concepts which are already well understood. In Chapter 3 the concepts that proved least understood, least frequently used, and most difficult were exporting code, indefinite loop, variable scope, recursion, functional decomposition, and exception handling. Here, some of these same concepts have fewest number of examples, if any. Exporting code, indefinite loops, recursion, and functional decomposition[5] were all among the six least used coding constructs here.

4.4.2 Implications for Next Steps

The results in this chapter have clear implications for the design of educational resources for graphic and web design end-user programmers. Given the scope of the concepts covered, there is a need to explicitly target those concepts were seemingly absent. Notably, this nicely overlaps with the concepts that were also recognized by the intended user population as difficult and not well understood. Targeting the more basic concepts is unnecessary as EUPers express high degrees of familiarity with them and there are currently many examples from which a user can draw.

Having confirmed the need for resources that address the set of difficult topics, I have chosen to limit the focus of the remainder of this dissertation to the following concepts.

[5]I group functional decomposition among these due to the general lack of modular coding practices observed in the repository.

- selection statement
- indefinite loops
- exception handling
- recursion
- functional decomposition
- importing code

This limited set of concepts will provide a tractable set of learning goals for the time-constrained evaluation study described in Chapter 6. It intentionally contains selection as an easier construct, but focuses primarily on the more difficult concepts.

4.5 Chapter 4 Summary

In this chapter I have presented a content analysis of Photoshop scripting projects posted in a user-driven resource online. The detailed breakdown of introductory concepts covered by projects in the repository provides insight into the concepts for which current example-based resources may lack sufficient detail. Notably, I found significant overlap with the concepts previously noted as difficult, not understood, and infrequently used in Chapter 3. Taken together, the results of these two chapters seem to indicate a gap in the coverage of these topics by current domain-specific example materials. In the next chapter I will present the design of a new case-based learning aid intended to explicitly scaffold learning about these topics through use of examples.

CHAPTER V

DESIGN OF SCRIPTABLE

This chapter introduces the design of ScriptABLE, a new case-based learning aid for end-user programmers who script Photoshop. In designing and building this system, I sought to address the lack of explicit instruction and coverage of introductory programming concepts that I observed in Chapter 4. Additionally, I aimed to develop a resource that would closely align with the learning strategies and preferred resources discussed by graphic and web designers in Chapter 3. Specifically, I had five design criteria distilled from the work described in the previous chapters:

1. **Focus on Project-Driven Learning** I noted in Chapter 3 that graphic and web designers learned new programming techniques in the context of new projects. As their learning was typically contextualized within particular projects, I proposed that informal instruction should take a similar approach.

2. **Connect Concepts to Example Code** Two of the most highly utilized resources I observed in Chapter 3 were code examples and online documentation (e.g., walkthroughs and tutorials). I aimed to extend these types of resources by augmenting them with explicit instruction about introductory programming concepts. This conceptual instruction would be directly tied to the code being developed for a project.

3. **Highlight Errors and Recovery** The theoretical models I discussed in Chapter 2 highlight the importance of learning from errors. Case-Based Reasoning (Kolodner et al., 2004), Cognitive Load Theory (Sweller, 1988), and Minimalist

Instruction (Carroll, 1998) all emphasize the need for project-driven instructional materials to provide explicit reference to likely errors and strategies to recover from them. Thus, I sought to leverage errors as natural points to introduce conceptual content enabling one to overcome a problem.

4. **Multiply Label Concepts** Some participants in the card sorting study from Chapter 3 had difficulty recognizing formal programming terminology without the aid of definitions or example code. In order to maximize the likelihood that learners locate relevant instruction offered by the resource, I sought to use multiple means of indexing content.

5. **Target Unfamiliar and Misunderstood Content** I identified a number of concepts that were universally difficult, misunderstood, and infrequently used by graphic and web designers in Chapter 3. Further, in Chapter 4 I observed that examples of many of these same concepts were not prevalent in online code repositories for Photoshop scripting. Thus, I targeted the six concepts outlined at the end of the previous chapter: selection statement, indefinite loops, exception handling, recursion, functional decomposition, and importing code. These concepts provided a tractable set of content that could potentially result in learning.

These criteria lend themselves quite naturally to a case-based learning aid. Recall from Chapter 2 that a case-based learning aid is a collection of interrelated cases (or projects) that distill the experiences of others so that a novice can learn underlying content (Kolodner et al., 2004).

Throughout the remainder of this chapter I will describe ScriptABLE's notable features and will emphasize the design aspects that address each of the five criteria above. Section 5.1 makes up the majority of the chapter and discusses ScriptABLE's relevant components. In Sections 5.2 and 5.3, I detail ScriptABLE's coverage of programming concepts and briefly discuss an expert review of the system's content.

5.1 ScriptABLE

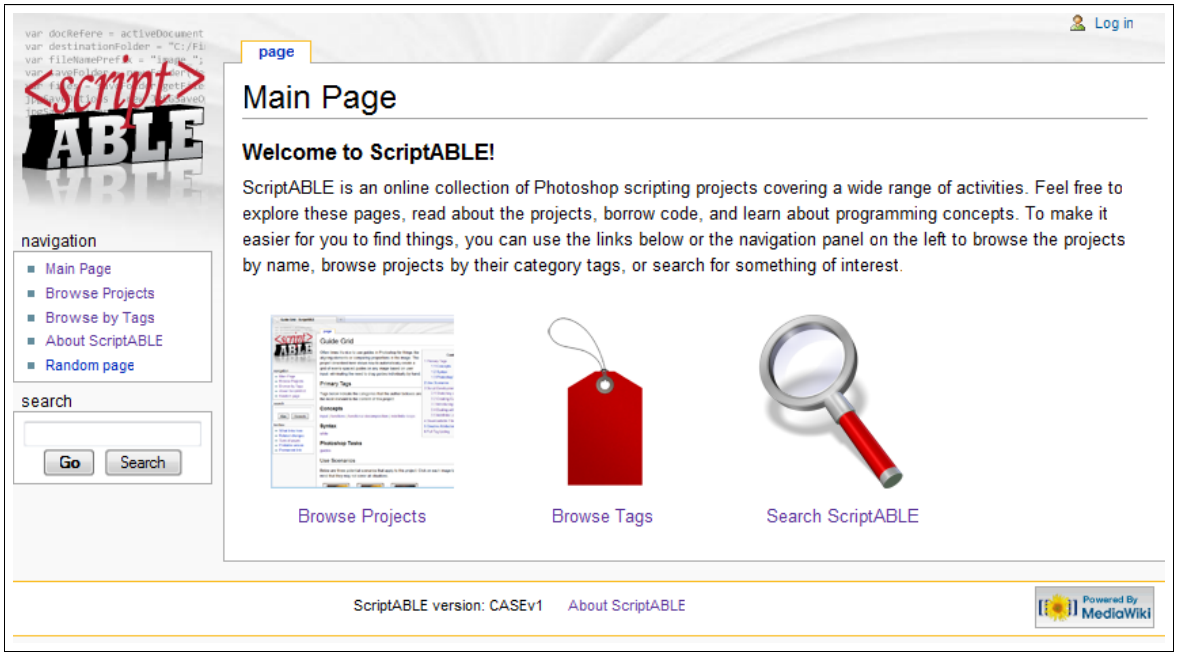
ScriptABLE is implemented using a customized installation of the MediaWiki software.[1] However, the use of a wiki here is largely incidental to the primary purpose of this research. It was primarily used as a content management system to simplify the content creation process, and I relied on MediaWiki's built in search and indexing (i.e., categories) features to complete ScriptABLE's implementation. Modifications to the code were made as necessary to customize aspects of the interface and to disable many extraneous wiki features. Open editing of ScriptABLE's pages is prohibited and users access the content anonymously. ScriptABLE is available online at `http://cases.scriptable.org`.

Figure 9 illustrates ScriptABLE's initial welcome screen. Following a brief greeting, users are given a choice to navigate the system by browsing individual projects, browsing by the tags used for indexing purposes, or by performing a search query. Links to each of these three navigation mechanisms are always available using the navigation panel at the left of the page.

ScriptABLE's individual case design is based loosely on that of *Designing Pascal Solutions* (hereafter, *DPS*). This textbook is a collection of case studies that introduces students to introductory computing concepts through developing solutions to programming problems (Clancy & Linn, 1995). Clancy and Linn describe the components of their cases and the manner in which learners engage with the content:

> In each case study in *Designing Pascal Solutions*, students first read and analyze a Problem Statement. Then they use the Commentary on the design to guide their own knowledge construction. They answer self-test questions while reading the Commentary. At appropriate intervals they

[1] `http://www.mediawiki.org/`


Log in

page

Main Page

Welcome to ScriptABLE!

ScriptABLE is an online collection of Photoshop scripting projects covering a wide range of activities. Feel free to explore these pages, read about the projects, borrow code, and learn about programming concepts. To make it easier for you to find things, you can use the links below or the navigation panel on the left to browse the projects by name, browse projects by their category tags, or search for something of interest.

ScriptABLE version: CASEv1 About ScriptABLE

Figure 9: ScriptABLE Front Page

> extend their understanding by doing on-line and off-line exercises. They consolidate their knowledge using the Review section, doing exercises to link their ideas to related problems and issues. As they engage in these activities they read the Pascal Code for each problem solution and work with it on line in a computer laboratory. (Clancy & Linn, 1995, p. xvii)

ScriptABLE incorporates many of these components, but leaves others out as a consequence of differences in audiences and intended use. Most notably, the Review section from *DPS* has no corresponding component in ScriptABLE. The Review section primarily enumerates a number of related practice problems for the reader which could be used for assessment purposes. Given that ScriptABLE is not intended for a formal learning environment, this component is unnecessary. Also, because ScriptABLE is a collection of hyperlinked pages, its cases have been augmented with multiple tags corresponding to entries in the library's indexing system.

5.1.1 Anatomy of a Project

The primary components of a ScriptABLE project[2] are:

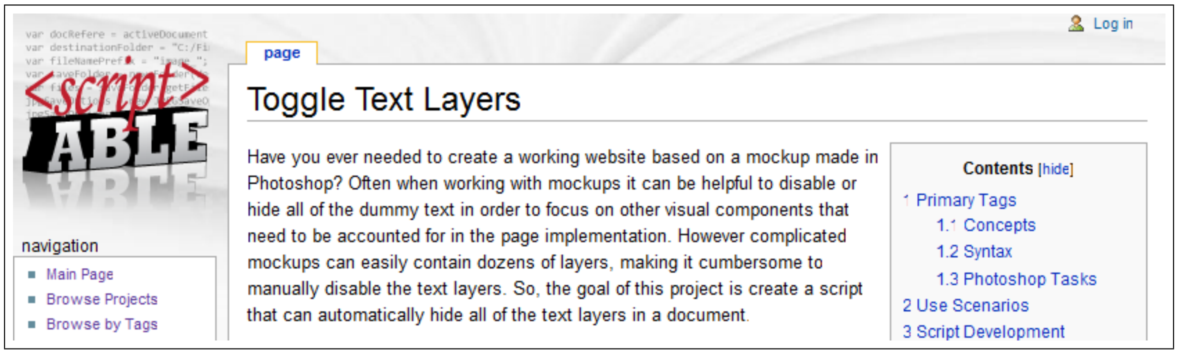
- Project Description
- Primary Tags
- Use Scenarios
- Script Development
- Downloadable Files
- Creative Attribution
- Full Tag Listing

I will discuss each of these components in this subsection. Following the description of a project, I introduce additional details regarding the tagging and indexing system as well as the search interface used in ScriptABLE. Throughout, I will provide screenshots from the interface for each of the main components.

5.1.1.1 Project Description

Each of the seven ScriptABLE projects begins with a brief description that outlines the purpose of the script to be written. This roughly corresponds to the Problem Statement used by Clancy and Linn (1995) in *DPS*. The description motivates the project with a real world problem situated within the context of typical Photoshop activities a graphic or web designer would likely encounter. In this way it immediately connects the project to the intended user's community of practice (Lave & Wenger, 1991) in an effort to begin providing a sense of "thick" authenticity (Shaffer & Resnick,

[2]The ScriptABLE interface uses the more general term "project" in lieu of "case" to avoid terminology confusion.



Toggle Text Layers

Have you ever needed to create a working website based on a mockup made in Photoshop? Often when working with mockups it can be helpful to disable or hide all of the dummy text in order to focus on other visual components that need to be accounted for in the page implementation. However complicated mockups can easily contain dozens of layers, making it cumbersome to manually disable the text layers. So, the goal of this project is create a script that can automatically hide all of the text layers in a document.

Contents [hide]

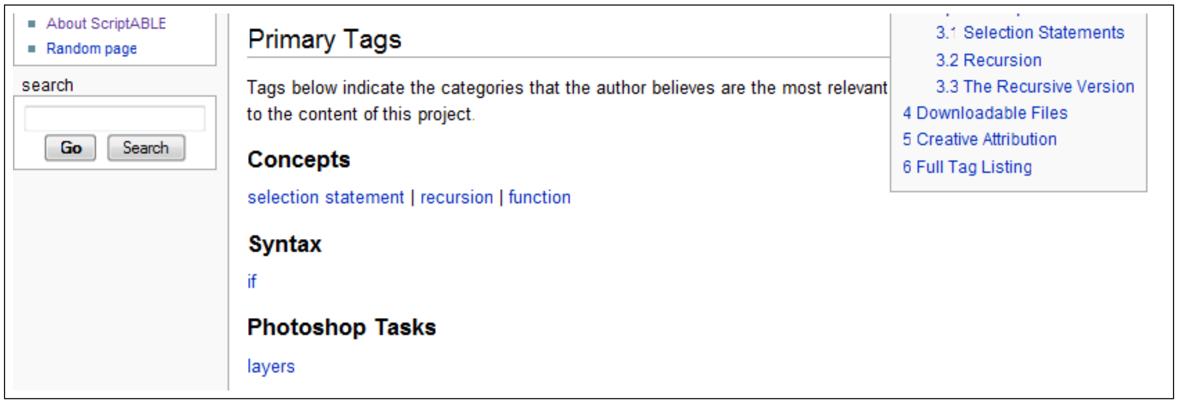
1 Primary Tags
1.1 Concepts
1.2 Syntax
1.3 Photoshop Tasks
2 Use Scenarios
3 Script Development

Figure 10: Toggle Text Layers Project Description

1999). That is, the description highlights a real world problem likely to be of personal interest to the user due to its connection to the tools and practices of the user's domain of expertise. From this context set by the description, other aspects of the project structure and writing style are then intended to continue contributing to the overall feeling of authenticity. An example description from the Toggle Text Layers project is shown in Figure 10.

5.1.1.2 Primary Tags

Following the project description is the primary tags section (see Figure 11). This section lists the tags which are the most relevant to the content of the project. These tags provide links into ScriptABLE's indexing system and can be thought of more generally as index terms. Tags are shown in three separate groups: Concepts, Syntax, and Photoshop Tasks. Thus, the primary tags can be viewed as an indication of what programming concepts, JavaScript syntax, and Photoshop tasks will be addressed while developing the code necessary for this project. For example, from Figure 11 one can tell that the Toggle Text Layers project will introduce selection statements (`if` syntax), recursion, and functions while working with Photoshop layers. There may, of course, be additional relevant tags for ancillary concepts or syntax used in the development of the script which are not listed in this section. Primary tags are limited to only those concepts, syntax, and tasks which are explicitly discussed and



Primary Tags

Tags below indicate the categories that the author believes are the most relevant to the content of this project.

Concepts

selection statement | recursion | function

Syntax

if

Photoshop Tasks

layers

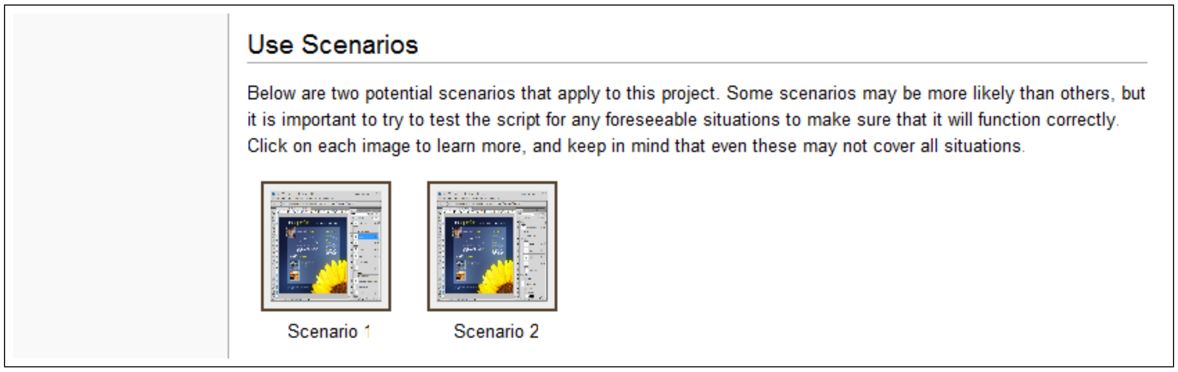
Figure 11: Toggle Text Layers Primary Tags

explained in the body of the project write up (i.e., the Script Development section). I will provide a more detailed description of all of the possible tags and ScriptABLE's index later in Section 5.1.2.

5.1.1.3 Use Scenarios

Each project contains one or more use scenarios that describe different circumstances in which the script can be run. These serve as use cases for testing the code developed for the project. Each use scenario is selected so that there are some inputs or conditions that will cause the script to fail. Those failures are then used as motivation for additional versions of the code, progressively approaching the final result that functions correctly for all scenarios.

Within the project page itself, use scenarios are presented as thumbnail pictures that show a small preview of the input details (see Figure 12). When a user clicks on a particular thumbnail, they are presented with a new pop-up window with details about that scenario, as shown in Figure 13. Scenario descriptions all have the same general form. First, each scenario is given a unique name. Then a series of instructions follows, which detail how to run the script in the appropriate way. In most cases this involves how to setup the input environment either as a Photoshop document or the values that will be supplied for the script's prompt dialogs. Example images and


Use Scenarios

Below are two potential scenarios that apply to this project. Some scenarios may be more likely than others, but it is important to try to test the script for any foreseeable situations to make sure that it will function correctly. Click on each image to learn more, and keep in mind that even these may not cover all situations.

Figure 12: Toggle Text Layers Script Use Scenarios

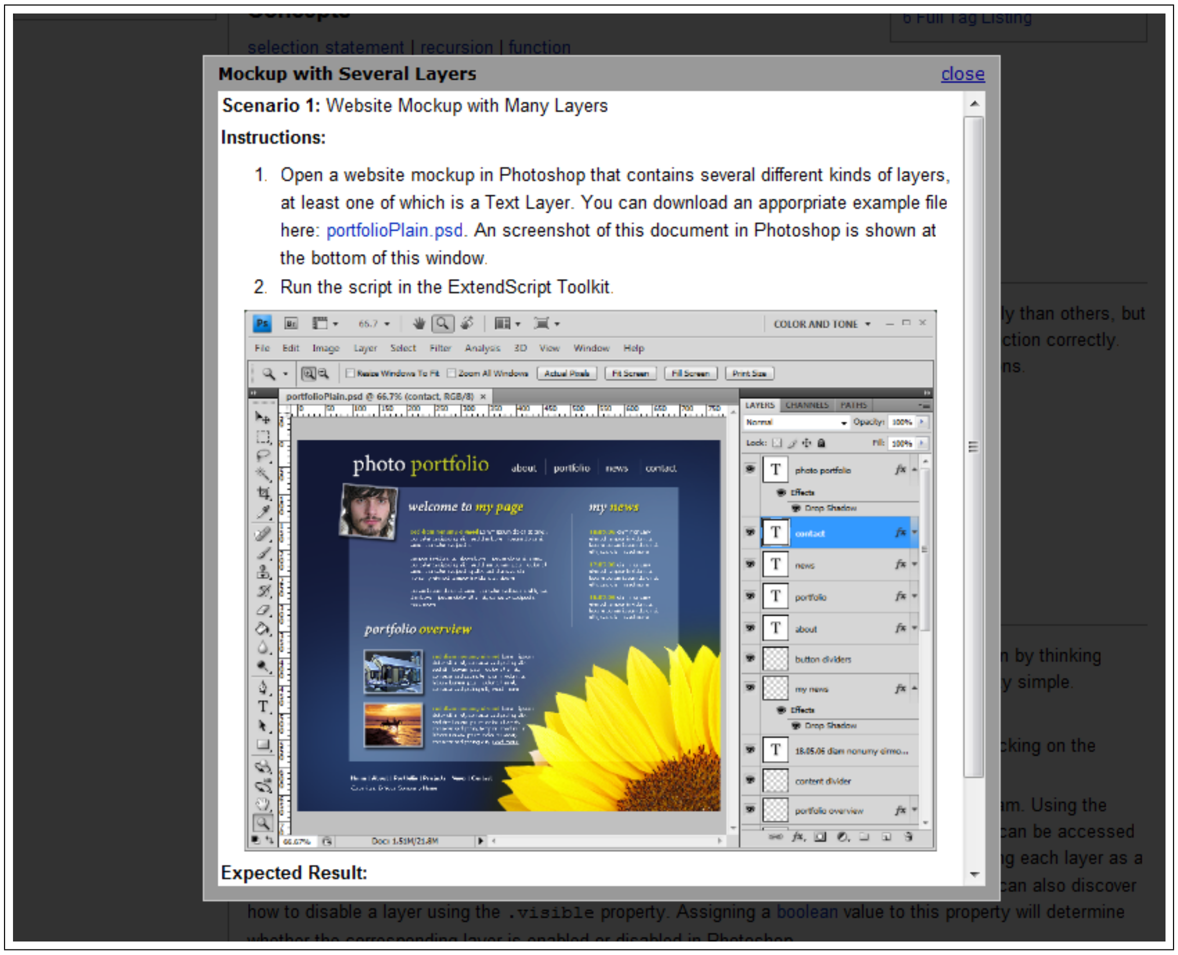
documents are provided for a user to download and test, should they choose to run the script as instructed. Lastly, each use scenario ends with a description of the expected result from running the script.

I designed the set of use scenarios for each ScriptABLE project to intentionally highlight a need for an additional version of the code, writing which would entail the introduction of a new programming concept or technique. The provided scenarios are not intended to be an exhaustive list of every possible test case, and users are cautioned to this effect in the project text.

5.1.1.4 Script Development

The bulk of each project page consists of the script development section. This section serves the same purpose as the Commentary in *DPS* cases. It provides a narrative description of how the code for this project is developed and interleaves discussion of the necessary conceptual content within the context of the project. To illustrate the typical structure of the script development section, I will walk through the development section of the Toggle Text Layers (TTL) project as an exemplar.

The goal of TTL is to produce a script which will walk through the collection of layer objects in a Photoshop document and disable those layers which consist of text. These layers are denoted in the Photoshop interface with an icon containing the capital letter T. As shown in Figure 14, the script development section begins



selection statement | recursion | function

Mockup with Several Layers close

Scenario 1: Website Mockup with Many Layers

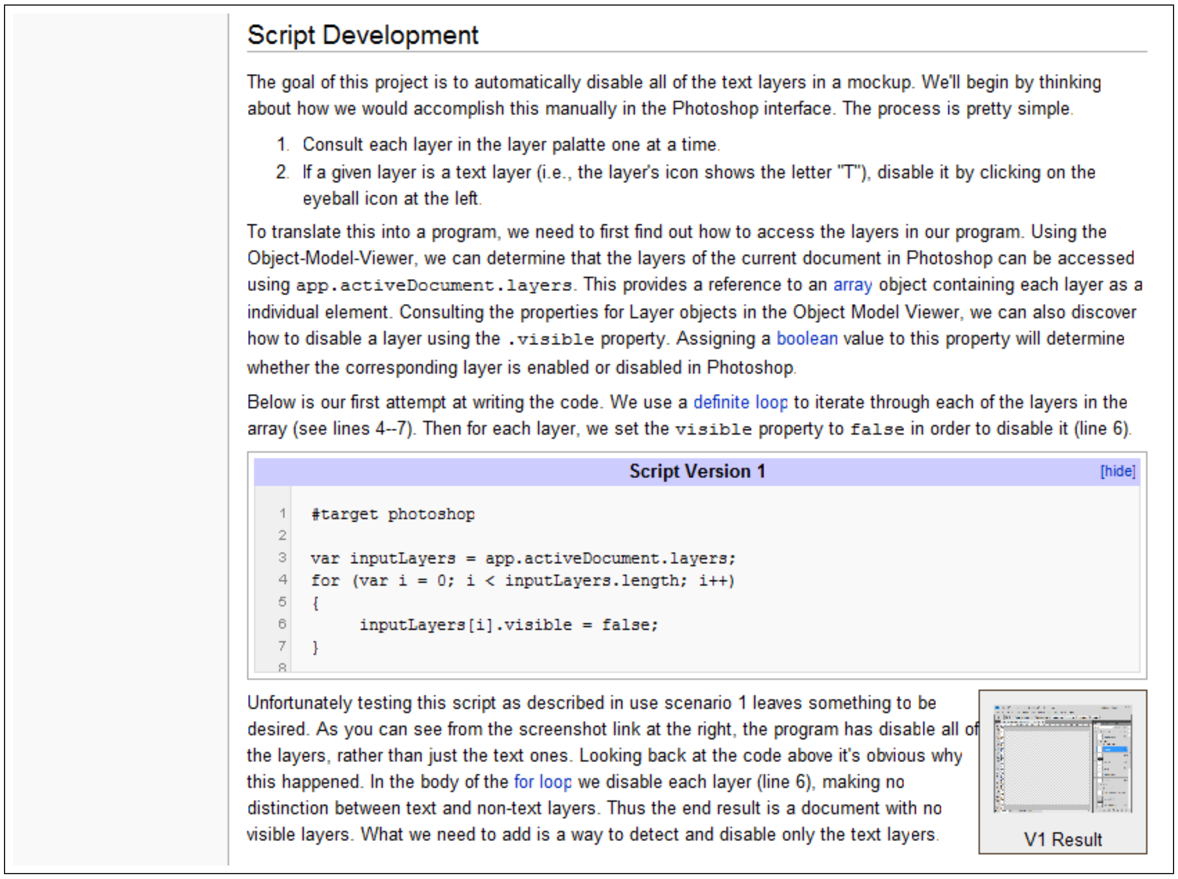
Instructions:

1. Open a website mockup in Photoshop that contains several different kinds of layers, at least one of which is a Text Layer. You can download an appropriate example file here: portfolioPlain.psd. An screenshot of this document in Photoshop is shown at the bottom of this window.
2. Run the script in the ExtendScript Toolkit.

Expected Result:

how to disable a layer using the `.visible` property. Assigning a boolean value to this property will determine

Figure 13: Toggle Text Layers Script Use Scenario 1 Popup



Script Development

The goal of this project is to automatically disable all of the text layers in a mockup. We'll begin by thinking about how we would accomplish this manually in the Photoshop interface. The process is pretty simple.

1. Consult each layer in the layer palatte one at a time.
2. If a given layer is a text layer (i.e., the layer's icon shows the letter "T"), disable it by clicking on the eyeball icon at the left.

To translate this into a program, we need to first find out how to access the layers in our program. Using the Object-Model-Viewer, we can determine that the layers of the current document in Photoshop can be accessed using `app.activeDocument.layers`. This provides a reference to an array object containing each layer as a individual element. Consulting the properties for Layer objects in the Object Model Viewer, we can also discover how to disable a layer using the `.visible` property. Assigning a boolean value to this property will determine whether the corresponding layer is enabled or disabled in Photoshop.

Below is our first attempt at writing the code. We use a definite loop to iterate through each of the layers in the array (see lines 4--7). Then for each layer, we set the `visible` property to `false` in order to disable it (line 6).

Script Version 1 [hide]

```
1  #target photoshop
2
3  var inputLayers = app.activeDocument.layers;
4  for (var i = 0; i < inputLayers.length; i++)
5  {
6       inputLayers[i].visible = false;
7  }
8
```

Unfortunately testing this script as described in use scenario 1 leaves something to be desired. As you can see from the screenshot link at the right, the program has disable all of the layers, rather than just the text ones. Looking back at the code above it's obvious why this happened. In the body of the for loop we disable each layer (line 6), making no distinction between text and non-text layers. Thus the end result is a document with no visible layers. What we need to add is a way to detect and disable only the text layers.

Figure 14: Toggle Text Layers Script Development Excerpt 1

by enumerating a step-by-step process by which a user could manually produce the desired result. This naïve process is used as a basic algorithm for the script. The narrative quickly moves toward a first version of the script by directly translating the manual process into code. The first version of the TTL script simply uses a definite loop to iterate through each of the elements of an array containing the layers and disables them by changing their visible property.

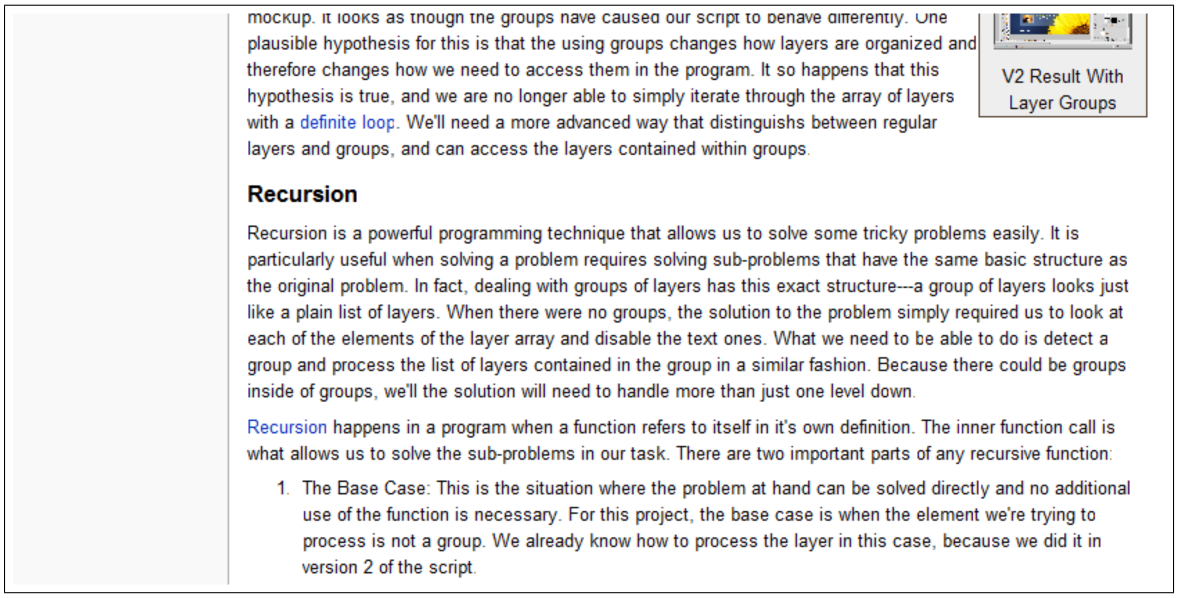
Throughout the development section, each version of the script is followed by an attempt to execute the program as described by one of the use scenarios. In TTL, the first version of the script fails to produce the expected output for use scenario one because the script does not check whether or not a layer is a text layer; it merely disables all layers. Along with a description of the script's unexpected result, users are provided with a screenshot of the output (an example can be seen as an expandable thumbnail captioned "V1 Result" in Figure 14).

Failure on a use scenario is used to motivate the introduction of a new programming concept or technique. For example, at this point in the TTL script development narrative, readers are introduced to the concept of selection statements, provided with an explanation of the basic `if` syntax in JavaScript, and shown how to correctly incorporate the concept into the previous version of the script. This, in turn, leads to a new version of the script, which must be again tested using a use scenario. Like many of the projects, the intermediate code for the TTL project successfully functions for the first use scenario, but fails on the subsequent one.

The second version fails for TTL because it is a straightforward iterative solution that does not take into account the fact that layers can be grouped to form a nested, tree-like data structure of layers and groups. Correctly handling the second use scenario (in which there are nested groups) necessitates a recursive solution. Thus, the script development section then introduces the concept of recursion, shows a simplified example of the necessary recursive function, and then incorporates it into the third and final version of the script. Excerpts of this portion of the TTL project can be seen in Figures 15 and 16.

The script development section ends with confirmation that the script does indeed produce the desired results for the each of the use scenarios. This involves testing the failed scenario leading to the final version of the code, as well as re-testing prior scenarios to ensure that they have not been negatively affected by later changes to the code.

By presenting concepts as they arise throughout the script development section, I am able to meet several of ScriptABLE's design goals. First, the vast majority of the instructional content designed into ScriptABLE takes place within the script development section. This results in a highly project-driven learning environment. Second, as a direct result of the project-driven focus, conceptual content is inherently



mockup. It looks as though the groups have caused our script to behave differently. One plausible hypothesis for this is that the using groups changes how layers are organized and therefore changes how we need to access them in the program. It so happens that this hypothesis is true, and we are no longer able to simply iterate through the array of layers with a definite loop. We'll need a more advanced way that distinguishes between regular layers and groups, and can access the layers contained within groups.

V2 Result With Layer Groups

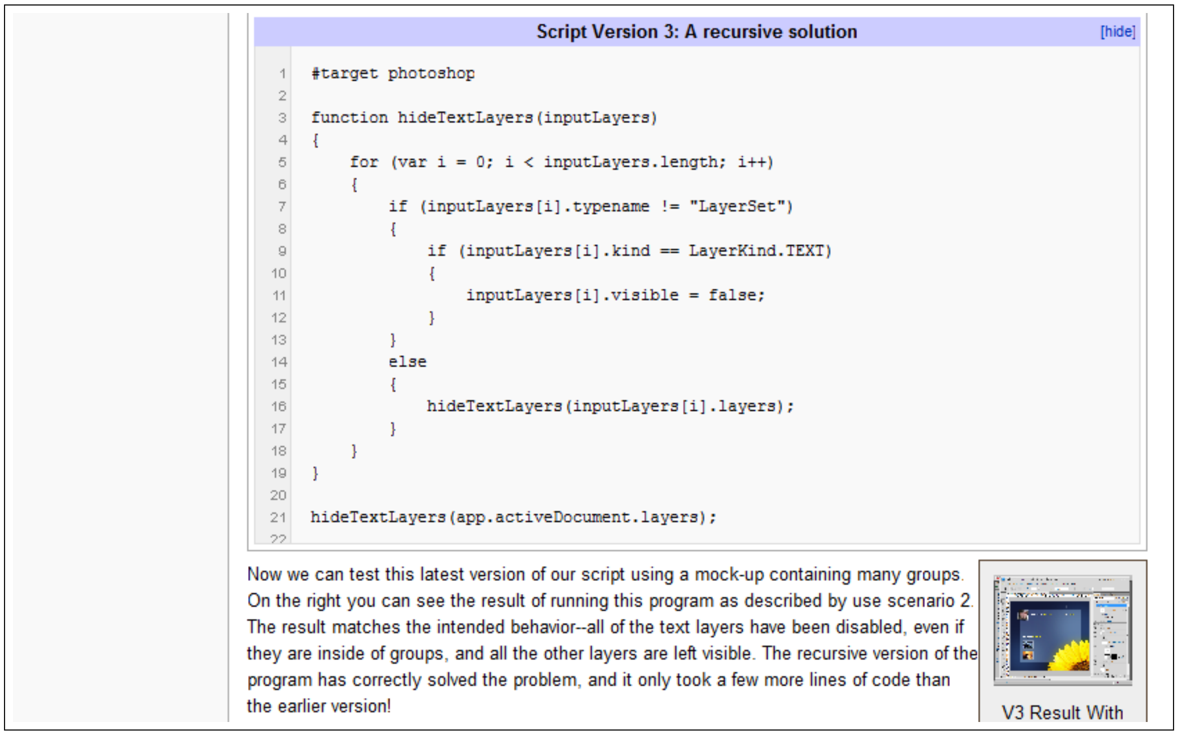
Recursion

Recursion is a powerful programming technique that allows us to solve some tricky problems easily. It is particularly useful when solving a problem requires solving sub-problems that have the same basic structure as the original problem. In fact, dealing with groups of layers has this exact structure---a group of layers looks just like a plain list of layers. When there were no groups, the solution to the problem simply required us to look at each of the elements of the layer array and disable the text ones. What we need to be able to do is detect a group and process the list of layers contained in the group in a similar fashion. Because there could be groups inside of groups, we'll the solution will need to handle more than just one level down.

Recursion happens in a program when a function refers to itself in it's own definition. The inner function call is what allows us to solve the sub-problems in our task. There are two important parts of any recursive function:

1. The Base Case: This is the situation where the problem at hand can be solved directly and no additional use of the function is necessary. For this project, the base case is when the element we're trying to process is not a group. We already know how to process the layer in this case, because we did it in version 2 of the script.

Figure 15: Toggle Text Layers Script Development Excerpt 2



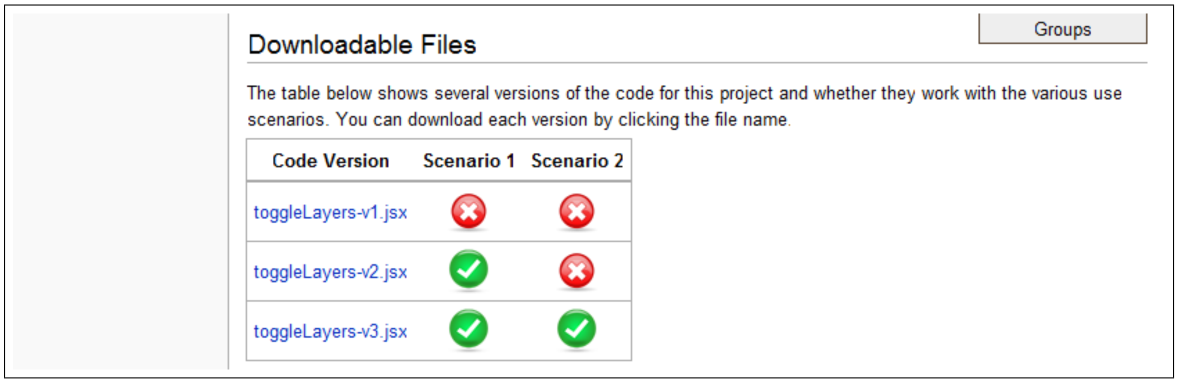
Script Version 3: A recursive solution [hide]

```
#target photoshop

function hideTextLayers(inputLayers)
{
    for (var i = 0; i < inputLayers.length; i++)
    {
        if (inputLayers[i].typename != "LayerSet")
        {
            if (inputLayers[i].kind == LayerKind.TEXT)
            {
                inputLayers[i].visible = false;
            }
        }
        else
        {
            hideTextLayers(inputLayers[i].layers);
        }
    }
}

hideTextLayers(app.activeDocument.layers);
```

Now we can test this latest version of our script using a mock-up containing many groups. On the right you can see the result of running this program as described by use scenario 2. The result matches the intended behavior--all of the text layers have been disabled, even if they are inside of groups, and all the other layers are left visible. The recursive version of the program has correctly solved the problem, and it only took a few more lines of code than the earlier version!

V3 Result With

Figure 16: Toggle Text Layers Script Development Excerpt 3



Groups

Downloadable Files

The table below shows several versions of the code for this project and whether they work with the various use scenarios. You can download each version by clicking the file name.

Code Version	Scenario 1	Scenario 2
toggleLayers-v1.jsx	✗	✗
toggleLayers-v2.jsx	✓	✗
toggleLayers-v3.jsx	✓	✓

Figure 17: Toggle Text Layers Script Downloadable Files

connected to example code. By focusing exclusively on one concept in each new iteration of the code, users can directly compare the two subsequent versions of the script to identify the relevant components of the concept's implementation. Second, the role of code errors and expectation failure in ScriptABLE are evident because the projects have been written to take advantage of these situations as an impetus for additional learning. Strategies for recovering from errors are then intertwined naturally into the narrative of the project.

5.1.1.5 *Downloadable Files*

Following the lengthy narrative about the code, there is a simple section providing quick access to all of the code created for the various versions of the project. The code is presented in a tabular format (as seen in Figure 17) and maps each code version to an indicator of success for each of the use scenarios. This visual reference is intended to serve as a reminder of the iterative development process as well as a means to promote comparison between versions of the code.

5.1.1.6 *Creative Attribution*

A creative attribution section provides additional authenticity by linking to a source of inspiration for the project. Currently such attributions point to graphic design rules of thumb, questions posed on online forums by users of Photoshop, and/or example scripting code another user has uploaded. While perhaps tangential to the content of the project, the purpose of this section is to help users identify with the problems described in the project and to begin to see the code and concepts explained in the script as relevant to their activities. My prior work also indicated that recognition for the intellectual property and creative efforts of others are commonplace within ScriptABLE's target user community (Dorn et al., 2007). Accordingly, the attribution section aims to recognize the provenance of the ideas used in ScriptABLE projects.

5.1.1.7 Full Tag Listing

Each project page ends with a full tag listing that shows all of the tags relevant to this project. As mentioned above, the primary tags section only includes those concepts and syntax which are explicitly discussed in the script development section. However, there are often other concepts used throughout the project. For example, in the TTL project, definite looping and the corresponding `for` loop syntax are used in the code but are not explained in the text. Despite the fact that they are not discussed, the code in this project could still serve as a useful example of a definite loop for a learner. The full tag listing section includes all such relevant tags and thus enables users to locate additional examples of concepts in projects where they are not the primary focus.

5.1.2 Tagging and Indexing

Now that I have described the structure of ScriptABLE projects, I will turn to the tagging and indexing provided within the system. As described in the previous subsection, projects in ScriptABLE are annotated with a number of tags. These tags serve as index terms in the system and can be used to browse or locate projects that make use of similar features or concepts.

ScriptABLE's tags are arranged in a simple hierarchy, as shown in Figure 18.

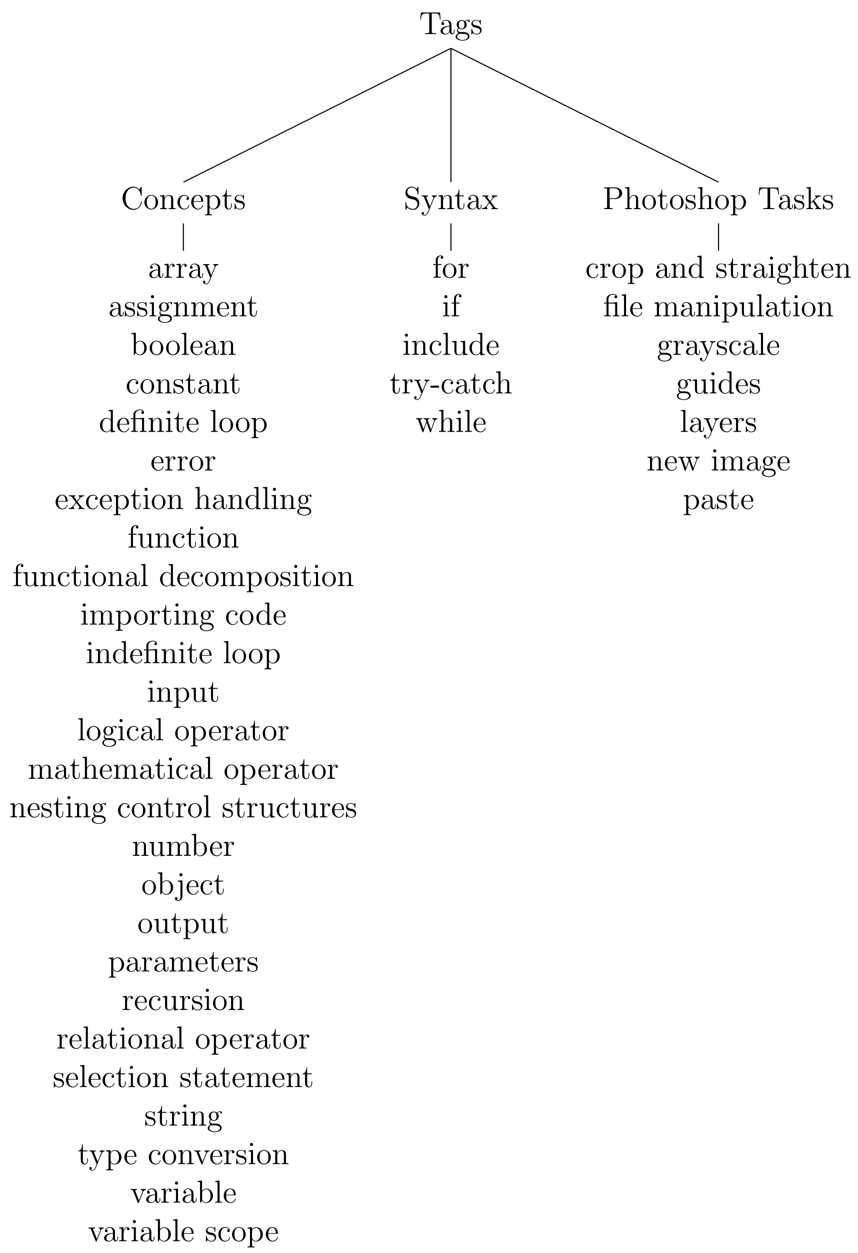


Figure 18: ScriptABLE Tag Hierarchy

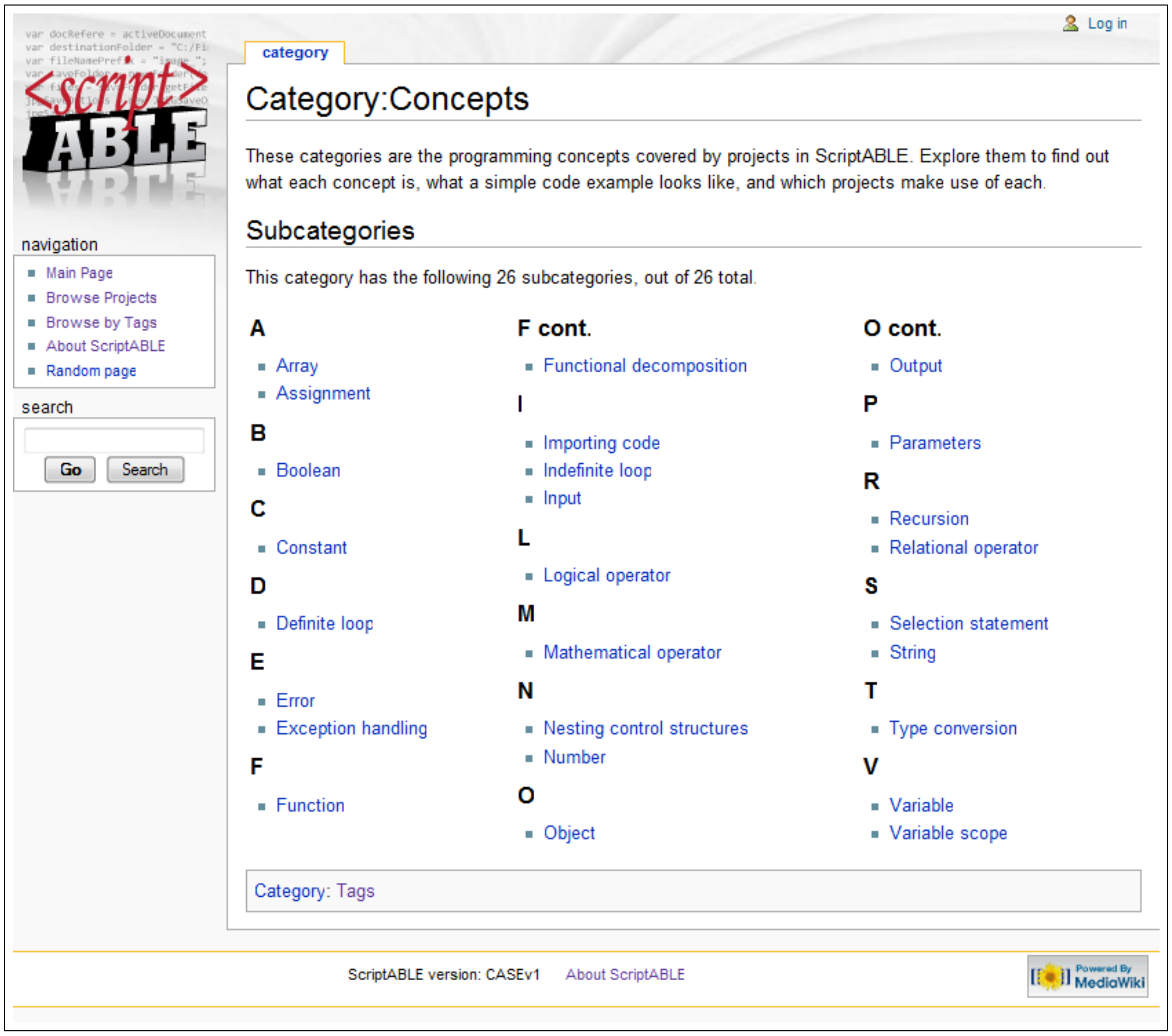
There are three top level groups: concepts, syntax, and Photoshop tasks. The majority of the tags (26 in total) are contained in the concepts category. These concept tags are drawn directly from the list of concepts I examined in Chapter 3. There are also five syntax tags which are unique from the corresponding concepts in name (e.g., exception handling and `try-catch`). When a concept and an element of syntax have an identical name (e.g., function/`function`) the tag only appears once in the concept category. Syntax tags were included separately in order to provide increased recognition of index terms, similar to what I observed in the card sorting study in Chapter 3. Lastly, there are seven different task related tags that are tied to particular features of Photoshop used in the projects (e.g., guide rules).

ScriptABLE users can browse the tag hierarchy using navigation links provided throughout the interface. When a user views one of the three top level categories, he or she is presented with a simple alphabetical list of all tags contained therein. For example, Figure 19 shows the page displayed when a user views the concepts category.

Index pages for individual tags can be viewed by clicking on the link for a tag's name anywhere in the ScriptABLE system. All of these index pages show the tag name and an alphabetical list of the projects that are marked with the tag; however, different auxiliary information is displayed on the index page depending on the top level category to which a tag belongs:

- Concept tags contain the most information, with a brief definition for the concept and a snippet of JavaScript code illustrating the abstract concept. Figure 20 shows the index page for the selection statement concept. These definitions and examples are identical to those used in the card sorting study from Chapter 3.

- Syntax tags serve as simple cross-indexes for their corresponding concept. Each



Log in

category

Category:Concepts

These categories are the programming concepts covered by projects in ScriptABLE. Explore them to find out what each concept is, what a simple code example looks like, and which projects make use of each.

Subcategories

This category has the following 26 subcategories, out of 26 total.

A
- Array
- Assignment

B
- Boolean

C
- Constant

D
- Definite loop

E
- Error
- Exception handling

F
- Function

F cont.
- Functional decomposition

I
- Importing code
- Indefinite loop
- Input

L
- Logical operator

M
- Mathematical operator

N
- Nesting control structures
- Number

O
- Object

O cont.
- Output

P
- Parameters

R
- Recursion
- Relational operator

S
- Selection statement
- String

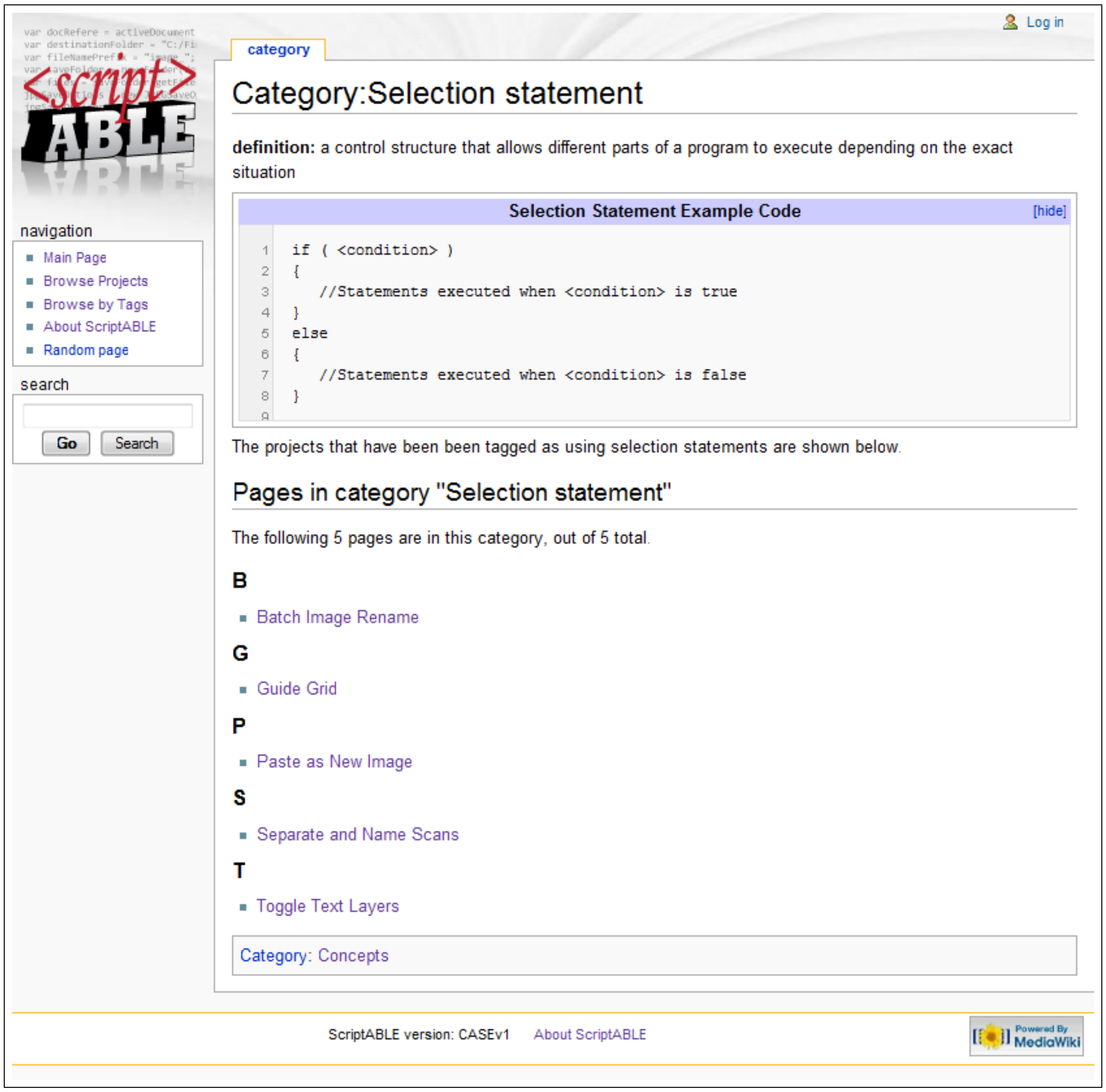
T
- Type conversion

V
- Variable
- Variable scope

Category: Tags

ScriptABLE version: CASEv1 About ScriptABLE

Figure 19: ScriptABLE Concepts Tag Page



Category:Selection statement

definition: a control structure that allows different parts of a program to execute depending on the exact situation

Selection Statement Example Code

```
if ( <condition> )
{
   //Statements executed when <condition> is true
}
else
{
   //Statements executed when <condition> is false
}
```

The projects that have been been tagged as using selection statements are shown below.

Pages in category "Selection statement"

The following 5 pages are in this category, out of 5 total.

B
- Batch Image Rename

G
- Guide Grid

P
- Paste as New Image

S
- Separate and Name Scans

T
- Toggle Text Layers

Category: Concepts

ScriptABLE version: CASEv1 About ScriptABLE

Figure 20: ScriptABLE Selection Statement Tag Page

of these index pages instruct users to see the concept page to learn more about the particular syntactic element, the goal being to direct users towards the conceptual listing to the extent possible.

- Photoshop task tags have no additional information beyond the alphabetical listing of tagged projects.

5.1.3 Search

A simple search engine is incorporated into ScriptABLE. With it, users can perform plain-text searches of the contents of project pages and/or the index pages. Search results are presented as a basic list of pages that match the criterion. As I made no significant changes to the default search functionality provided by the MediaWiki software, I will not discuss this feature further here.

5.2 Concept Coverage

In designing ScriptABLE's projects, I specifically aimed to provide instruction for the six concepts that I outlined at the end of Chapter 4. Recall that these concepts were: selection statement, indefinite loops, exception handling, recursion, functional decomposition, and importing code. With the exception of selection statement (which is intentionally included as an easier concept), these concepts were among those least frequently used, most difficult, and least understood (Chapter 3) and least prevalent in an existing code repository for scripters (Chapter 4).

Table 8 depicts the coverage of these concepts among the seven projects included in ScriptABLE. The concepts marked in this table correspond to the primary concept tags for the projects. All of the targeted concepts are covered by two different projects, allowing for basic comparison between different uses of the concept. Most projects explicitly discuss two of the targeted concepts. However, the Grayscale Variations and Paste as New Image projects only cover one concept each. This was necessary

Table 8: Concept Coverage of ScriptABLE Projects

ScriptABLE Project	**Selection Statement**	**Functional Decomposition**	**Exception Handling**	**Indefinite Loop**	**Importing Code**	**Recursion**
Batch Image Rename					X	X
Center Guide		X			X	
Grayscale Variations			X			
Guide Grid		X		X		
Paste as New Image			X			
Separate and Name Scans	X			X		
Toggle Text Layers	X					X

for the sole reason of keeping the script development sections to a reasonable length.

5.3 Content Review

Following completion of ScriptABLE's content, a third-party reviewed all of the project and index pages. The reviewer was an expert computer science educator with 15 years of experience as a university instructor and professor. The purpose of this review step was to verify the conceptual content of the projects, identify any missing information needed for a novice to understand the projects and concepts, and to confirm the overall readability of the projects. I collaboratively revised ScriptABLE's content with the reviewer based on her independent feedback. These revisions were incorporated into the final version of ScriptABLE used in the evaluation study which I will describe in the following chapter.

5.4 Notes on ScriptABLE's Design

In this section I briefly comment on aspects of ScriptABLE's design by making comparisons to some existing commercial resources and prior case-based learning aids. The style and structure of ScriptABLE projects is reminiscent of tutorials, and a skeptic might ask how it is any different from any other tutorial one might find on the Web. To distinguish ScriptABLE from other tutorial style documentation, consider two of the popular online tutorial sites mentioned by participants in Chapter 3: W3schools[3] and Smashing Magazine[4]. With respect to Web-based JavaScript programming, W3schools provides detailed coverage of most aspects of the language syntax, from comments and variables to objects and exception handling. However, its presentation style differs from ScriptABLE significantly. W3schools presents a series individual examples on separate web pages consisting of small snippets (≈ 10 lines) of JavaScript and HTML code. Following the collection of related examples,

[3]`http://www.w3schools.com`
[4]`http://www.smashingmagazine.com`

terse syntax-based instruction is presented that explains the basic semantics of various language constructs shown in the examples. ScriptABLE, by contrast, relies on code examples set in the context of larger projects. The example code is revisited multiple times, with progressively more sophistication with each presentation. Instruction in ScriptABLE is interleaved with versions of the code, rather than being presented separately.

A somewhat different approach is taken by the popular site Smashing Magazine. As its name implies, content on this site most often takes the form of magazine-like articles about topics related to web and graphic design. Generally speaking, tips or techniques are presented in the form of rules of thumb or how tos (e.g., actual recent JavaScript articles included "The Seven Deadly Sins of JavaScript Implementation," "Commonly Confused Bits of jQuery," and "The Poetics of Coding"). Here code examples are often presented within the context of the article's topic, but the degree to which any given article contains intentional instruction about the syntax or semantics of the code being used varies widely based on the individual author's purpose. ScriptABLE, on the other hand, uses a common structure for all projects and makes its role in each as an instructional aid explicit. Its concept-driven indexing and tagging further distinguishes it from tutorial sites like Smashing Magazine, which often rely on general tags like "coding" or "JavaScript" to classify articles.

It is also useful to examine how ScriptABLE's design is unique from other forms of case-based learning aids (CBLAs). In Chapter 2 I briefly discussed STABLE (Guzdial & Kehoe, 1998), a Web-based case library that supported undergraduate students in learning about SmallTalk and object-oriented design. STABLE presented example cases as a hierarchy of steps, each with multiple levels of detail. Learners interacted with the STABLE case library by exploring examples and drilling-down on the aspects that they wished to know more about. STABLE presented conceptual information about object-oriented design principles on separate concept pages which linked to (and were linked from) related example pages. This format and intended interaction is common to other CBLAs as well (see, e.g., Bhat & Kolodner, 2009).

ScriptABLE differs most significantly from STABLE in that its projects are presented in their entirety on one page, rather than a series of increasingly concrete representations of the project. While this drill-down approach is quite natural for teaching about abstraction in object-oriented systems alongside a formal course experience, this interaction style does not closely match that of the online resources currently used by web/graphic design end-user programmers (see Chapter 3)—an overarching design goal here. Accordingly, ScriptABLE projects take on an intentionally didactic, tutorial-like style for presenting conceptual information. While I chose to adopt a project structure for ScriptABLE based on the work of Clancy and Linn (1995), exploring other case-based resources that more closely resemble STABLE could be equally promising and is a possible avenue of future research.

5.5 Chapter 5 Summary

In this chapter I have described ScriptABLE, a case-based learning aid for end-user programmers of Photoshop. I began by outlining five basic design criteria derived from the studies in earlier chapters. In highlighting the structure of ScriptABLE's project pages, I demonstrated how they accomplish three of these criteria: focusing on project-driven learning, connecting conceptual content knowledge to code examples, and highlighting the role of errors in code evolution. I addressed the remaining two criteria through design of ScriptABLE's tag hierarchy and the concept coverage of its projects. Tagging provides multiple labels for concepts (both normative terminology and syntactic constructs), and collectively the seven projects contain multiple opportunities for instruction of the six targeted concepts. I concluded the chapter by comparing and contrasting the design of ScriptABLE to existing tutorial sites and the STABLE case-based learning aid.

CHAPTER VI

SCRIPTABLE EVALUATION

The previous chapter outlined the design of ScriptABLE. Ultimately my goal in building ScriptABLE was to develop a resource that could promote measurable learning gains for the targeted normative programming constructs. Further, I intended that such learning gains could occur while using the system during task-oriented problem solving, similar to scripting activities an that end-user programmer would likely encounter. In this chapter, I detail the lab study that I conducted to evaluate ScriptABLE's effectiveness at achieving these goals. I focus primarily on the following research questions in this study:

RQ3: How does the presentation of conceptual information as a case library influence the way end users interact with resources?

RQ4: To what extent does ScriptABLE as a case-based learning aid enable the appropriation of computing knowledge for users actively engaged in project-oriented programming activities?

Unlike previous chapters, I will postpone the operationalization of these research questions until I have outlined necessary details regarding the study design. The remainder of the chapter proceeds as follows. Section 6.1 presents methodological details about the study including recruitment information, participant demographics, an overview of the study design, and the operationalized research questions under investigation. Results are divided into two parts. Section 6.2 explores data about ScriptABLE usage and user satisfaction (RQ3), and Section 6.3 addresses results related to learning differences attributable to use of the system (RQ4). I then distill answers to the research questions and provide extended discussion of the results in Section 6.4. Finally, I address limitations of the study and present alternative study designs that could mitigate these problems in Section 6.5.

6.1 Methods

6.1.1 Recruitment

I recruited participants for this study using a variety of strategies. As in the card sorting study from Chapter 3, solicitation emails were posted to several online Meetup groups in the Atlanta metropolitan area. The groups I contacted all had a primary focus on graphic or web design or use of specific tools like Photoshop or languages like JavaScript. A majority of study participants were recruited through these online groups. I also posted advertisements seeking graphic/web design professionals for the study on Craigslist[1] and in the Savannah College of Art and Design student newspaper.

While these methods yielded ten participants, I ultimately had to also recruit from student populations at Georgia Tech to fill the study with a minimum number of subjects needed for the quantitative analysis described later in this section. My choice to include students was pragmatic, but reasonable given that college students are often used as proxies for practitioners in the end-user programming research literature (see e.g., Ko et al., 2004; Subrahmaniyan et al., 2007; Brandt et al., 2009; Kulesza et al., 2009). Further, I was deliberate in recruiting from student groups that maintained the overall integrity of the participant pool's connection to digital media. I recruited undergraduate Computational Media majors[2] with an email solicitation sent through the student advising staff. Given that many of these majors go on to careers in interactive design or digital media (*Computational Media*, 2008), they share many

[1]`http://www.craigslist.org`

[2]Computational Media is an interdisciplinary major jointly offered by the School of Interactive Computing and the School of Literature, Communication, and Culture.

similarities with the professional populations originally targeted. Though they have prior formal training in programming, it is not the primary focus of their major coursework.

I also recruited directly from two undergraduate programming courses at Georgia Tech. I asked for volunteers in CS1316 (Representing Structure and Behavior, a data structures course in the context of media computation (Guzdial & Ericson, 2010b)) in the second week of class, so that they could complete the study prior to being formally introduced to any of the concepts under investigation. Additionally, undergraduates from CS1315 (Introduction to Media Computation, an introductory programming course for non-CS majors (Guzdial & Ericson, 2010a)) were recruited for the study following the first midterm exam, at which point they had basic familiarity with imperative programming but lacked knowledge of the concepts being studied here. Volunteers from these two courses also had similarities with the professionals I recruited. They had experience using scripting or programming to manipulate images and other media, but they were by no means expert programmers.

Volunteers for the study were sent a screening survey via email to confirm that they met the study criteria. I declined those volunteers who had no prior experience with scripting or programming languages, as a basic ability to read and understand code was required for participation. I also excluded volunteers who had no clear connection to graphic, web, or digital media either by profession or current coursework (in the case of the CS1315/16 students).

All participants were compensated with a $75.00 Amazon gift card upon completion of the study.

6.1.2 Participant Demographics

Eighteen participants, 11 men and 7 women, completed the study. Generally speaking, they represented a wide cross-section of the recruitment pool. A third of participants (6/18) indicated a primary occupation in the web or graphic design industries, about 40% (7/18) were full-time post-secondary students, and the remaining 5 participants reported a combination of the two (e.g., a part-time student with an industry position). A variety of job titles were given, but careers in photography, web development, graphic design, or other design disciplines accounted for over 70% (13/18) of the total.

Participants had a wide variety of prior experience with image editing and scripting tools. Experience with Photoshop ranged from 1 to 18 years with an average of 5.9 years ($\sigma = 4.8$), with participants reporting that they used it 9.6 hours per week, on average ($\sigma = 14.9$[3]). Participants had less exposure to scripting or programming with an average of 4.5 years ($\sigma = 4.1$) of prior experience. On a scale from one (novice) to five (expert), the average self-reported rating of scripting expertise was 2.5 ($\sigma = 1.1$). Thus, on the whole, my participants were experienced Photoshop users with at least basic knowledge of programming.

I again noted a similar pattern of high user preference for resources like tutorials, online documentation, and code examples, echoing the results I discussed earlier in Chapter 3. Every participant said they would be likely or very likely to consult tutorials or frequently asked question documents when attempting to learn something new, and all but one said they would refer to existing examples from which they could borrow ideas or code. Less than half (7/18) said they would attend a class or seminar on the subject and only two would be likely to call technical support phone numbers.

[3]The large standard deviation here is largely the effect of the split in the participant occupations. Professional graphic/web designers reported using Photoshop on the order of 20-40 hours a week, while students indicated significantly less regular use.

Despite my choice to recruit participants from a larger number of sources here, the overall makeup of the participants and their resource preferences was remarkably similar to those in the card sorting study from Chapter 3. The most notable differences were that a higher number of these participants were currently enrolled as students, and they had comparatively less experience with scripting languages (likely a side effect of the increased representation of students).

6.1.3 Study Design

The ScriptABLE evaluation that I conducted was a multi-part lab study with two different treatment conditions allowing for both within and between subjects comparisons. While a laboratory study sacrifices some ecological validity when studying the practices of professionals, it allows for a best-case-scenario where constrained tasks directly map to the instruction available in ScriptABLE. Additionally a lab study was necessary to provide the level of detail required to investigate the proposed research questions. Ultimately, the study involved a series of project-oriented tasks related to the activities, projects, and tools that the target audience uses in their professions. To the extent possible I attempted to balance concerns of ecological validity with pragmatics in order to perform initial verification of ScriptABLE's design goals.

All participants completed two 2-hour study sessions in a controlled usability lab. In each session, participants were asked to complete a series of tasks in an existing Photoshop scripting project written in JavaScript. These tasks consisted of a combination of code comprehension, bug fixing, and feature extension. While different projects were used in the two sessions, the nature of the tasks completed was analogous and the amount of code required to correctly complete the tasks was not significantly different. The first session was designed to establish a baseline for participants' performance on the tasks using only their pre-existing knowledge and information sources. Then, performance during the second session, in which they

Table 9: Study Design and Participant Groups

Participant Group	**Session 1 (Use Internet)**	**Session 2 (Use ScriptABLE)**
Case Library	9	9
Repository	9	9

had access to a version of ScriptABLE, could be compared as a measure of learning. Table 9 illustrates the final study design by showing the number of participants from each group in each session.[4] Additional details about the format of the two sessions is provided below.

6.1.3.1 Session 1 Structure

At the start of the first session, participants completed a brief demographic survey about their scripting experience and background (see Appendix E.1 for the full survey instrument). Following the survey, participants were introduced to the scripting project on which they would be working and shown the ExtendScript Toolkit programming environment provided with Photoshop. The project for this session was a script that automatically extracted meta-data from a collection of photos and wrote it to a comma-separated variable text file. Participants were given time to review the project description, instructions, and primary code file for the project.[5] They were then asked to complete two warmup tasks to further familiarize themselves with the IDE and source code. During these warmup tasks I provided assistance as necessary.

After the warmup tasks, participants were instructed to complete each of the six assigned individual tasks in order in 90 minutes. For the first session, participants were given unrestricted access to the Internet as well as access to any documentation provided in the programming environment's help menu. They were instructed to use these various resources to help them with the tasks. Lastly, the following rules

[4]Despite the visualization in Table 9 it should not be interpreted that participants were assigned to a group prior to session 1. As discussed later, the assignment occurred after session 1 had been completed. Thus all participants had the same environment in session 1, regardless of treatment.

[5]A copy of the instructions provided to participants is given in Appendix E.2. Additionally, I have included listings of the three source code files given to participants for this session in Appendix C.

were used in the event that a participant became stuck on a task and was unable to complete it:

1. If a participant spent a minimum of 15 minutes (i.e., 90 minutes / 6 tasks) on a given problem and indicated verbally that they were at an impasse, they were allowed to move on to the subsequent task.

2. If a participant spent 20 minutes and still had failed to indicate completion of the task, I inquired if they were stuck on the task and invited them to move to the next task.

Regardless of the rule used to move a participant to the next task, I aided them in commenting out any code written for the incomplete task which would prevent the script from functioning as intended on the following tasks.

At the end of the session, I conducted a short semi-structured interview with participants about their experience. I inquired about their confidence in the code they had produced, what they struggled with during the tasks, and their use of various resources during the tasks. Questions from the interview guide are provided in Appendix E.3.

6.1.3.2 Session 2 Structure

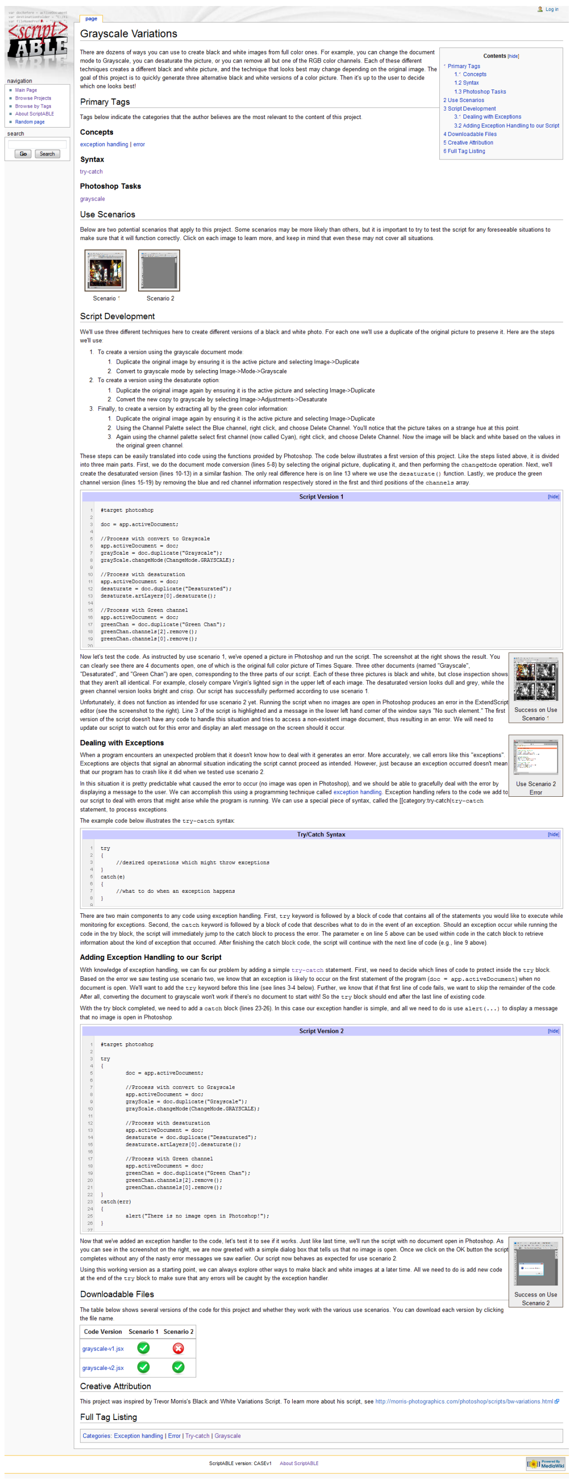
Prior to the start of the second session, I assigned each participant to one of two treatment groups corresponding to alternate versions of ScriptABLE. The first of these two groups would be given access to the full version of ScriptABLE described in Chapter 5. I will refer to this treatment group as the "case library group" or simply as "case" in the remainder of this chapter. The other group of participants received a paired down version of ScriptABLE that was identical to the case library version with the exception that the Script Development section had been removed from each project page (see Figure 21 for screenshots comparing the case library and repository versions of the same project page). This group of participants will hereafter be referred to as the "repository group" (or "repo"). The two alternative versions of ScriptABLE allowed me to explore the relative impact of the various components of the system and compare user performance in the baseline web-only session through a mixture of between and within-subjects analysis.

I assigned participants to a treatment group based on their assessed performance on both coding activities and conceptual answers from session one. In general, I attempted to divide participants with adjacent performance scores into the two groups. Throughout this process, I worked to maintain comparable average performance measures between the treatment groups. By intentionally balancing participants to preserve comparable distributions of performance measures from session one, I sought to eliminate preexisting ability as a potential confound to the extent possible.

The second session was generally scheduled at least one week, but no more than two weeks following the first session. This session proceeded in a similar fashion to the first with one notable exception. Regardless of treatment condition, I performed a brief walkthrough of ScriptABLE's features and gave each participant a five minute period to explore the system on their own. I then introduced the second project: a script which automatically generates thumbnails, preview images, and associated HTML pages for a web gallery from a specified directory of photos.[6]

After completing two analogous warmup tasks, participants were once again given a 90 minute period to complete the individual tasks. The protocol for this portion of the study was identical to that described for session one. Following the individual tasks, participants completed a short user satisfaction survey, and I again conducted a semi-structured interview about their experience during session two (see Appendix E.5 and E.6, respectively).

[6]Appendix E.4 contains the full project description and instructions for session two. See Appendix D for source code listings for the initial files given to participants in session two.



Grayscale Variations

There are dozens of ways you can use to create black and white images from full color ones. For example, you can change the document mode to Grayscale, you can desaturate the picture, or you can remove all but one of the RGB color channels. Each of these different techniques creates a different black and white picture, and the technique that looks best may change depending on the original image. The goal of this project is to quickly generate three alternative black and white versions of a color picture. Then it's up to the user to decide which one looks best!

Contents [hide]

Primary Tags

Tags below indicate the categories that the author believes are the most relevant to the content of this project.

Concepts

exception handling | error

Syntax

try-catch

Photoshop Tasks

grayscale

Use Scenarios

Below are two potential scenarios that apply to this project. Some scenarios may be more likely than others, but it is important to try to test the script for any foreseeable situations to make sure that it will function correctly. Click on each image to learn more, and keep in mind that even these may not cover all situations.

Scenario 1   Scenario 2

Script Development

We'll use three different techniques here to create different versions of a black and white photo. For each one we'll use a duplicate of the original picture to preserve it. Here are the steps we'll use:

1. To create a version using the grayscale document mode:
   1. Duplicate the original image by ensuring it is the active picture and selecting Image->Duplicate
   2. Convert to grayscale mode by selecting Image->Mode->Grayscale
2. To create a version using the desaturate option:
   1. Duplicate the original image again by ensuring it is the active picture and selecting Image->Duplicate
   2. Convert the new copy to grayscale by selecting Image->Adjustments->Desaturate
3. Finally, to create a version by extracting all by the green color information:
   1. Duplicate the original image again by ensuring it is the active picture and selecting Image->Duplicate
   2. Using the Channel Palette select the Blue channel, right click, and choose Delete Channel. You'll notice that the picture takes on a strange hue at this point.
   3. Again using the channel palette select first channel (now called Cyan), right click, and choose Delete Channel. Now the image will be black and white based on the values in the original green channel.

These steps can be easily translated into code using the functions provided by Photoshop. The code below illustrates a first version of this project. Like the steps listed above, it is divided into three main parts. First, we do the document mode conversion (lines 5-8) by selecting the original picture, duplicating it, and then performing the `changeMode` operation. Next, we'll create the desaturated version (lines 10-13) in a similar fashion. The only real difference here is on line 13 where we use the `desaturate()` function. Lastly, we produce the green channel version (lines 15-19) by removing the blue and red channel information respectively stored in the first and third positions of the `channels` array.

Script Version 1 [hide]

```
1   #target photoshop
2
3   doc = app.activeDocument;
4
5   //Process with convert to Grayscale
6   app.activeDocument = doc;
7   grayScale = doc.duplicate("Grayscale");
8   grayScale.changeMode(ChangeMode.GRAYSCALE);
9
10  //Process with desaturation
11  app.activeDocument = doc;
12  desaturate = doc.duplicate("Desaturated");
13  desaturate.artLayers[0].desaturate();
14
15  //Process with Green channel
16  app.activeDocument = doc;
17  greenChan = doc.duplicate("Green Chan");
18  greenChan.channels[2].remove();
19  greenChan.channels[0].remove();
20
```

Now let's test the code. As instructed by use scenario 1, we've opened a picture in Photoshop and run the script. The screenshot at the right shows the result. You can clearly see there are 4 documents open, one of which is the original full color picture of Times Square. Three other documents (named "Grayscale", "Desaturated", and "Green Chan") are open, corresponding to the three parts of our script. Each of these three pictures is black and white, but close inspection shows that they aren't all identical. For example, closely compare Virgin's lighted sign in the upper left of each image. The desaturated version looks dull and grey, while the green channel version looks bright and crisp. Our script has successfully performed according to use scenario 1.

Success on Use Scenario 1

Unfortunately, it does not function as intended for use scenario 2 yet. Running the script when no images are open in Photoshop produces an error in the ExtendScript editor (see the screenshot to the right). Line 3 of the script is highlighted and a message in the lower left hand corner of the window says "No such element." The first version of the script doesn't have any code to handle this situation and tries to access a non-existent image document, thus resulting in an error. We will need to update our script to watch out for this error and display an alert message on the screen should it occur.

Dealing with Exceptions

Use Scenario 2 Error

When a program encounters an unexpected problem that it doesn't know how to deal with it generates an error. More accurately, we call errors like this "exceptions". Exceptions are objects that signal an abnormal situation indicating the script cannot proceed as intended. However, just because an exception occurred doesn't mean that our program has to crash like it did when we tested use scenario 2.

In this situation it is pretty predictable what caused the error to occur (no image was open in Photoshop), and we should be able to gracefully deal with the error by displaying a message to the user. We can accomplish this using a programming technique called exception handling. Exception handling refers to the code we add to our script to deal with errors that might arise while the program is running. We can use a special piece of syntax, called the [[category:try-catch|`try-catch` statement, to process exceptions.

The example code below illustrates the `try-catch` syntax:

Try/Catch Syntax [hide]

```
1   try
2   {
3       //desired operations which might throw exceptions
4   }
5   catch(e)
6   {
7       //what to do when an exception happens
8   }
9
```

There are two main components to any code using exception handling. First, `try` keyword is followed by a block of code that contains all of the statements you would like to execute while monitoring for exceptions. Second, the `catch` keyword is followed by a block of code that describes what to do in the event of an exception. Should an exception occur while running the code in the try block, the script will immediately jump to the catch block to process the error. The parameter `e` on line 5 above can be used within code in the catch block to retrieve information about the kind of exception that occurred. After finishing the catch block code, the script will continue with the next line of code (e.g., line 9 above).

Adding Exception Handling to our Script

With knowledge of exception handling, we can fix our problem by adding a simple `try-catch` statement. First, we need to decide which lines of code to protect inside the `try` block. Based on the error we saw testing use scenario two, we know that an exception is likely to occur on the first statement of the program (`doc = app.activeDocument`) when no document is open. We'll want to add the `try` keyword before this line (see lines 3-4 below). Further, we know that if that first line of code fails, we want to skip the remainder of the code. After all, converting the document to grayscale won't work if there's no document to start with! So the `try` block should end after the last line of existing code.

With the try block completed, we need to add a `catch` block (lines 23-26). In this case our exception handler is simple, and all we need to do is use `alert(...)` to display a message that no image is open in Photoshop.

Script Version 2 [hide]

```
1   #target photoshop
2
3   try
4   {
5           doc = app.activeDocument;
6
7           //Process with convert to Grayscale
8           app.activeDocument = doc;
9           grayScale = doc.duplicate("Grayscale");
10          grayScale.changeMode(ChangeMode.GRAYSCALE);
11
12          //Process with desaturation
13          app.activeDocument = doc;
14          desaturate = doc.duplicate("Desaturated");
15          desaturate.artLayers[0].desaturate();
16
17          //Process with Green channel
18          app.activeDocument = doc;
19          greenChan = doc.duplicate("Green Chan");
20          greenChan.channels[2].remove();
21          greenChan.channels[0].remove();
22  }
23  catch(err)
24  {
25          alert("There is no image open in Photoshop!");
26  }
27
```

Now that we've added an exception handler to the code, let's test it to see if it works. Just like last time, we'll run the script with no document open in Photoshop. As you can see in the screenshot on the right, we are now greeted with a simple dialog box that tells us that no image is open. Once we click on the OK button the script completes without any of the nasty error messages we saw earlier. Our script now behaves as expected for use scenario 2.

Using this working version as a starting point, we can always explore other ways to make black and white images at a later time. All we need to do is add new code at the end of the `try` block to make sure that any errors will be caught by the exception handler.

Success on Use Scenario 2

Downloadable Files

The table below shows several versions of the code for this project and whether they work with the various use scenarios. You can download each version by clicking the file name.

Code Version	Scenario 1	Scenario 2
grayscale-v1.jsx	✓	✗
grayscale-v2.jsx	✓	✓

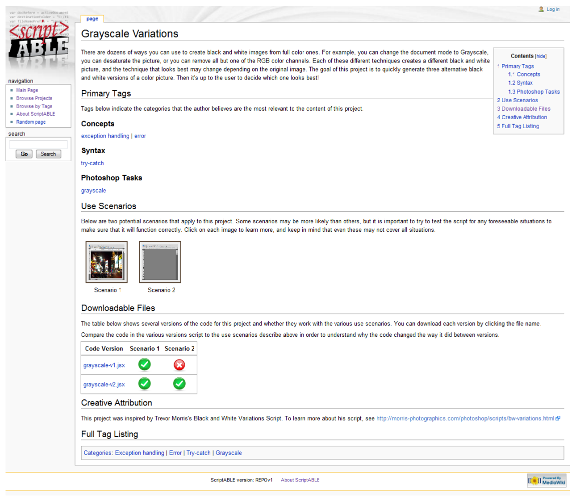
Creative Attribution

This project was inspired by Trevor Morris's Black and White Variations Script. To learn more about his script, see http://morris-photographics.com/photoshop/scripts/bw-variations.html

Full Tag Listing

Categories: Exception handling | Error | Try-catch | Grayscale

ScriptABLE version: CASEv1   About ScriptABLE



Grayscale Variations

There are dozens of ways you can use to create black and white images from full color ones. For example, you can change the document mode to Grayscale, you can desaturate the picture, or you can remove all but one of the RGB color channels. Each of these different techniques creates a different black and white picture, and the technique that looks best may change depending on the original image. The goal of this project is to quickly generate three alternative black and white versions of a color picture. Then it's up to the user to decide which one looks best!

Contents [hide]

Primary Tags

Tags below indicate the categories that the author believes are the most relevant to the content of this project.

Concepts

exception handling | error

Syntax

try-catch

Photoshop Tasks

grayscale

Use Scenarios

Below are two potential scenarios that apply to this project. Some scenarios may be more likely than others, but it is important to try to test the script for any foreseeable situations to make sure that it will function correctly. Click on each image to learn more, and keep in mind that even these may not cover all situations.

Scenario 1   Scenario 2

Downloadable Files

The table below shows several versions of the code for this project and whether they work with the various use scenarios. You can download each version by clicking the file name.

Compare the code in the various versions script to the use scenarios describe above in order to understand why the code changed the way it did between versions.

Code Version	Scenario 1	Scenario 2
grayscale-v1.jsx	✓	✗
grayscale-v2.jsx	✓	✓

Creative Attribution

This project was inspired by Trevor Morris's Black and White Variations Script. To learn more about his script, see http://morris-photographics.com/photoshop/scripts/bw-variations.html

Full Tag Listing

Categories: Exception handling | Error | Try-catch | Grayscale

ScriptABLE version: REPOv1   About ScriptABLE

Figure 21: Visual Comparison of Case Library and Repository Projects

Table 10: Concept Coverage of Tasks

Task	Selection Statement	Functional Decomposition	Exception Handling	Indefinite Loop	Importing Code	Recursion
1	X					
2		X				
3				X		
4		X			X	
5			X			
6						X

6.1.3.3 Projects and Tasks

Each session required participants to complete six tasks within a project. I designed the tasks to cover the six introductory computing topics outlined at the end of Chapter 4: selection statement, indefinite loops, exception handling, recursion, functional decomposition, and importing code. Recall from Chapter 5 that these topics are also the concepts explicitly covered by ScriptABLE's content. Table 10 illustrates the mapping of these concepts to the required tasks. All concepts are covered by at least one task, and all but one task is designed to address a unique concept. Task 4 is an exception to that rule because, in covering the importing code concept (i.e., external code libraries and modules), the related concept of functional decomposition arises naturally.

I designed each of the six tasks to be isomorphic (to the extent possible) in the two sessions. Despite obvious surface-level changes like variable and function names caused by differences in the assigned project code, the underlying problem to be solved was identical for a given task. In fact, where participants needed to produce additional JavaScript code, the ideal solution was largely the same across the two sessions. More detail about the various tasks in the two sessions appears in Table 11.

Each task consisted of two or more sub-tasks. These related sub-tasks were used to assess the intended construct from different perspectives. For example, consider task

Table 11: Assigned Tasks by Session

Task Number	Question Type	Session 1 Task	Session 2 Task
1A	describe	setting `colorMode` string	setting `portraitFlag`
1B	code		
2A	code	`getPropertyValue` function	`applyWatermark` function
2B	strategize		
3A	strategize	dealing with cancel	dealing with cancel
3B	code		
4A	describe	explaining `writeRow`	explaining `writeHeader`
4B	strategize		
5A	describe	skipping corrupt files	skipping non-image files
5B	strategize		
5C	code		
6A	describe	processing subfolders	processing subfolders
6B	strategize		

number five from session one. This task asked participants to diagnose and debug an error in the script when it encounters a corrupt picture file while processing an input directory. In this situation an unchecked exception would be thrown by the runtime engine and participants had to determine what happened, why it happened, and how to correctly modify the script to gracefully avoid the error. The task consisted of three separate prompts indicated as 5A, 5B, and 5C in Table 11.

The different question types (describe, strategize, and code, respectively) indicate the nature of the prompt. Like all "describe" prompts, 5A instructed a participant to run the program in a particular manner and describe what happened based on the execution result and any input files. For this sub-task, participants were expected to indicate that an error had occurred and name the image file they suspected caused the program to crash. 5B, a "strategize" question, asked participants to devise a strategy for preventing the error and outline the programming technique they would use in plain English. Here, I expected an answer involving exception handling mechanisms. Lastly, "code" questions like 5C required participants to edit the script to implement their proposed strategy. A correct answer for 5C required a participant to write a

Table 12: Task Categorization on Bloom's Taxonomy

Task	**Remember**	**Understand**	**Apply**	**Analyze**	**Evaluate**	**Create**
Task 1				Part A		Part B
Task 2					Part B	Part A
Task 3					Part A	Part B
Task 4				Part A	Part B	
Task 5				Part A	Part B	Part C
Task 6				Part A	Part B	

`try-catch` statement in the code, correctly defining the scope of the try and catch blocks so that erroneous input files would be skipped without preempting program execution.

The three question types make it possible to investigate both conceptual knowledge about a topic in addition to practical code development ability. This is important because it was foreseeable that someone could correctly understand what was needed to complete a task, but struggle with elements of JavaScript syntax in implementing their idea. Additionally, the sub-tasks were designed to assess knowledge at different levels along the cognitive dimension of understanding. Table 12 illustrates the cognitive sophistication of sub-tasks using Bloom's Taxonomy.

Originally outlined in 1956, Bloom's Taxonomy is a classification scheme for various goals and objectives in the educational domain (Bloom, Englehart, Furst, Hill, & Krathwohl, 1956). It is a tool intended to aid instructional designers in building curricula and assessment practices while also enabling educators to better communicate their course objectives and outcomes. The original taxonomy consisted of six categories, with each subsequent category building on those prior: Knowledge, Comprehension, Application, Analysis, Synthesis, and Evaluation. This classification was recently revised by Anderson et al. (2001). The revision clarified the original category labels for the cognitive dimension and provided additional classification guidelines for assessment activities. I have used the revised category names and definitions in my placement of sub-tasks in Table 12.

By comparing question types for sub-tasks to their corresponding taxonomic classifications one notices an exclusive relationship. All "describe" prompts fall in the analyze category, "strategize" prompts appear as evaluation measures, and "code" sub-tasks are categorized at the create level. In evaluating ScriptABLE's ability to promote learning, I focus on the most cognitively sophisticated measures—sub-tasks from the evaluate and create levels. I use elements from the evaluate column as indicators of conceptual understanding about programming concepts, and I use those in the create column as measures of participants' coding ability (i.e., their ability to translate their abstract idea into a syntactically correct solution).

6.1.4 Operationalized Research Questions

With this background about the evaluation study, I now introduce the specific research questions driving the investigation presented in this chapter. Questions RQ3.1 and RQ3.2 below address issues of resource usage patterns and perceptions of resource usefulness; the remaining three questions (RQs 4.1–4.3) deal with the degree to which the two versions of ScriptABLE promote the appropriation of conceptual and syntactic knowledge of programming concepts.

> **RQ3.1:** How do usage patterns of web-based references differ comparing participants' use of unrestricted Internet access and ScriptABLE in its repository and case-library forms?
>
> **RQ3.2:** How do participants' perceptions of value and usefulness compare for the two alternate versions of ScriptABLE?
>
> **RQ4.1:** To what extent do participants write better code, both conceptually and syntactically, when given the repository or case library form of ScriptABLE?

RQ4.2: To what extent do participants with access to the case library form of ScriptABLE produce more coherent conceptual explanations about project-oriented programming activities than those using the repository version?

RQ4.3: Is there a measurable difference in participants' self-identified learning of programming concepts attributable to using a particular version of ScriptABLE?

In order to answer these research questions, I relied primarily on quantitative measures. My analysis made use of non-parametric statistics, as was appropriate in situations like this involving small sample sizes where assumptions of normally distributed data cannot be made. In particular I employed the Wilcoxon signed-rank test for within-subjects comparisons, and both the Mann-Whitney U and Fisher's exact tests for between-subjects comparisons. These statistical tests mitigate for skewed data points (e.g., outliers) that could otherwise negatively affect the reliability and validity of the results. I will not only report statistical significance at the $\alpha = 0.05$ level, but also at the $0.05 \leq \alpha < 0.1$ level (denoted as "marginally significant"). Given the limited sample size in this study and the fact that these marginal values approach the standard $\alpha = 0.05$ significance level, their inclusion and discussion has merit.

6.2 Evaluating ScriptABLE Usage

In evaluating user interaction with the case library, I make use of two primary sets of metrics. First, I will examine and compare usage data from web activity gathered during each of the two study sessions. Then I will present user satisfaction data to explore participants' perceptions of the two ScriptABLE versions.

6.2.1 Resource Usage

Web usage histories were captured during each session using the Coscripter Reusable History add-on for Firefox (formerly called ActionShot (Li, Nichols, Lau, Drews, & Cypher, 2010)).[7] From the history data I tabulated values for each of the following: total number of pages visited, number of unique pages, total searches performed (using Google in session one and using the wiki search tool in session two), number of unique searches, and the number of projects and concept tags viewed during session two.

Table 13 presents resource usage data for the two sessions. The data gathered from session one appear first with the prefix 'S1' in the variable column. A double-line divides the first and second session values, and the data for each variable is disaggregated by treatment group. In addition to the mean and median values, I present the results of a Mann-Whitney U test comparing the repository versus case library groups on each variable (i.e., these are between subjects comparisons). Marginally significant p-values ($0.05 \leq p < 0.10$) are denoted with a single asterisk (*), and significant p-values ($p < 0.05$) are indicated by a double asterisk (**).

Comparing the same measures between the two sessions yields some interesting results. For both treatment groups, the average number of unique pages visited during the session increased (case: 13.00 to 19.22 and repo: 18.00 to 25.33). A Wilcoxon Signed Rank test[8] confirmed that participants indeed visited significantly more pages when they were given a version of ScriptABLE ($Z = -2.265$, $n = 18$, $p = 0.022$). Participants viewed more content pages on the web when using ScriptABLE, despite having access to considerably less information than when their Internet access was unrestricted. Several participants commented that, compared to the Internet,

[7]Histories from the second session were cleaned to remove data points from the ScriptABLE demo period.

[8]All p-values reported for signed rank tests are the exact 2-tailed significance values.

Table 13: Comparison on Resource Usage by Group

Variable	**Group**	**Mean**	**Median**	**Mean Rank**	**Mann-Whitney U**	**Mann-Whitney p-value**[1]
S1 Total Pages	case	21.44	19	8.44	31.00	0.423
	repo	29.78	33	10.56		
S1 Unique Pages	case	13.00	10	8.61	32.50	0.504
	repo	18.00	17	10.39		
S1 Total Searches	case	11.11	12	8.67	33.00	0.531
	repo	14.44	11	10.33		
S1 Unique Searches	case	5.56	5	7.94	26.50	0.227
	repo	8.56	9	11.06		
S2 Total Pages	case	68.67	54	8.11	28.00	0.286
	repo	93.67	70	10.89		
S2 Unique Pages	case	19.22	20	8.17	28.50	0.306
	repo	25.33	20	10.83		
S2 Total Searches	case	1.56	1	7.22	20.00	0.075*
	repo	5.00	2	11.78		
S2 Unique Searches	case	0.78	1	6.56	14.00	0.016**
	repo	2.44	2	12.44		
Projects Viewed	case	4.67	4	8.78	34.00	0.584
	repo	5.11	5	10.22		
Concept Tags Viewed	case	4.67	5	10.78	29.00	0.321
	repo	3.11	2	8.22		

[1] All Mann-Whitney p-values given in this chapter are 2-tailed, exact significance values computed using correction for tied ranks.

ScriptABLE was more helpful and made them feel more comfortable. I observed this among participants from both treatment groups. For example:

> P10c: The title, ScriptABLE, made it seem like it was for novices. And that it wasn't going to use terms and things that I didn't understand without explaining them or providing resources to explain those terms. ... I had help this time. Structured help that would make sense to me. I don't know where to start with the Web sometimes. Sometimes I go off on tangents, [ScriptABLE] kind of kept it concentrated. I wasn't shown unnecessary information.
>
> P11r: It's funny because I felt more confident going into it last week having the Internet, but with defined sources this time that like I knew which task they would perform and what they would tell me, it was a lot easier. ... rather than going to the vast Internet and trying to figure things out from there.
>
> P16c: When you're searching the Internet you get a whole bunch of junk and a lot of people asking questions that maybe won't apply. So you really have to bend whatever their question was to what your question is, and you're not even sure if those answers they're getting are right. And having something that was written for the sole purpose of giving answers about scripting means that your information is in one place.
>
> P17r: With the Internet you just got so fed up, so at the point that you put down a search, you almost have to skip the first 8 or 9 before you found anything that was going to be helpful. I mean even with Google. And with [ScriptABLE] it was—it's all kind of laid out there for you and the tools are there, you just have to find it.

Perceptions like these are indicative of the increased value that participants gave to ScriptABLE. The marked increase in web activity during session two suggests that these perceptions may have encouraged users to refer to the system for their questions more frequently.

Furthermore, there were also significant differences in the search behaviors of both groups during the two sessions. The mean number of unique searches performed during the session decreased between sessions one and two (case: 5.56 to 0.78 and repo: 8.56 to 2.44). This shift was statistically significant ($Z = -3.514$, $n = 18$, $p < 0.001$). When given access to the Internet at large, participants relied on an information seeking strategy that can be best characterized as search-driven. Over 40% of the unique pages viewed were search result listings in the first session. Generally web activity began with a Google search query, and then search results were selectively examined, with participants going back to the search result page relatively quickly.

However, the navigation behavior when using either version of ScriptABLE was markedly different. Browsing the system through the internal links was the typical behavior in session two, with participants occasionally returning to the tag index or the project listing pages—fewer than 10% of the unique pages were search results. Based on interview comments like those above, I postulate that the shift from searching to browsing is the result of ScriptABLE's content being more focused and intentionally interconnected than other resources commonly found on the Web.

To look more closely at search behaviors, I compared search queries used in both sessions of the study. Figure 22 lists all unique strings used to search with Google during session one, and Figure 23 shows the unique ScriptABLE search engine input strings used in session two. The sheer difference in the sizes of these lists is striking, but there are also other interesting observations. Many of the Internet searches included verbatim substrings from the source code or instructions given to participants (e.g., the `colorMode` variable name, the `csvFile.writeRow` method call, the `isColorImage` helper function). While this also occurred with ScriptABLE (e.g., `writeHeader`), queries like this were much less frequent in session two. The use of programming language names to contextualize searches also appeared significantly different in the two sessions. With the Internet, nearly every search string included a reference to a programming language like JavaScript, ExtendScript, or Java. Unfortunately many of these searches yielded unrelated results and participants expended considerable energy looking at topics that were not applicable, especially in the case when they mistakenly used Java in their query. Note in Figure 23 that no ScriptABLE search string made such reference to a language. Participants trusted the context of the system to eliminate their need to refine their queries.

There was also a measurable difference in search behaviors when comparing use of the two ScriptABLE versions. For participants who received the case library version, the decreased search frequency was even more pronounced; on average, they performed less than one search query in the system. As indicated in Table 13, there was a statistically significant difference in the search behavior between the case library and repository treatment groups ($U = 14.00$, $n1 = n2 = 9$, $p = 0.016$). Repository users conducted considerably more searches (though still fewer than they did in session one). I believe this stems from the relative lack of explanatory content in the repository version; when browsing the project pages failed to yield an answer, participants simply resorted to their default search strategies used on the Web.

Beyond search behaviors, I noted no statistically significant differences on other between-subjects indicators (see Table 13). The repository and case library groups did not appear to access project pages, concept tag pages, or other pages in any noticeably different patterns.

6.2.2 Perception of ScriptABLE's Usefulness

In addition to usage data, I also gathered indicators of participants' satisfaction with ScriptABLE to further examine users' perceptions about the value and usefulness of

detecting color in images photoshop
detecting colormode photoshop javascript
photoshop javascript color document mode
error handling javascript
detecting color in images photoshop javascript
photoshop javascript document mode
conditional color detection photoshop color document mode
javascript user prompt file
javascript user prompt
push java
javascript colorMode ExtendScript Tookit
javascript
ExtendScript Toolkit documentation
extendscript toolkit cs4
javascript colorMode
ExtendScript Toolkit
javascript colorMode ExtendScript Toolkit
getpropertyvalue
color Mode in javascript
colorMode in javascript
javascript try catch
exif
exif colormode in photoshop file info
view .jpg tags in photoshop
exif colormode
javascript print to console
simple JavaScript function
JavaScript File.openDialog
simple JavaScript function with input parameter
exif parameter
adobe extendscript tutorial functions
how to define a function in adobe extendscript
error handling extendscript
javascript isColorImage
javascript "open options are incorrect"
csv file.writeRow
writing functions in javascript
javascript "open options are incorrect
javascript isBWImage
javascript basics
javascript color codes
can javascript identify color
javascript code reprompting
app.open
javascript color selection
javascript colors
csvfile.writeRow
javascript alert
javascript grayscale
javascript dectect RGB
javascript defining in writer
create function java
javascript
boolean javascript
javascript define function
javascript rgb object
javascript for loops
javascript detect RGB
conditionals javascript
javascript detect color
cs4 extendkit functions
javascript equals string
cannot open the file because the open options are incorrect
try catch javascript
javascript undefined
javascript functions syntax
javascript throw
javascript functions
adobe application object
javascript output
java tutorial + functions
writing functions in javascript
java tutorial boolean
for loops javascript
java tutorial + function notation
setting colorMode in photoshop coding
setting colorMode in Java
setting color Mode in java
setting color Mode in java + boolean
setting colorMode in java for photoshop
setting colorMode in java
batch file stop services
getPropertyValue java
javascript cancel prompt
javascript how to skip a file in a batch
isColorImage(imgDocument) javascript
how to use getPropertyValue java
javascript prompt loop
javascript repromtping
javascript if then
detect color mode of image in photoshop
photoshop scripting color mode
photoshop, iscolorimage
extendscript toolkit tutorial
adobe extendscript iscolorimge
extendscript error in opening file
extendscript error in opening image file
extendscript cannot open files because the open options are incorrect
coding functions in extendscript
adobe extendscript is color image
javascript alert cancel button
DocumentInfo photoshop
user click cancel openDialog java
DocumentInfo object
File Dialog javascript cancel
user presses cancel dialog javascript
click cancel on openDialog java
user cancel dialog javascript
stop for loop java
javascript delete confirmation dialog
functions javascript
DocumentInfo
java try catch
DocumentInfo photoshop cs4
boolean javascript color mode
boolean expression java
boolean
iscolorimage
boolean expression
extendscript try catch
colorMode extendscript
extendscript javascript
javascript function
CSVWriter
javascript error handling
color mode photoshop cs4
javascript null

Figure 22: Internet Search Queries from Session 1

imgDocument.height
imgDocument
img
try-catch
function
close
while
if
height
writeheader
try
if else than
image
false
true false

true
exception
variable name expected
open options
catch
image
tif
syntax error
cancel
dialog
writeHeader
error
open
try

Figure 23: ScriptABLE Search Queries from Session 2

the system's content. Four Likert-scale questions were asked on the survey at the end of the second session. These prompts asked participants to rate ScriptABLE on a scale from one (strongly disagree) to five (strongly agree) for whether it helped them in the session, whether they felt it was a good way to learn new things for them or others, and whether they would use it in their daily scripting projects. These results are summarized in Table 14.

Fisher's exact tests conducted on each of the response distributions only showed a significant difference for whether ScriptABLE was a good tool for other people to learn. However, closer examination of the actual response pattern between the treatment groups shows that, from a practical standpoint, this difference is not all that different. Repository group participants were more likely to use the strongly agree category (resulting in the significant Fisher's exact test result), but the overall number of participants agreeing with the statement is comparable. In fact, a Mann-Whitney test comparing the rank distributions for these two groups on the same question showed no statistically significant difference. In general, there were no meaningful differences in user satisfaction reported on these satisfaction indicators.

The survey data indicates that participants held largely similar beliefs about how useful ScriptABLE was, regardless of whether they saw the case library or repository

Table 14: User Satisfaction Likert Responses

Group	SD (1)	D (2)	N (3)	A (4)	SA (5)	Fisher p-value	Mann-Whitney U	Mann-Whitney p-value
ScriptABLE helped me complete the tasks today								
case	1	2	3	2	1	0.303	28.00	0.227
repo	0	0	4	5	0			
ScriptABLE is a good resource for me to learn new things.								
case	1	1	0	7	0	0.229	36.50	0.791
repo	1	0	2	4	2			
ScriptABLE is a good resource for others to learn new things.								
case	0	0	2	7	0	0.043**	32.00	0.519
repo	1	0	2	2	4			
I would use ScriptABLE in my daily scripting projects.								
case	2	0	2	4	1	0.827	37.50	0.813
repo	1	2	2	3	1			

Table 15: Which Resource Helped You the Most Today?

Group	Object-Model Viewer	ScriptABLE
case	4	5
repo	6	3

version. However, when I asked participants during the interview following session two which resource was most useful in completing the tasks today, I observed an interesting difference. All participants mentioned either ScriptABLE or the Object Model Viewer (a built-in API tool accessed through the IDE's help menu). Table 15 shows their responses. The majority of participants who had used the case library version preferred ScriptABLE, while those who had used the repository version preferred the Object Model Viewer two to one.

When asked to elaborate on why they preferred the Object Model Viewer to ScriptABLE participants said things like:

> P16c: I thought its search capabilities were better than ScriptABLE. And, not to dis [sic] on ScriptABLE, but the answers that ScriptABLE gave were stuff that I already knew basically. But if I didn't know it already it

> would be extremely useful, especially since it had all that example code.
>
> P2r: ScriptABLE helped a little bit but I think I still got more help from the toolkit. Just because most of the things that I wanted to know were basic like API things.
>
> P15r: I knew within that I could search any object or method that JavaScript uses, or you know Photoshop/JavaScript whatever, and that it would be there. Unlike, I had the feeling that ScriptABLE was just catering to some small minority of terms and tasks, and not necessarily JavaScript.
>
> P20r: I feel like I did better with the Object Model Viewer. The things I typed in the search box it actually did help me, even in the warm-up activity. I was able to use that information better. ScriptABLE has very specific projects, and I wasn't, like it had a grayscale project, so I didn't look at that. I did for a second to see if the code would help me at all, but it was kind of specific so I wasn't—some of the projects on there weren't even relevant to this, because this was just compiling like a gallery.

Although not statistically significant, this difference in resource preferences may be indicative of underlying perceptions about the two versions of ScriptABLE. For those who simply needed an API reference like P16c and P2r, the Object Model Viewer was an obvious choice, regardless of the version of ScriptABLE used. For those who needed help with JavaScript concepts, it appeared that repository group was more likely to discount ScriptABLE as a collection of irrelevant projects as P15r and P20r did. Similar participants who were given the case library, on the other hand, identified it as an important resource and felt it help them complete the tasks.

6.3 ScriptABLE as a Task-Oriented Learning Aid

My evaluation of ScriptABLE's effectiveness as a scaffold for learning computing concepts is divided into three parts. First I examine participants' ability to complete syntactically and semantically correct coding tasks, then I present findings related to their ability to answer conceptual questions about the tasks, and lastly I present data about self-reported learning gains.

6.3.1 Code Correctness

As I discussed earlier in this chapter, there were four 'code' questions (see Table 11). To analyze the code produced for these questions I developed a simple rubric with four ordered categories to be applied to the final script. The categories, from most correct to least, are:

4 points Code functions correctly, uses the intended construct, closely resembling the ideal solution.

3 points Code functions correctly, uses the intended construct, but also includes unnecessary additional constructs that could be removed or otherwise simplified.

2 points Code does not function correctly but uses the intended construct.

1 points Code does not function correctly and does not use the intended construct.

0 points No code edited or inserted.

Two independent raters applied the rubric to all 36 scripts produced in the two sessions. Ratings were compared and any disagreements were collaboratively reconciled between the raters to produce the final value. I then aggregated ratings for each of the four coding questions to produce a single value between 0 and 15 for code performance in a session. Table 16 presents summary statistics for participants' code performance during session one and session two, and it also presents data for

Table 16: Code Performance Statistics

Variable	Group	Mean	Median	Mean Rank	Mann-Whitney U	Mann-Whitney p-value
Session 1 Code	case	7.78	6	9.83	37.50	0.812
	repo	7.33	5	9.17		
Session 2 Code	case	10.11	13	9.28	38.50	0.872
	repo	10.11	8	9.72		
Code Improvement	case	2.33	2	8.72	33.50	0.558
	repo	2.78	3	10.28		

the degree to which their aggregate code score improved (calculated simply as session two score minus session one score). The rightmost columns show Mann-Whitney test results for comparisons between the two treatment groups.

You can see from Table 16 that my intentional balancing of participants based on session one performance yielded treatment groups with very similar performance distributions. A Mann-Whitney U test comparing session one performance between the groups resulted in non-significance ($p = 0.812$). Thus, it is reasonable to consider these two groups as comparable in their demonstrated coding ability going into the second session.

Without a doubt, participants' code scores improved across the board from session one to session two. The mean and median scores for both the case library and repository groups increased markedly. A Wilcoxon signed-rank test confirmed that there was a significant positive shift in scores ($Z = -3.172$, $n = 18$, $p < 0.001$). However, I did not observe significant differences between the groups in their overall performance on the second task ($U = 37.50$, $n1 = n2 = 9$, $p = 0.872$) nor in the degree to which their code improved across the sessions ($U = 33.50$, $n1 = n2 = 9$, $p = 0.558$). In other words, code scores improved for both groups, but neither group improved significantly more than the other.

There are several likely explanations for these observations about coding ability. For example, there could have been learning effects from session one (more than one participant commented on the similarity in tasks), or there could have been differences between the projects that made the second session's code easier to understand and/or edit. However, the important result for the research question I posed is that, while participants did better on the second session, these gains were not attributable to the version of ScriptABLE that they used during that session.

6.3.2 Conceptual Answers

Having examined participants' ability to write code, I now turn to an analysis of participants' ability to complete open-ended questions about programming concepts. Recall from earlier in the chapter that I will be using responses from the five "strategize" sub-tasks as indicators of conceptual understanding. These prompts asked participants to outline a programming technique or concept that would be applicable in the current situation and would allow them to overcome an issue in the code. Responses to these prompts were written in plain English.

Similar to the analysis of code performance, I developed a rubric for judging the quality and correctness of conceptual responses. Again, two independent raters applied the rubric to all 180 responses gathered during the two sessions and collaboratively reconciled any disagreements. The final rubric used for each question had the following four categories, from ordered most to least correct:

3 points Response correctly identifies the intended construct by name, using normative terminology.

2 points Response correctly describes an approach which makes use of the intended construct, but does not explicitly use normative terminology in the answer.

1 point Response does not address the intended construct, but does exhibit an algorithmic or programmatic approach to solving the problem which could work under limited conditions.

Table 17: Conceptual Performance Statistics

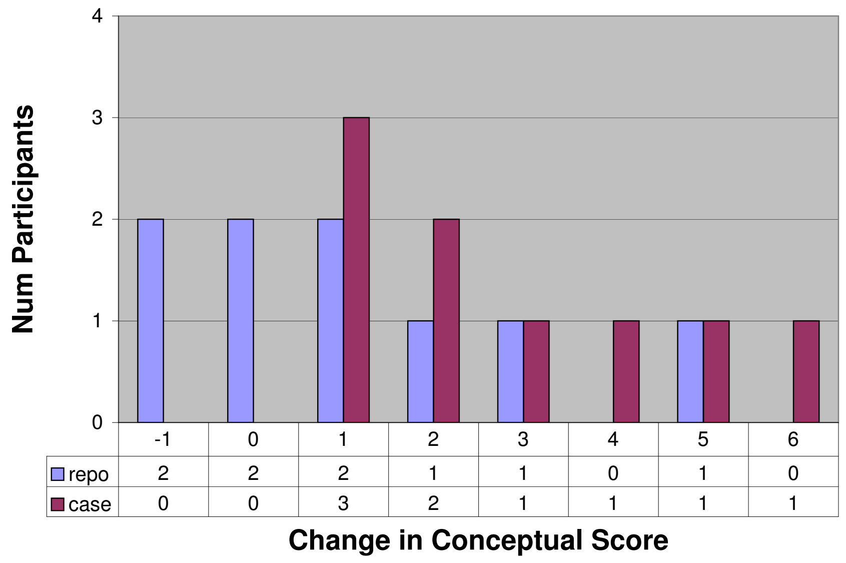
Variable	Group	Mean	Median	Mean Rank	Mann-Whitney U	Mann-Whitney p-value
Session 1 Concept	case	7.44	7	9.22	38.00	0.862
	repo	7.67	8	9.78		
Session 2 Concept	case	10.22	10	10.56	31.00	0.426
	repo	8.78	9	8.44		
Concept Improvement	case	2.78	2	11.78	20.00	0.071*
	repo	1.11	1	7.22		

0 points No answer given, or response was nonsensical or wholly unrelated to the issue being addressed.

I combined conceptual performance scores for individual sub-tasks in a session to create a single value for each participant ranging from zero to 15. I then used this aggregate value as the indicator of each participants' performance on conceptual questions. I have summarized statistics for conceptual performance in Table 17. As I did earlier for code performance, this table illustrates the mean and median conceptual performance values in sessions one and two and the observed improvement in the answers disaggregated by treatment group. Statistics for between-subjects comparisons on these three variables appear in the rightmost columns.

The treatment groups performed quite comparably on conceptual questions in the first session as a result of my deliberate efforts to balance the groups. You can see in Table 17 that the mean and median values for session one performance were nearly identical, and a Mann-Whitney U test showed a similarly non-significant result ($U = 38.00$, $n1 = n2 = 9$, $p = 0.862$). Thus, the groups were alike enough to permit meaningful comparisons.

There was a statistically significant increase in the quality of conceptual responses from session one to session two (Wilcoxon signed-rank, $Z = -3.135$, $n = 18$, $p = 0.001$). That is, their responses were more technically accurate and were more likely to make use of normative computing terminology. However, unlike the code



	-1	0	1	2	3	4	5	6
🟦 repo	2	2	2	1	1	0	1	0
🟥 case	0	0	3	2	1	1	1	1

Figure 24: Conceptual Score Increase by Treatment

measures, there were detectable differences in conceptual performance tied to which version of ScriptABLE participants received. Simply looking at the average values for session two it appeared that the case library group outperformed the repository group (see Table 17). The session two performance values alone were not different enough to result in a statistically significant finding, but the difference became much more apparent upon comparing the improvement in conceptual responses. I noted a marginally significant difference in the degree to which answers improved ($U = 20.00$, $n1 = n2 = 9$, $p = 0.071$); participants who used the case library form of ScriptABLE improved on average about 2.5 times better than those who used the repository version.

Figure 24 presents a histogram illustrating the observed shift in performance values for the treatment groups. As you can see, every participant in the case library group improved by at least one point, and several people improved by three or more points. Those in the repository condition improved with much less regularity and four participants showed either negative or no improvement.

These results indicate a measurable difference in the quality and correctness of conceptual responses that can be attributed, in some part, to the version of ScriptABLE used. Participants who used the full case library version described in Chapter 5 exhibited significantly more growth in the technical correctness of their responses and their use of normative terminology to describe their proposed solutions. As an objective indicator of conceptual learning, this serves as evidence that the case library was able to promote transfer of normative computing knowledge from ScriptABLE's projects and indices, at least in the short term. Whether participants retain this knowledge beyond the study session is beyond the scope of this dissertation. I will return to these results later in the discussion section where I will elaborate on reasons for the case library's success here, but not on code performance.

6.3.3 Self-Reported Learning

The last measure of ScriptABLE's effectiveness makes use of self-reported learning gains. On the exit survey following session two, participants were asked to rate their current understanding of the six concepts being studied. A follow-up question asked them to compare their current level of knowledge on each concept to what it was prior to the second study session on a 5-point Likert-type scale ranging from significantly worse (1) to significantly better (5). Participant's responses on this prompt are tabulated in Table 18. The table also indicates Fisher's exact test and Mann-Whitney U test statistics for comparisons between the case library and repository groups.

On the whole, little can be said based on the self-reported learning gains. The majority of participants indicated that their level of understanding was unchanged for all of the concepts except exception handling. Only one participant said they were more confused about a topic (recursion) after completing the second session. The only statistically significant difference in the distributions of self-reported learning gains was for importing code (Mann-Whitney test, $U = 18.00$, $n1 = n2 = 9$, $p = 0.029$), with five people in the repository group indicating that they felt their knowledge

Table 18: Self-Reported Concept Learning Responses

Concept	**Group**	**Sig. Worse (1)**	**Worse (2)**	**About the Same (3)**	**Better (4)**	**Sig. Better (5)**	**Fisher p-value**	**Mann-Whitney U**	**Mann-Whitney p-value**
selection statement	cases	0	0	9	0	0	0.471	31.50	0.471
	repo	0	0	7	0	2			
functional decomposition	cases	0	0	7	2	0	0.620	30.50	0.457
	repo	0	0	5	3	1			
definite loop	cases	0	0	8	1	0	0.365	26.00	0.188
	repo	0	0	5	2	2			
importing code	cases	0	0	9	0	0	0.029**	18.00	0.029**
	repo	0	0	4	4	1			
exception handling	cases	0	0	5	4	0	0.311	25.50	0.209
	repo	0	0	3	3	3			
recursion	cases	0	0	5	4	0	0.620	35.50	0.698
	repo	0	1	5	2	1			

increased and no one in the case library group reporting a change.

Ultimately I believe inferences based on these self-report prompts would be precarious. When one considers these reported gains alongside participants' rating of their knowledge there are several inconsistencies. For example, some participants indicated that they did not recognize a term, but at the same time said their knowledge of the term was significantly better on this prompt. Further, it would appear that overall participants in the repository group reported significantly more learning than their case library counterparts, despite the fact that they performed similarly on coding questions and worse on conceptual questions.

There are perils with self-reported data, and I believe there was a significant participant effect (Gay & Airasian, 2000) at work here. Participants were inclined to say that their understanding improved for these topics because they believed they were supposed to have learned. Accordingly I believe I lack sufficiently reliable data to address the research question originally posed about self-identified learning gains (RQ 4.3: Is there a measurable difference in participants' self-identified learning of programming concepts attributable to using a particular version of ScriptABLE?). As such, I will eliminate it from further discussion and instead rely on the objective performance measures of coding ability and conceptual knowledge in answering the top level research question (RQ4).

6.4 Discussion

In this section I explicitly answer each of the four remaining operationalized research questions under investigation in this study. I also provide additional commentary for the difference in observed results for the final two research questions.

6.4.1 Revisiting the Research Questions

6.4.1.1 How do usage patterns of web-based references differ comparing participants' use of unrestricted Internet access and ScriptABLE in its repository and case-library forms? (RQ3.1)

There were clear differences in how participants used the web as a resource in the two sessions. When given access to the Internet at large in the first session, participants' usage revolved around search queries. Individual pages in the result listings were visited briefly, and then participants returned to search results. On the other hand, when given ScriptABLE, participants were far more likely to exhibit browsing behaviors using the internal links in the system to move through content. Many fewer searches were performed in the second session, and participants using the case library form of ScriptABLE rarely used the search feature at all.

This result might be explained by information foraging theory (Pirolli, 2007), a model of information seeking behavior that likens human cognitive information seeking strategies to the foraging tactics of animals in the wild. Essentially, Pirolli (2007) posits that a user looking for information attempts to maximize the information gained while minimizing the effort expended during the search. Further, users make decisions about where to look next based on *information scent*—that is, they actively predict which path will lead to optimum information gain using cues in the environment (e.g., links, page rankings).

I argue that the case library form of ScriptABLE inherently has a higher degree of information scent as a result of the additional information presented in the Script Development section. The content of that section contextualizes programming content knowledge with example code and test scenarios. This additional context enables users to see the relevance of linked pages, promoting browsing behaviors. In the case of the repository, I believe that users may have perceived a low degree of information scent. This is a plausible interpretation of the comments from repository participants indicating they believed ScriptABLE's content was irrelevant for their tasks.

Further it may explain the observation of increased use of search by the repository group—participants attempted to rely on their default web search behaviors to find information that they had trouble locating on their own. Unfortunately their search queries often contained no matches, due to the limited content in the repository.

6.4.1.2 How do participants' perceptions of value and usefulness compare for the two alternate versions of ScriptABLE? (RQ3.2)

Overall, there were somewhat mixed results related to users' perceptions of the two versions of ScriptABLE. On survey questions about the system, there were no appreciable differences for users' ratings of the case library and repository. Qualitative interview data suggest that perhaps case library users valued it more in helping them complete the tasks than those who were exposed to the repository version. I believe that participants in the repository group had more difficulty in seeing the relevance of the project contents to their current task because of the missing Script Development section. Recall that this section presents the complete narrative of how a script is written to meet the project goals. In order to understand how the tagged concepts related to the use scenarios and the different versions of the code, repository participants had to piece together the narrative on their own by comparing each of these separate elements in the interface. The repository provided even less guidance than the case library, and in this setting the increased cognitive load required of participants may have turned them away from the system (Kirschner, Sweller, and Clark (2006) outline a similar argument regarding the failure of other forms of minimally guided instruction).

In fact, the few repository group participants who preferred ScriptABLE did seem to make effective use of the information given in the tag pages, use scenarios, and code examples by repeatedly accessing in an intentional manner. That said, they also pointed out that they felt something was missing from ScriptABLE in the interview portion of the study. For example, consider these comments from two participants who preferred ScriptABLE about what resources they would like to have had access to during the session:

> P17r: Maybe some of these [gestures to a tag page in ScriptABLE's index on screen], if they were just a little more in-depth. The syntax [tag pages] that you have here, and then it just directs you to one of these projects. It doesn't really explain to you how it could be used. There's not much of an explanation behind that.
>
> P9r: I was surprised in ScriptABLE the scenarios were very deemphasized. I don't think it actually stopped me from doing anything. But it was hard to—you had to click inside the scenario to see what was going on, and it wasn't connected to the code directly. That would have probably sped a few things up.

The type of explanation that P17r is requesting and the connection between use scenarios and code mentioned by P9r is exactly the type of support that participants who used the case library version had access to in the Script Development section.

6.4.1.3 To what extent do participants write better code, both conceptually and syntactically, when given the repository or case library form of ScriptABLE? (RQ4.1)

Participants did generate more correct code during the second session than the first session of the study, as measured by the rubric. However, there were no significant differences in the correctness of the code generated by the case library and repository groups. Further, I found no significant difference in the improvement in code quality between the two sessions for the treatment groups. Thus, the observed increase in performance was likely the result of a learning effect or some ancillary detail not attributable to the version of ScriptABLE used.

6.4.1.4 *To what extent do participants with access to the case library form of ScriptABLE produce more coherent conceptual explanations about project-oriented programming activities than those using the repository version? (RQ4.2)*

I presented confirmatory, though marginally significant, evidence that users of the ScriptABLE case library did in fact exhibit greater improvement in their conceptual answers to prompts than users of the repository version. The small sample size and lab setting used in this study limit the generalizibility of this finding, but I contend that it does demonstrate that the ScriptABLE case library can measurably lead to the appropriation of normative computing knowledge.

6.4.2 Examining the Case Library Conceptual Gains

Given the findings of RQ4.1 and RQ4.2, it is natural to wonder why the case library users showed increased gains in conceptual performance but not in performance on code measures. Let us consider two types of participants at opposite ends of the spectrum: advanced users[9] who already have a fairly solid understanding of the concepts being studied and beginners whose knowledge of JavaScript is largely the product of simple copy/paste operations of pre-existing code.

Advanced users need resources to look up unfamiliar functionality or, most often, to remind themselves about small syntactic details which they have forgotten (Brandt et al., 2009). The underlying concepts have already been learned. For these users, either version of ScriptABLE provides enough information through the tag pages and example code to help them with such tasks as remembering how to write a function header in JavaScript—they are simply seeking small hints about specific syntax elements. Thus, these users are able to create code that works just as well given either ScriptABLE version. Their conceptual answers are strong in the first session because they already know of the various constructs, and they perform equally

[9]To be clear, I intend "advanced" here to be contextualized within the participant demographics of the study, and not to refer to expert professionally trained programmers.

well in the second session.

On the other hand, beginners must attend to many more details in the two sessions. I observed that some participants who could be classified as beginners in the study had never heard of many of the concepts (e.g., `while` loops, exception handling, recursion). Lacking a strong understanding of the relevant concepts in the first session, these participants often provided non-specific answers for the strategize subtasks. They often wrote little or no code to solve the problems in the first session, and later remarked that they had trouble figuring out how to get the specific JavaScript code right. In the second session, their improvement on these factors was influenced by the version of ScriptABLE they received. Lacking much instructional content, repository users improved little on their understanding of the concepts, and were still unable to get started with selecting the necessary JavaScript syntax in the code. However, those who used the case library were often able to identify the correct concept for a given task based on the explanations given in ScriptABLE's project text. The Script Development narrative helped them confirm that the concept they were reading about could help solve their problem, and they used this in their responses to the strategize prompts. This resulted in pronounced improvements in their conceptual performance scores. However, when they were required to put their strategy into code, they still struggled to transfer knowledge about the concept to write completely new code in their project—none of the code examples presented in ScriptABLE could be directly copy/pasted to solve a subtask. In unsuccessfully transferring between the concrete details of the ScriptABLE project and the assigned task, their code performance scores did not improve by the same margin. If this is in fact the case, one might imagine that case library users would eventually demonstrate code performance gains given use the system beyond the time constraints in this particular study.

6.5 Limitations and Alternate Study Designs

As with any research effort, there are threats to validity that limit the results of this study. Recruitment proved to be considerably more difficult than originally anticipated, and as a result, the study was necessarily smaller. Ideally, I would have had significantly more participants even for the study design ultimately used. It is encouraging that, given these circumstances, I was able to detect some interesting and meaningful differences in the data. The remainder of this section enumerates a number of possible alternative study designs that could inform future research efforts.

1. **Add a True Control Group** The two condition design I used suffers from a potential confound in that learning gains between sessions one and two are to be expected (i.e., a rehearsal effect), but not entirely separable in the final data set. As mentioned in the results section, participants in both conditions improved significantly on both coding and conceptual responses, but it cannot be determined how much of that was simply the result of a learning effect or differences in the difficulty of the assigned projects. Adding a control group, who again had access to the Internet in session two, would have allowed for greater precision in the comparisons. Lacking this, I have been careful in this chapter to only draw my conclusions about learning from the between-subjects results.

2. **Counterbalance Assigned Projects Across Sessions** Another option that could provide greater power to observe the effect of ScriptABLE versions on perfomance is to counterbalance the assigned projects across the two sessions. In this study all participants received the same project in session one and then the same second project in session two. As I noted earlier, ancillary differences in the projects could have made one of them more difficult than the other. The

collection of search terms is suggestive that participants may have had considerably more trouble with task 1 from session 1 than the corresponding task in session 2—the helper function named in the session 1 task description appears many times in the list of queries. Counterbalancing would have reduced the effect of such differences on the average performance metrics; however, to conduct such a study would have required considerably more participants to achieve useful levels of statistical power. In this study, I consciously chose not to counterbalance due to expected small sample size, and I assumed that all participants were equally affected by differences in project difficulty. Therefore, I have only used analysis of the comparisons that are free of this potential confound in my argument in support of the thesis statement.

3. **Utilize a Third-Party Interviewer for Conducting Sessions** Conducting interviews without biasing the results is a highly skilled task (Seidman, 1991). Participants can be easily influenced by their perceptions of the interviewer and what they believe the interviewer would like them to say. Similarly, user behavior and feedback in software usability studies may be affected by the presence or absence of the system designer. Having a third party conduct the study sessions and exit interviews with participants would have helped ensure that this affect was not present. While I personally conducted the study, I did take measures to limit the extent to which participants knew I was responsible for building ScriptABLE in order to mitigate this effect. Introduction of ScriptABLE was done entirely in the third person, using "the author" rather than "I" to refer to the creator of ScriptABLE's content. Additionally, I also provided instruction and reminders about the Object Model Viewer's interface at the outset of the sessions so as to not unfairly favor use of one resource over the other.

4. **Couple Results with Pre/Post Assessments** Ideally, I would have been able to conduct pre/post assessments of participants' conceptual understanding of introductory computing concepts that were not tied to their performance on the tasks. This would provide an external measure of participants' knowledge. However, at the time of this study, no such valid assessment in computing existed. The recent development of the Foundational CS1 Assessment by Tew (2010) is promising in enabling such comparisons in future work, but even it lacks coverage of some of the concepts studied here (e.g., exception handling, importing code). Even in the ideal setting, adding another component to the study design would require an increased time commitment from participants in an already lengthy study, which is a serious consideration for recruitment.

5. **Allow Access to Personal Resources** Lastly, performance on the tasks may have been influenced by the lack of access to familiar personal resources. Some participants mentioned that they would have liked to have had access to their books or personal development environment (i.e., their desk, their computer, etc.) while completing the tasks. Obviously, working in an unfamiliar setting may have impacted how they behaved and limits the degree to which I can rely on session one performance as a true measure of how they would have behaved in this situation normally. That said, only 16.7% (3/18) of participants mentioned this after session two, and 44.4% (8/18) indicated they had no additional resources they would have liked to use.[10] Given these observations it seems reasonable to say that their behaviors in this study were faithful reproductions of their tendencies.

[10]The remaining comments related to specific ScriptABLE feature suggestions or access to one-on-one tutoring.

6.6 Chapter 6 Summary

In this chapter, I presented the design and results of an evaluation study for ScriptABLE. Eighteen participants with some prior knowledge of programmatic manipulation of pictures and other media completed the two part study. Data collected was used to investigate resource usage patterns, user satisfaction, and ScriptABLE's ability to promote task-oriented learning of normative computing concepts. I provided evidence that use of ScriptABLE is markedly different than use of the Internet at large to solve similar problems, and I demonstrated that the case library form of ScriptABLE can lead to measurable improvement in the responses to questions about programming concepts among end-user programmers. Lastly, I discussed a number of potential threats to the validity of this work and provided rationale to support the interpretations I presented in this chapter, despite the non-ideal circumstances of the study design.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

With this dissertation I have investigated the challenges that end-user programmers face from a computer science education perspective. I have engaged in a cycle of learner-centered design to answer the high-level questions: What do users know; what might they need to know; how are they learning; and how might we help users discover and learn what they need or want to know? I have used this unique lens to frame EUPer challenges as issues related to knowledge and understanding about computer science. Rather than building new languages or programming tools, I have addressed these difficulties through new types of instructional materials and opportunities for felicitous engagement with them. Recall the thesis statement I introduced in Chapter 1:

> A case-based learning aid for graphic and web design end-user programmers can leverage current user practices of project and example-driven learning, promote the use and browsing of instructional content, and thereby foster the appropriation of knowledge about normative programming concepts.

To affirm this thesis statement I have conducted and presented the results of three unique studies, as well as the design of a new case-based learning aid named ScriptABLE. The first two studies, presented in Chapters 3 and 4, provided the necessary context about graphic and web design EUPers to guide the design of ScriptABLE's conceptual content and the nature of its presentation. The last study (Chapter 6) demonstrated ScriptABLE as a proof-of-concept, verifying that it can indeed foster conceptual knowledge gains for users actively engaged in project-oriented programming activities.

In this conclusion chapter, I will revisit and extend some of the results presented throughout the dissertation. I first examine the findings by providing answers to the original four research questions that I posed in Chapter 1. I then turn to a discussion of the specific contributions made by this dissertation work, followed by a series of recommendations for designers of educational resources that have similar goals to those of ScriptABLE. I bring the chapter to a close by elaborating on possible future research directions that stem from the work presented here.

7.1 Answering the Primary Research Questions

The results presented in the chapters of this document provided concrete answers to operationalized research questions for each of the four large questions put forward in the introduction. I will now review those results and, in so doing, answer those top-level research questions.

7.1.1 What is the nature of graphic/web design end-user programmers' knowledge of normative computing concepts? (RQ1)

The study I presented in Chapter 3 spoke to RQ1. In particular, the open and closed card sorting activities I carried out with graphic and web designers enabled me to explore how much they knew about a fixed set of common terms taken from introductory computer science. Graphic and web designers from the study had some experience writing scripts and/or programs in various languages. Many of them recognized the vast majority of the computing terms that I presented to them, however, often they required the help of a definition and example of the term. Simply put, formal terminology was a potential roadblock for these designers. Additionally, I identified a set of six introductory concepts that were consistently ranked as not well understood, difficult to learn, and infrequently used: exporting code, indefinite loop, variable scope, recursion, functional decomposition, and exception handling. Finally, results from this study suggested that graphic and web designers associate these introductory concepts differently than typical students of computer science. Rather than grouping concepts by syntactic or semantic concerns, participants focused more on pragmatic or concrete relationships between the concepts and how they are used in daily practice.

Generally speaking, this evidence suggests end-user programmers (or, at least those in graphic and web design fields) are unique in their relationship to normative computing knowledge. There is little intrinsic value for them in knowing about programming for programming's sake; their knowledge of concepts is tied closely to the tasks they need to complete in order to get paid. They are able to identify concepts if shown concrete syntax (e.g., JavaScript keywords), but they are not overly concerned with knowing the abstract terminology for concepts. The difficulty with this disposition is that is it foreseeable that such users will produce sub-optimal solutions by relying exclusively on familiar constructs. Additionally, lacking common and consistent vocabulary, they may encounter difficulty in communicating with colleagues or in locating relevant information when needed.

7.1.2 What learning practices do graphic/web design end-user programmers currently employ, and to what extent do typical resources provide opportunities to learn about normative computing concepts? (RQ2)

Results from Chapters 3 and 4 shed light on this research question. Qualitative interview comments from graphic and web designers indicated that learning about programming is often driven by specific project demands. Learning for these end-user programmers was often reported as a process of trial and error. In a way, their process resembled bricolage programming discussed by Turkle and Papert (1991). Content was not identified and learned before writing code; rather, participants described that information seeking was regularly interleaved with writing code, testing, and examining the outcome. Finally, while graphic and web designers overwhelmingly preferred online codes examples and documentation, I found a primary repository for Photoshop scripts lacking in content for the previously identified difficult and misunderstood concepts.

The important takeaway from these findings is that end-user programmers who rely on online, community-driven websites for information may find themselves at a loss when what they need to learn something new. This is particularly pronounced when their information need is outside of the typical, frequently used scripting constructs. Without some form of external support for new conceptual content, repositories like these may become self-reinforcing libraries of a small number of already well understood concepts.

7.1.3 How does the presentation of conceptual information as a case library influence the way end users interact with resources? (RQ3)

The ScriptABLE evaluation study presented in Chapter 6 outlined my results pertaining to RQ3. Participants utilized ScriptABLE while solving problems considerably more than when they had access to the Internet to complete similar tasks. Further, I found that study participants were significantly more likely to engage in browsing behaviors in the ScriptABLE case library, versus their almost exclusively search-driven information seeking strategies employed when they were allowed access to the Web at large. Perhaps this was due to the restricted size of ScriptABLE, but nonetheless its structure and content seemed to have an impact on users' behaviors. Interestingly, participants did not exhibit markedly divergent opinions about the value of the different versions of the ScriptABLE system. However, those who had access to the full version with commentary about example projects may have favored ScriptABLE slightly more than those who lacked the commentary.

The increase in ScriptABLE use compared to general Internet use taken together with qualitative feedback from participants indicates that I was successful in building a case-based learning aid that users perceived as useful. These results support my argument that case-based materials are a good fit for the practices of end-user programmers. Namely, they leverage project-driven learning while contextualizing conceptual content and presenting it in a just-in-time manner. One participant went so far as to remark:

> P3c: It was more zoned in or written in a way that I can understand it a little bit better. It wasn't written by a programmer for programmers who already knew how to program.

7.1.4 To what extent does ScriptABLE as a case-based learning aid enable the appropriation of computing knowledge for users actively engaged in project-oriented programming activities? (RQ4)

In the same study presented in Chapter 6, I detailed results covering RQ4. I used differences in measures of code quality and open-ended conceptual responses to assess the degree to which participants appropriated content knowledge in ScriptABLE. While I noted that participants produced significantly better code when using either of two versions of ScriptABLE, I was not able to attribute those differences to specific features. More directly, the addition of narrative explanations about example projects did not significantly impact code quality. However, I did observe marginally significant differences in the sophistication of participant answers to free form questions. Participants who had access to ScriptABLE's narrative commentary improved more in their ability to produce coherent conceptual explanations for questions about a project not contained in the system. That is, they appeared to learn more computing content than those who did not see the commentary.

These results confirm that use of ScriptABLE in its full case library form can have a measurable impact on learning of computing concepts. As the first learner-centered attempt at supporting end-user programmers, these findings are encouraging. I have shown that it is possible to design and implement an educational resource informed by existing end-user programmer practices that can promote learning of normative programming concepts. Further, I observed these learning gains for the concepts which similar users previously noted were most difficult, least well understood, and least frequently used.

7.2 Contributions

In answering the four questions this dissertation makes several tangible contributions to the research community. Specifically:

1. **The first detailed characterization of graphic and web design end-user programmers.** Much of the research on end-user programming has concentrated on users in disciplines that are closely allied with STEM fields (science, technology, engineering, and mathematics). For example, accountants (Beckwith et al., 2006; Wilson et al., 2003), computational scientists (Segal, 2007), and architects (Gantt & Nardi, 1992). There has also been considerable attention for tools to support those engaged in various forms of knowledge work (Scaffidi, Myers, & Shaw, 2008; Leshed et al., 2008). However, to date little attention has been given to EUPers who use programming for artistic and creative endeavors. This dissertation work is an in-depth exploration of this largely unexplored space. Further, in approaching EUPer challenges from a learner-centered stance, I have provided a novel and detailed depiction of what graphic and web designers know about normative computing concepts and the issues they encounter while learning.

2. **An analysis of the existing information space of graphic and web design EUPers.** People make use of a vast number of resources when learning something new about programming. In order to adequately scaffold graphic and web design EUPers I had to understand the information ecology in which

they live. This dissertation not only describes what resources EUPers use, but it also examines one high profile online code repository with a critical eye. This content analysis is unique in that it explicitly views the repository's content as an educational support, highlighting the deficiencies a novice programmer would face in using the site to learn.

3. **A prototype case-based learning aid with the design and implementation of ScriptABLE.** The design of ScriptABLE and the processes I used can serve as a model for those who intend to build instructional resources for informal learners of computing, especially for end-user programmers. From its tagging and indexing system to the content of its project pages, I drew on research evidence about what resources graphic and web design EUPers use and how they engage with them. It is is an example of how to present instruction about computing while leveraging the existing information seeking habits of EUPers.

4. **Initial confirmatory evidence supporting case-based learning aids for informal computing education among EUPers.** The evaluation study I conducted as part of this dissertation confirms the proof of concept embodied by ScriptABLE. I was able to demonstrate that EUPers engaged in project-oriented tasks do come to rely on the ScriptABLE case library for their inquiries. They also responded favorably to the system, which is indicative of a good fit with their current practices. Lastly, and most importantly, I confirmed that the narrative commentary about script development, which was unique to the ScriptABLE case library, corresponded to measurably improved performance on open-ended conceptual prompts. These findings provide justification for the future study of case-based learning aids designed for EUPers.

5. **An argument in support of the value of normative computing knowledge**

among EUPers. One might make the case that end-user programmers have no use for normative computing knowledge—that the purely situated knowledge they acquire along the way is sufficient. In this dissertation I have rejected such notions and have argued in favor of formal conceptual knowledge. My results point to clear reasons why EUPers would benefit from a more sophisticated relationship with computing. A formal vocabulary could enhance their ability to communicate with others and locate relevant resources, and their productivity might improve through greater attention to code modularization and designing for reuse.

7.3 Recommendations and Open Questions for Resource Designers

In this section I distill high-level take-aways for resource designers based on my observations in the ScriptABLE evaluation study. I also raise some open questions about my design decisions that could have important implications, but were beyond the scope of this dissertation work. I should note that the comments I make here are my informed impressions; however, they are, at the very least, tied to anecdotal evidence drawn from exit survey data about what participants liked most and least about ScriptABLE.

7.3.1 Provide Connections to Normative Computing Content

Drawing directly from the findings and contributions of the dissertation work outlined in this chapter, I argue that resource designers should make a more concerted effort to connect to normative computing content in their materials, where appropriate. ScriptABLE takes a direct route in providing such connections by interleaving discussion of computing concepts within narratives about example projects. I was able to demonstrate that doing so has a positive measurable impact on participants' conceptual explanations. Additionally, those users who did not need the full explanations provided by ScriptABLE were easily able to ignore them and still make use of the components of the system that they needed. That is, the interleaved style did not appear to negatively impact more knowledgeable users.

I contend that attempts at connecting to computing content that are afterthoughts or merely side notes in a resource (as might be argued was the case for the repository version of ScriptABLE) cause users to lose the benefits of context in the learning environment. Certainly, there may be equally valid ways to enact this principle beyond the narrative structure used in ScriptABLE. However, the manner in which such connections are made to normative computing content should be explicit and in the foreground of activity for resource users.

7.3.2 Consistency in Organization and Writing is Important

As I discussed in Chapter 5, one of the things that distinguishes ScriptABLE from other tutorial sites is consistency in the structure of project articles and their explicit educational intent. Several participants responded positively to this, noting it was the characteristic they liked most. I believe this was key in contributing to the increased sense of comfort they felt using ScriptABLE compared to the Internet at large. The consistent format allows them to quickly understand how articles are written and predict what content is present and where it might be found. Resource designers should take this into consideration and incorporate scaffolding for article or example authoring tools, especially where content is contributed by third parties.

7.3.3 Larger Project Collections

The study I described in Chapter 6 only relied on seven projects in ScriptABLE. This collection of projects was quite limited, but intentionally scoped to cover only the content needed to complete the tasks in the evaluation study. The small size allowed most users to exhaustively explore the case library, but it also detracted from

ScriptABLE's appeal. Some participants commented that they felt the content was too limited to be of practical use outside of the study, but noted that they would be interested in using a later version of the system. Providing reasonably complete coverage of introductory programming concepts and JavaScript syntax seems necessary to generate sufficient interest in order to sustain real-world use of a resource like ScriptABLE.

7.3.4 Index Use is Directly Proportional to Library Size

While this may seem somewhat obvious, some participants remarked that the large number of ScriptABLE pages devoted to tagging and indexing was overkill given the aforementioned small number of project pages. As a result, these index pages were underutilized by some people who, instead, just reviewed each of the seven projects. For hierarchical indexing systems like the one I implemented in ScriptABLE to have self-evident value to users, the content being indexed needs to be sufficiently large. I conjecture that a library size on the order of 20–30 projects would begin to show much more positive opinions of tagging and indexing.

7.3.5 Revisions to the Tagging System are Needed

I relied on MediaWiki's existing category system to implement the tagging and indexing in ScriptABLE. While project pages distinguished between primary tags and other tags (see Chapter 5), the tag pages themselves did not separate index entries based on this distinction. For example, when a user views the tag page for selection statements, he or she is shown five project pages that incorporate the concept. However, only two of these pages have explicit instructional content about that topic. In the study sessions, I observed participants most often visiting the pages listed first in the cross index list, rather than finding the pages most comprehensively covering the term. A clear place for improvement in ScriptABLE's design (and that of other resources) is to demarcate the primary/secondary tag distinction on the index page.

7.3.6 Examine Wikis as a CBLA Platform

On a related note, I chose to use MediaWiki as the platform for ScriptABLE primarily for its ease of content development and potential for future expansion to community-driven authoring. However, I believe this decision had some unintended consequences. Most notably, the ScriptABLE search feature used the standard features of MediaWiki. Unfortunately, this proved somewhat inadequate for searching over the mixture of plain text and source code contained in ScriptABLE projects. While not highly used, search was the most commonly mentioned feature by participants when asked what they liked least about the system. Resource developers need to consider alternative means of specifying search parameters over narrative style explanations that incorporate programming code. Further, the extent to which wiki-style interfaces are appropriate in this problem space remains an open question.

7.3.7 Reconsider Intended Resource Use Patterns

While developing the project structure and content for ScriptABLE, I envisioned that participants would try out at least some of the scripts and use scenarios described in the case library. Strikingly, no participant ever downloaded or ran the code for a project (though some people did copy and paste portions of code). As I discussed in Chapter 5, traditional case-based learning aids often require a learner to actively work through a case, piecing together information from various components of the system or drilling down through content. Given the usage patterns I observed, it seems unlikely that end-user programmers like those in Chapter 6 would engage in such learning behaviors in a natural setting. ScriptABLE's design proved robust in this environment, as it allowed users to still make sense of the project content without having to run the program first hand. Additional empirical evidence is needed to further guide resource designers about interaction patterns that are considered natural or non-obstructive to end-user programmers.

7.3.8 Measure the Effect of Project Length

Some participants commented on the fact that they believed ScriptABLE's project pages were too long. Certainly, project page length is a valid concern for both resource designers and users. I made a concerted effort to constrain the length of pages by presenting no more than three versions of a script in each project. Still, some script development sections became considerably larger than others. Discovering the maximally effective balance between the inclusion of instructional content and project "digestibility" (to borrow a metric from Chapter 3) is an important question for future work.

7.4 Future Directions

This final section briefly explores three possible avenues for future work. While I believe that there are many opportunities for novel learner-centered studies in end-user programming (and HCI more broadly), I intentionally limit my discussion here to the future work that I see as most relevant to ScriptABLE and case-based learning aids.

7.4.1 Moving Out of the Lab

As I mentioned in Chapter 6, the evaluation study for ScriptABLE necessitated a balance of pragmatic concerns with the preservation of ecological validity. The laboratory setting I ultimately chose enabled me to gather the data needed for the study, but it leaves many obvious questions unanswered. For example: Would EUPers actually read the content of a case library under the pressure of day to day time constraints? How much additional content (and what kind of content) is needed to make ScriptABLE useful for web and graphic designers outside of the contrived tasks assigned in this study? At what point does ScriptABLE cease to be useful for a EUPer? A real world deployment study would allow me to investigate these questions.

Additionally, an in-situ study would permit more longitudinal investigations. Of particular interest is how information seeking behaviors change for EUPers. I discussed in Chapter 3 that participants indicated they collectively relied on over a dozen different resources while trying to learn. I also commented on notable interaction patterns with ScriptABLE and Adobe's Object Model Viewer tool in Chapter 6. Given these observations, I believe studying how users interleave references to resources over time is promising. Even studying a limited collection of digital resources like ScriptABLE, Google searches, and use of the Object Model Viewer could yield unique patterns for different types of end-user programmers which could further inform the design of new instructional materials and programming environments.

7.4.2 ScriptABLE as a User-Generated Site

One of the challenges in building ScriptABLE was the large per project time cost. The limited set of seven projects in the system used for the evaluation study took me several weeks to build. While some of this time was spent devising an appropriate set of examples that could cover the necessary concepts, considerable effort still went into developing appropriate use scenarios, creating several versions of each script, and generating the script development narrative. Thus growing the content of ScriptABLE, an essential task if it is to be used more widely, is a daunting task.

The success of online communities like Wikipedia,[1] Stack Overflow,[2] and others point to the power of user-generated content as a possible solution to this problem. Several years ago I debated whether to build an online community for my research and discarded the idea as intractable in the short term. The challenges of building and sustaining an audience for the community would certainly still exist and are well beyond the scope of what I have presented here. However, I believe my dissertation results may provide some guidance moving forward in structuring user contributions.

[1]`http://www.wikipedia.org`

[2]`http://stackoverflow.com`

As discussed in Chapter 6, I did not observe significant differences of opinion between users of the repository and case library versions of ScriptABLE. They all generally had moderately positive views about the system. A hybrid approach for content creation could mitigate the time commitment for project development, quickly increase the size of the case library, and at the same time preserve overall visitor opinion about the site. In essence a contributor could start a stub entry for a project with as little information as what was given in the repository treatment. Then other, perhaps more advanced, contributors could collaboratively edit the document to craft the script development narrative and highlight relevant conceptual content. Building a dedicated group of such curators (or gardners in Nardi's (1993) terms) could potentially ensure that unfamiliar conceptual content continues to be introduced into the case library.

7.4.3 Extension to Other End-User Programming Domains

In this dissertation I have explored graphic and web design end-user programmers from a variety of angles. This has resulted in detailed characterizations of the population and a novel educational resource designed explicitly with their needs and habits in mind. As I mentioned in the introduction, the number of end-user programmers is conservatively estimated to be roughly four times that of professional programmers (Scaffidi et al., 2005). However, EUPers cannot be necessarily be considered en masse because they are made up of distinct user groups, each with their own set of domain-specific tools and practices. Thus, the extent to which the results presented here generalize to other end-user programmer groups is yet to be established. There is considerable reason to be hopeful, however. Some of ScriptABLE's underlying design guidelines are based on observations that have been made in other EUP contexts as well. Most notably is the prominent role of examples and example-centric development habits (e.g., Rosson et al., 2005; Brandt, Dontcheva, Weskamp, & Klemmer, 2010). This repeated theme may suggest that case-based approaches may generally prove powerful among diverse end-user programmers and others engaged in informal learning about computer science.

APPENDIX A

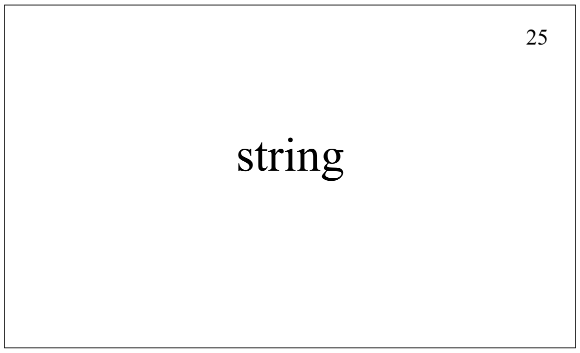
INDEX CARDS FROM CARD SORTING STUDY

This appendix contains copies of all cards used in the study described in Chapter 3. Each of these cards was printed on a 3x5 notecard. The first 26 cards contain a single programming concept in the middle of the card and a number in the upper right hand corner that was randomly assigned. The numbers were used to ease recording of participant answers and have no other significance. The next set of 26 cards were used in the event that a participant did not recognize a term. These cards contain the term and the same number (with the letter A as a suffix), but they also have a brief definition and a JavaScript example in the center of the card.

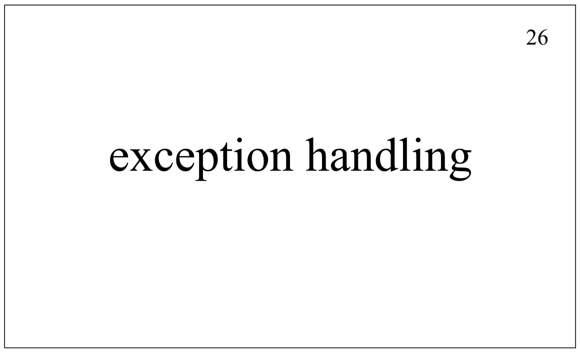
1 recursion	2 array
3 output	4 relational operator
5 importing code	6 type conversion
7 logical operator	8 selection statement

9 function	10 variable
11 number	12 functional decomposition
13 boolean	14 variable scope
15 parameters	16 indefinite loop

17 mathematical operator	18 object
19 nesting control structures	20 exporting code
21 assignment	22 input
23 definite loop	24 constant


25

string


26

exception handling

recursion 1A

definition: a function or definition that refers to itself, either directly or indirectly

```
function foo()
{
     …
    foo();
}
```

array 2A

definition: a collection of similar objects that can be accessed through indexing

```
var myArray = new Array(3);
myArray[0] = "value 1";
myArray[1] = "value 2";
myArray[2] = "value 3";
```

output 3A

definition: data printed in a human readable form or written to a file for future use

```
alert("The value is: " + x);
```

relational operator 4A

definition: a comparison between values that returns true or false

```
<

>

<=

>=

==
```

importing code 5A

definition: makes code from an external library module available for use within a program

```
import library.identifer;


#include filename
```

type conversion 6A

definition: an operation that converts a value of one data type to another

```
stringFour = 4.toString();



numFour = parseInt("four");
```

logical operator 7A

definition: an operator that can be applied to boolean values

```
&&


||


!
```

selection statement 8A

definition: a control structure that allows different parts of a program to execute depending on the exact situation

```
if (condition)
{
    …
}
else
{
    …
}
```

function 9A

definition: a subprogram within a program

```
function funcName()
{
    …
}
```

variable 10A

definition: an identifier that labels a value for future reference

```
var x;
```

number 11A

definition: a data type for representing a numeric value

```
4


3.485
```

functional decomposition 12A

definition: the process of building a system by starting with a very high-level alogirthm that describes a solution in terms of subprograms

```
function stepOne() { … }
function stepTwo() { … }
function stepThree() { … }

stepOne();
stepTwo();
stepThree();
```

boolean 13A

definition: a data type for representing a logical truth value

```
true


false
```

variable scope 14A

definition: the area of a program where a given variable may be referenced

```
function foo()
{
    var x = 10;
}

alert("x = " + x);

// causes error "x is undefined"
```

parameters 15A

definition: special variables in a function that are initialized at the time of call with information passed from the caller

```
function foo(parameter1, parameter2)
{
    …
}
```

indefinite loop 16A

definition: a loop for which the number of iterations required is not necessarily known at the time the loop begins to execute

```
while (condition)
{
    …
}
```

mathematical operator 17A

definition: an operator that can be applied to number values

```
+

-

*

/

%
```

object 18A

definition: a program entity that has some data and a set of operations to manipluate that data

```
function Person(name, age) {
    this.name = name;
    this.age = age;
    this.displayName =
         function() { alert(this.name); }
};

myPerson = new Person("Bob", 40);
myPerson.displayName();
```

nesting control structures 19A

definition: the process of placing one control structure inside of another. Loops and decisions may be arbitrarily nested.

```
if (condition1)
{
    if (condition2)
    {
        …
    }
}
```

exporting code 20A

definition: an external collection of useful functions or classes that can be imported and used in a program

```
#engine name


export identifier, id2
```

assignment 21A

definition: the process of giving a value to a variable

```
x = 500;
```

input 22A

definition: data retrieved from a user or read from an external file

```
userName = prompt("Please enter your name");


response = confirm("Is your name " +
                      userName + "?");
```

definite loop 23A

definition: a kind of loop where the number of iterations is known at the time the loop begins executing

```
for (initialize; condition; increment)
{
    …
}
```

constant 24A

definition: an identifier that labels a fixed value for future reference

```
const x = 500;
```

string 25A

definition: a data type for representing a sequence of characters (text)

```
"this is a string"


'so is this'
```

exception handling 26A

definition: a programming language mechanism that allows the programmer to gracefully deal with errors that the language detects when a program is running

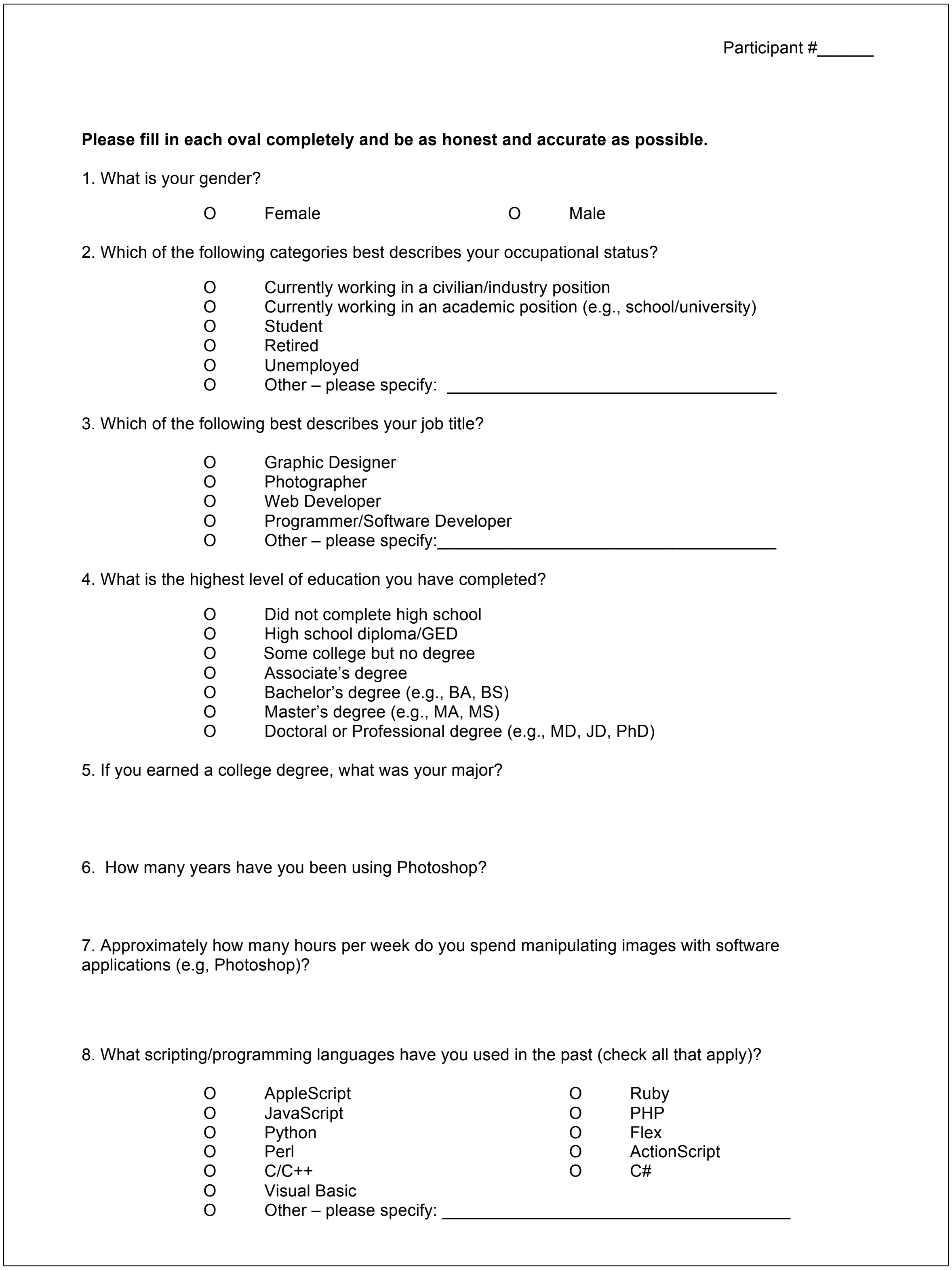
```
try
{
   …
}
catch(exception)
{
   …
}
```

APPENDIX B

STUDY 1 SURVEY AND INTERVIEW GUIDE

This appendix contains the survey document and interview guide questions used in the study described in Chapter 3. These materials were used for the portions of the study not pertaining to the card sorting activity.

B.1 Background Survey



Participant #______

Please fill in each oval completely and be as honest and accurate as possible.

1. What is your gender?

O	Female	O	Male

2. Which of the following categories best describes your occupational status?

O Currently working in a civilian/industry position
O Currently working in an academic position (e.g., school/university)
O Student
O Retired
O Unemployed
O Other – please specify: ____________________________________

3. Which of the following best describes your job title?

O Graphic Designer
O Photographer
O Web Developer
O Programmer/Software Developer
O Other – please specify:____________________________________

4. What is the highest level of education you have completed?

O Did not complete high school
O High school diploma/GED
O Some college but no degree
O Associate's degree
O Bachelor's degree (e.g., BA, BS)
O Master's degree (e.g., MA, MS)
O Doctoral or Professional degree (e.g., MD, JD, PhD)

5. If you earned a college degree, what was your major?

6. How many years have you been using Photoshop?

7. Approximately how many hours per week do you spend manipulating images with software applications (e.g, Photoshop)?

8. What scripting/programming languages have you used in the past (check all that apply)?

O	AppleScript	O	Ruby
O	JavaScript	O	PHP
O	Python	O	Flex
O	Perl	O	ActionScript
O	C/C++	O	C#
O	Visual Basic		
O	Other – please specify: ______________________________________		

Participant #______

9. On a scale from 1 (novice) to 5 (expert), how would you rate your scripting/programming expertise?

10. How many years have you been scripting and/or programming?

11. Approximately how many hours per week do you spend developing scripts and/or programs?

12. Have you had formal or professional training (e.g., classes, degrees, certificates) in programming? If so, briefly describe your training in the space provided.

O Yes O No

13. If you were attempting a new task with a piece of software rate how likely would you be to consult the following resources for assistance on a scale from 1 (very unlikely) to 5 (very likely)?

	Very Unlikely 1	**Unlikely 2**	**Neither Likely nor Unlikely 3**	**Likely 4**	**Very likely 5**
An interactive wizard that takes you step-by-step through the task	O	O	O	O	O
Examples of similar tasks from which you can borrow ideas and/or copy code	O	O	O	O	O
A class or seminar	O	O	O	O	O
Reference books	O	O	O	O	O
FAQs, tutorials or online documentation	O	O	O	O	O
A friend or coworker	O	O	O	O	O
Technical support/telephone hotline	O	O	O	O	O

B.2 Interview Questions

- What are some typical projects you use Photoshop for?
- What are your most common uses for scripting?
- Can you tell me about a particular project that you completed recently that you were really excited about?
- Could you tell me a little about how you most often go about learning a new technique or skill in Photoshop and/or scripting?
- Could you tell me a little about how you go about writing your scripts?
- What would you like to learn about next with regard to Photoshop?
- Is there a particular technical detail about scripting you would like to know more about? If so, what?
- Is there a particular project you're interested in pursuing next? If so, tell me about it. (Participants were prompted for both scripting and Photoshop answers to this question.)
- Is there anything else you'd like to share?

APPENDIX C

STUDY 3 SESSION 1 PROBLEM CODE

This appendix contains code from the three source files given to participants in the first study session described in Chapter 6. To complete the assigned tasks, participants only needed to make edits to "Extract Meta Data.jsx". The other two files, "CSVWriter.jsxinc" and "utils.jsxinc" were included as library files.

C.1 Extract Meta Data.jsx

```
#target photoshop

//Purpose:  This script will save meta-data about photos in an
//          easily searchable spreadsheet (CSV) file.

#include CSVWriter.jsxinc
#include utils.jsxinc

////////////////// Functions Go Here ///////////////////////



/////////////// Main Program Below ////////////////////////

//Prompt user for the CSV file.  A user can specify a
//filename that doesn't exist yet too.
fileName = File.openDialog("Select␣CSV␣database␣file");
csvfile = new CSVWriter(fileName);
csvfile.writeHeader(["File","Author","Auth␣Pos","Title","Created", \
                     "Camera","ColorMode","Keywords"]);

//Prompt user for the folder of images to process
alert("Select␣input␣folder␣containing␣images");
imageFolder = Folder.selectDialog();
files = imageFolder.getFiles("*.*");

//Process each file in the input folder
for (var i = 0; i < files.length; i++)
{
  //Open picture
  app.open(files[i]);                //app is an Application object
  imgDocument = app.activeDocument;//imgDocument is a Document object
  info = imgDocument.info;           //info is a DocumentInfo object
  //Build an array of the values we'd like to save
  //in the proper order
  var values = new Array();

  values.push(imgDocument.name);     //File Name
  values.push(info.author);          //Author Name
  values.push(info.authorPosition); //Author Position
  values.push(info.title);           //Picture Title

  //Extract the creation time from camera data
  for (var j = 0; j < info.exif.length; j++)
  {
     if (info.exif[j][0] == "Date␣Time␣Original")
     {
        values.push(info.exif[j][1]);
     }
  }

  //Extract the camera's model information
  for (var j = 0; j < info.exif.length; j++)
  {
     if (info.exif[j][0] == "Model")
     {
        values.push(info.exif[j][1]);
     }
  }

  //Detect color mode and write out correct value (incomplete)
  colorMode = "Color";
  values.push(colorMode);

  values.push(info.keywords);   //Picture Keywords

  //Write out all the values for this image
  csvfile.writeRow(values);

  //All done with this image
  imgDocument.close();
}

//All done, clean up now
csvfile.closeFile();
```

C.2 csvWriter.jsxinc

```
function CSVWriter(filename) {
   if (filename == null)
   {
      throw "No␣filename␣specified!";
   }
```

```
this.fileStream = new File(filename);

if (this.fileStream.exists == false)
{
   this.fileStream.open('w');
   this.numberFields = 0;
}
else
{
   this.fileStream.open('e');
   var fileLength = this.fileStream.length;
   var fields = this.fileStream.readln();
   this.numberFields = (fields.split(",")).length;
   this.fileStream.seek(fileLength);
}

//========================================
//Methods of the CSVWriter object below
//========================================

this.writeHeader = function(fieldArray)
   {
      if (this.numberFields == 0)
      {
         this.numberFields = fieldArray.length;
         this.writeRow(fieldArray);
      }
      else if (this.numberFields != fieldArray.length)
      {
         throw "Mismatch␣in␣number␣of␣columns!";
      }
   }

this.writeRow = function(valueArray)
   {
      if (valueArray.length != this.numberFields)
         throw "Unexpected␣number␣of␣values!";

      for (var i = 0; i < this.numberFields; i ++)
      {
         //Write out value surrounded by quotes
         this.fileStream.write("\"" + valueArray[i] + "\"");

         //Write the comma delimiter for all but the last column
         if (i < this.numberFields -1)
            this.fileStream.write(",");
      }
      this.fileStream.write("\n");
   }

this.closeFile = function()
   {
      //Since we're done writing the file, we should
```

```
            //close the stream
            this.fileStream.close();
        }
}
```

C.3 utils.jsxinc

```
function isColorImage(imgDocument)
{
    return imgDocument.mode == DocumentMode.RGB;
}

function isBWImage(imgDocument)
{
    return imgDocument.mode == DocumentMode.GRAYSCALE;
}
```

APPENDIX D

STUDY 3 SESSION 2 PROBLEM CODE

This appendix contains code from the three source files given to participants in the second study session described in Chapter 6. To complete the assigned tasks, participants only needed to make edits to "CreateGallery.jsx". As with the first sessions, the other two files, "HTMLWriter.jsxinc" and "utils.jsxinc" were included as library files.

D.1 *CreateGallery.jsx*

```
#target photoshop

//  Purpose: This script will automatically generate a
//           basic web gallery of pictures for clients

#include "HTMLWriter.jsxinc"
#include "utils.jsxinc"

///////////  Functions go here  /////////////////////



////////////  Main program starts below  /////////////

//Some initial settings
const thumbnailSize = 75;
const previewSize = 480;
const galleryName = "Photo␣Proofs";
const photographerName = "George␣Burdell";

currentDate = new Date();    //currentDate is a Date object
copyright = "2009";

//Prompt user for input and output folders
alert("Select␣input␣folder␣containing␣images");
imageFolder = Folder.selectDialog();

alert("Select␣output␣folder␣where␣gallery␣will␣be␣created");
outputFolder = Folder.selectDialog();

// Build directory structure in output location for the web gallery
```

```
outFolder = new Folder(outputFolder);
testFolder(outFolder);

outputImageDir = new Folder(outputFolder + "/images");
outputImageDir.create();

outputThumbDir = new Folder(outputFolder + "/thumbs");
outputThumbDir.create();

outputPagesDir = new Folder(outputFolder + "/pages");
outputPagesDir.create();

//Retrieve the list of files in input folder
folder = new Folder(imageFolder);
testFolder(folder);
files = folder.getFiles("*.*");

//Create menu bar frame HTML file
menuPage = new HTMLWriter(outputFolder + "/MenuFrame.html");
menuPage.writeHeader("Gallery␣Menu");

//Process each file in the input folder
for (i = 0; i < files.length; i++)
{
  //Open picture
  app.open(files[i]);              //app is an Application object
  imgDocument = app.documents[0]; //imgDocument is a Document object

  //Detect if the image is portrait or landscape (incomplete)
  portraitFlag = false;

  //Build the thumbnail from a copy of original
  thumbDocument = imgDocument.duplicate();
  if (portraitFlag)
    thumbDocument.resizeImage( \
      (imgDocument.width/imgDocument.height) * thumbnailSize);
  else
    thumbDocument.resizeImage(thumbnailSize);
  thumbnailFile = new File(outputThumbDir.absoluteURI + "/thumb-" +
                           files[i].name);
  thumbDocument.saveAs(thumbnailFile, JPEGSaveOptions);
  thumbDocument.close();

  ////////////////////////////////////////////////////////////////
  //  Changes made after this point will only affect the
  //  fullsize and preview images (not the thumbnail)
  ////////////////////////////////////////////////////////////////

  if (portraitFlag)
    imgDocument.rotateCanvas(90);

  //Add in a watermark so clients can't just print the full
  //resolution images themselves
  watermarkLayer = imgDocument.artLayers.add();
  watermarkLayer.kind = LayerKind.TEXT;
  watermarkLayer.opacity = 50;
  watermarkLayer.textItem.size = 300;
  ColorWhite = new SolidColor();
  ColorWhite.rgb.hexValue = "FFFFFF";
  watermarkLayer.textItem.color = ColorWhite;
  watermarkLayer.textItem.justification = Justification.CENTER;
  watermarkLayer.textItem.position = Array(imgDocument.width/2,
                                    imgDocument.height / 2 + 200);
  watermarkLayer.textItem.contents = "PROOF";
  imgDocument.flatten();

  if (portraitFlag)
    imgDocument.rotateCanvas(-90);

  //Save out full size copy
  copyToFile = new File(outputImageDir.absoluteURI + "/" +
                        files[i].name);
  imgDocument.saveAs(copyToFile, JPEGSaveOptions);

  //Save out smaller preview copy
  if (portraitFlag)
    imgDocument.resizeImage( \
      (imgDocument.width/imgDocument.height) * previewSize);
  else
    imgDocument.resizeImage(previewSize);
  previewFile = new File(outputImageDir.absoluteURI + "/small-" +
                         files[i].name);
  imgDocument.saveAs(previewFile, JPEGSaveOptions);

  //All done with this image
  imgDocument.close(SaveOptions.DONOTSAVECHANGES);

  //Build the HTML file for the preview page for the image
  framePage = new HTMLWriter(outputPagesDir.absoluteURI + "/" +
                             files[i].name + ".html");
  framePage.writeHeader(copyToFile.name);
  framePage.writeText("<TABLE␣CELLSPACING=0␣CELLPADDING=0␣BORDER=0"+
    "␣WIDTH=100\%␣HEIGHT=100\%><TR><TD␣ALIGN=CENTER>");
  framePage.writeImage("../images/" + previewFile.name)
  framePage.writeText("<BR>");
  framePage.writeLink("../images/" + copyToFile.name);
  framePage.writeText("</TD></TR></TABLE>");
  framePage.writeFooter();

  //Add thumbnail and link to menu page
  menuPage.writeLinkedImage("thumbs/" + thumbnailFile.name, \
            "pages/" + files[i].name + ".html", "previewPane");
  menuPage.writeText(files[i].name);
  menuPage.writeText("<HR>");

}

//Finish closing the menu HTML file
```

```
menuPage.writeFooter();

createIndexPage(outputFolder, galleryName, photographerName, \
                copyright);
```

D.2 HTMLWriter.jsxinc

```
function HTMLWriter(filename)
{
  this.fileStream = new File(filename);
  //This will return false if it can't open this
  this.fileStream.open('w');

  //Specify the methods for the HTMLWriter object
  this.writeHeader = function(pageTitle)
    {
      this.writeText("<HTML><HEAD><TITLE>" + pageTitle +
                     "</TITLE></HEAD>");
      this.writeText("<BODY>");
    }

  this.writeFooter = function()
    {
      this.writeText("</BODY></HTML>");

      //Since we're done writing the file, we should close
      //the stream
      this.fileStream.close();
    }

  this.writeText = function(outputText)
    {
      this.fileStream.writeln(outputText);
    }

  this.writeImage = function(filepath)
    {
      this.writeText("<IMG␣SRC=\"" + filepath + "\">");
    }

  this.writeLink = function(url)
    {
      this.writeText("<A␣HREF=\"" + url + "\">" + url + "</A>");
    }

  this.writeLinkedImage = function(image, url, target)
    {
      this.writeText("<A␣TARGET=" + target + "␣HREF=\"" +
        url + "\"><IMG␣SRC=\"" + image + "\"></A>");
    }
}
```

D.3 utils.jsxinc

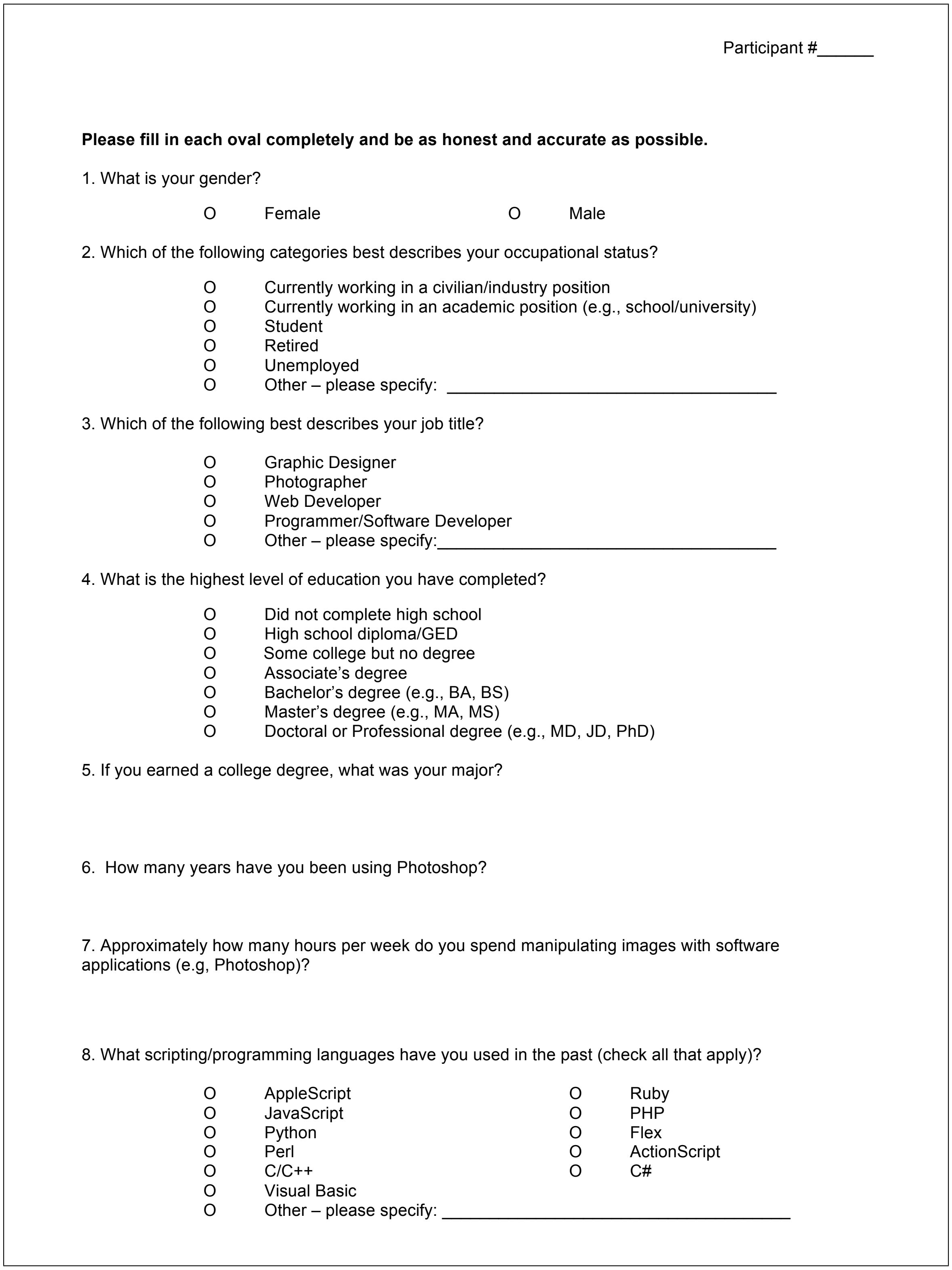
```
function createIndexPage(outputFolder, galleryName, \
                         photographerName, copyright)
{
  //Create index.html now
  indexPage = new HTMLWriter(outputFolder + "/index.html");
  indexPage.writeHeader("");
  indexPage.writeText("<TABLE␣BGCOLOR=#000000␣CELLSPACING=0␣" +
    "CELLPADDING=0␣BORDER=0␣WIDTH=100\%␣HEIGHT=100\%>" +
    "<TR><TD␣ALIGN=CENTER>")
  indexPage.writeText("<TABLE␣BGCOLOR=#FFFFFF␣BORDER=0␣" +
    "WIDTH=800␣HEIGHT=450><TR>");
  indexPage.writeText("<TD␣BGCOLOR=#AAAAAA␣ALIGN=CENTER␣WIDTH=225>"+
    "<B>" + galleryName + "<BR>" +
    "By:␣" + photographerName + "<BR>" +
    "(c)" + copyright + "</B></TD>");

  indexPage.writeText("<TD␣WIDTH=100\%␣HEIGHT=100\%␣ROWSPAN=2>␣" +
    "<IFRAME␣BORDER=0␣WIDTH=100\%␣HEIGHT=100\%␣NAME=previewPane␣" +
    "SRC=\"pages/" + files[0].name + ".html\">" +
    "Your␣browser␣does␣not␣support␣frames." +
    "</IFRAME></TD></TR>");

  indexPage.writeText("<TR><TD␣HEIGHT=100\%><IFRAME␣HEIGHT=100\%␣" +
    "BORDER=0␣WIDTH=225␣SRC=\"MenuFrame.html\">" +
    "Your␣browser␣does␣not␣support␣frames." +
    "</IFRAME></TD></TR></TABLE>");
  indexPage.writeText("</TD></TR></TABLE>");
  indexPage.writeFooter();
}

function testFolder(f)
{
  if (! f.exists)
    throw "Folder␣does␣not␣exist!";
}

JPEGSaveOptions.quality = 12;
preferences.rulerUnits = Units.PIXELS;
```

APPENDIX E

STUDY 3 INSTRUMENTS

This appendix contains all of the materials and instruments used in conducting the ScriptABLE evaluation study described in Chapter 6. The included materials are the demographic background survey, participant instructions for session 1, interview questions following session 1, participant instructions for session 2, a post session survey that followed session 2, and the final interview questions asked upon completion of the study.

E.1 Background Survey



Participant #______

Please fill in each oval completely and be as honest and accurate as possible.

1. What is your gender?

O Female O Male

2. Which of the following categories best describes your occupational status?

O Currently working in a civilian/industry position
O Currently working in an academic position (e.g., school/university)
O Student
O Retired
O Unemployed
O Other – please specify: ____________________________________

3. Which of the following best describes your job title?

O Graphic Designer
O Photographer
O Web Developer
O Programmer/Software Developer
O Other – please specify:____________________________________

4. What is the highest level of education you have completed?

O Did not complete high school
O High school diploma/GED
O Some college but no degree
O Associate's degree
O Bachelor's degree (e.g., BA, BS)
O Master's degree (e.g., MA, MS)
O Doctoral or Professional degree (e.g., MD, JD, PhD)

5. If you earned a college degree, what was your major?

6. How many years have you been using Photoshop?

7. Approximately how many hours per week do you spend manipulating images with software applications (e.g, Photoshop)?

8. What scripting/programming languages have you used in the past (check all that apply)?

O	AppleScript	O	Ruby
O	JavaScript	O	PHP
O	Python	O	Flex
O	Perl	O	ActionScript
O	C/C++	O	C#
O	Visual Basic		
O	Other – please specify: ______________________________________		

Participant #______

Please fill in each oval completely and be as honest and accurate as possible.

1. What is your gender?

O Female O Male

2. Which of the following categories best describes your occupational status?

O Currently working in a civilian/industry position
O Currently working in an academic position (e.g., school/university)
O Student
O Retired
O Unemployed
O Other – please specify: ______________________________

3. Which of the following best describes your job title?

O Graphic Designer
O Photographer
O Web Developer
O Programmer/Software Developer
O Other – please specify:______________________________

4. What is the highest level of education you have completed?

O Did not complete high school
O High school diploma/GED
O Some college but no degree
O Associate's degree
O Bachelor's degree (e.g., BA, BS)
O Master's degree (e.g., MA, MS)
O Doctoral or Professional degree (e.g., MD, JD, PhD)

5. If you earned a college degree, what was your major?

6. How many years have you been using Photoshop?

7. Approximately how many hours per week do you spend manipulating images with software applications (e.g, Photoshop)?

8. What scripting/programming languages have you used in the past (check all that apply)?

O AppleScript	O Ruby
O JavaScript	O PHP
O Python	O Flex
O Perl	O ActionScript
O C/C++	O C#
O Visual Basic	
O Other – please specify: ______________________________	

E.2 Session 1 Instructions and Tasks

Participant: _______

Project: Personal Database of Photo Data

Description

The goal of this project is to complete a script that can be used to save important pieces of information about photos in a collection to a personal database in the form of a CSV (comma separated variable) file. This CSV file can then be quickly searched to locate a specific photo within the collection, saving you time and energy when you need to retrieve a picture. This program should be able to continue adding new information to an existing CSV file so that the user can add additional photos to the database over time.

Instructions

In the time allotted, complete each of the following tasks in order.

- Some tasks will ask you to write out an answer about what you think is happening in the program. Please write your answers in the space provided.
- It is okay if you don't have enough time to finish all of the tasks, so don't worry about spending too much time on one task.
- However, if you feel you are really stuck on a task let me know.

You are free to look up any information on the Internet using Firefox. Also, you can make use of any of the help resources provided in the ExtendScript Toolkit to help you complete these tasks.

When you begin each task, please indicate to me that you are doing so.

Warmup Tasks

A. Before you continue, look over the files and code provided to you, try running the program using the file *myDatabase.csv* and the folder *imageSet1*. Look at the result by opening *myDatabase.csv* in Excel to get a feel for what the program does.

B. Edit the code for the open dialog used for `fileName` so that only files with the extension ".csv" are shown.

Participant: _______

Individual Tasks

1. **Part A:** Run the program using *imageSet2* as the input folder. Open the CSV file that you selected and view the result of the script. Compare the images in the folder to the values in the ColorMode column. In the space below describe what happened for the black and white images in the folder.

   Part B: Edit/insert the necessary lines of code to appropriately set the variable `colorMode` to "Color" or "B/W" depending on the picture document's color mode. To help you with this task we have provided two functions described below:

`isColorImage(imgDocument)`	is true when the document has RGB color mode and false otherwise
`isBWImage(imgDocument)`	is true when the document is in Grayscale mode and false otherwise

2. **Part A:** Currently the script has two areas of largely duplicated code; namely, extracting creation time and camera model information. Edit the script so that this code appears only once in a function called `getPropertyValue`. As parameters, this function should take the `info` variable and a string value that indicates which property should be retrieved. The output of the program should not change as a result of this step.

   Part B: Briefly explain why you might want to create a function such as `getPropertyValue` in your design of a script.

3. **Part A:** If you run the program and click the cancel button when prompted for the CSV file or input directory, you'll notice the program encounters an error. Devise a strategy so that when the user clicks cancel on either dialog, they are prompted again for these values. Your strategy should only require changes to Extract Meta Data.jsx. In the space below, briefly explain your strategy.

   Part B: Make the necessary changes to Extract Meta Data.jsx to implement your strategy.

4. **Part A:** Consider the line of code near the bottom of the program (approximately line 63) that starts with `csvfile.writeRow.` Where is `writeRow` defined and why you are able to call it here?

   Part B: Briefly justify why you might choose to design parts of your script like this.

Participant: _______

5. **Part A:** Run the program using *imageSet3* as the input folder. You'll notice that the script doesn't complete correctly. Using the CSV file results, the *imageSet3* folder of pictures, and any information in the ExtendScript Toolkit, determine the image that caused the program to crash. In the space below describe what you think happened.

   Part B: What programming technique could you use to prevent the program from crashing in this situation?

   Part C: Edit the script as necessary so that it runs correctly in this scenario. When a bad file is encountered you should display a message using the code below and then ignore the file.

   ```
   alert("Skipping a bad file");
   ```

6. **Part A:** Sometimes photos are organized in sub-folders to make them easier to find. Our script should be able catalog information about all photos in a collection, regardless of how it is organized. To test this scenario, run the program using *imageSet4* as the input folder. Did the script perform as intended? If not, describe what happened.

   Part B: What combination of programming strategies and/or techniques could you use to make the script behave as intended in this situation?

Accompanying these instructions were a collection of files for participants to use during session 1. The three code files used are shown in their entirety in Appendix C. An initial CSV database file was provided that contained only one row of header information. In addition, participants were provided with 4 input directories described as follows:

imageSet1 is a folder of 10 JPEG images, all in RGB color mode.

imageSet2 is a folder of 10 JPEG images, three of which are in a grayscale color mode with the rest in full RGB color.

imageSet3 is a folder of 10 JPEG images, two of which have been corrupted by inserting text at random into the image file

imageSet4 is a folder containing 2 JPEG images and two subfolders, each with 4 more JPEG images inside

E.3 Session 1 Post-Task Interview Questions

- Of the tasks you completed, which do you think was the most difficult?
- What did you struggle with?
- Did you learn anything today about Photoshop? Did you learn anything today about scripting? If so, what?
- What resources did you find most useful in solving these problems?
- On a scale of 1 to 10, how confident are you that the code you've written is correct?
- Is there anything you wish you would have had to help you solve these problems?

E.4 Session 2 Instructions and Tasks

Participant: _______

Project: Web Gallery of Proof Photos

Description

The goal of this project is to complete a script that can be used to automatically create a simple web gallery of proof photos for a client. The idea is that given a folder of pictures, the script should quickly resize the pictures, generate thumbnails, apply watermarks, and generate HTML necessary for a basic gallery. The resulting output folder will contain everything that needs to be uploaded to a server in order to give a client access to the gallery.

Instructions

In the time allotted, complete each of the following tasks in order.

- Some tasks will ask you to write out an answer about what you think is happening in the program. Please write your answers in the space provided.
- It is okay if you don't have enough time to finish all six, so don't worry about spending too much time on one task.
- However, if you feel you are really stuck on a task let me know.

In this session your use of the Internet is limited to a particular website, but you are free to use it as much as you'd like. You may also use help resources provided in the ExtendScript Toolkit interface.

When you begin each task, please indicate to me that you are doing so.

Warmup Tasks

A. Before you continue, look over the code provided to you, try running the program with *imageSet1* as the input folder, and look at the output folder to get a feel for what the program does.

B. Edit the code so that the copyright year stored in the variable copyright is retrieved from the computer's date, rather than hard coded to 2009.

Participant: _______

Individual Tasks

1. **Part A:** Run the program using *imageSet2* as the input folder. Open the gallery in your web browser to view the result of the script. Compare the relative sizes of the thumbnails/previews created for portrait and landscape pictures. In the space below describe what happens for images that have portrait orientation.

   Part B: Edit/Insert the necessary lines of code to set the variable `portraitFlag` to true or false depending on the picture's orientation. Hint: you can access the picture document's height and width as shown below:

   ```
   imgDocument.height

   imgDocument.width
   ```

2. **Part A:** There is a section of code in this script that handles the creation of the "PROOF" watermark on the picture. Edit the script so that all of the watermark code is contained in a function called `applyWatermark.` As parameters this function should take the picture document (`imgDocument`) and a string value with the watermark text (e.g., "PROOF"). The output of the program should not change as a result of this step.

   Part B: Briefly explain why you might want to create a function such as `applyWatermark` in your design of a script.

3. **Part A:** If you run the program and click the cancel button when prompted for the input or destination folder, you'll notice the program encounters an error. Devise a strategy so that when the user clicks cancel on either dialog, they are prompted again for these values. Your strategy should only require changes to CreateGallery.jsx In the space below, briefly explain your strategy.

   Part B: Make the necessary changes to CreateGallery.jsx to implement your strategy.

4. **Part A:** Consider the line of code near the bottom of the program (approximately line 108) that starts with `framePage.writeHeader.` Where is `writeHeader` defined and why are you able to call it here?

   Part B: Briefly justify why you might choose to design parts of your script like this.

Participant: _______

5. **Part A:** Run the program using *imageSet3* as the input folder. You'll notice that the script doesn't complete correctly. Using the resulting web gallery, the *imageSet3* folder of pictures, and any information in the ExtendScript Toolkit, determine the file that caused the program to crash. In the space below, describe what you think happened.

   Part B: What programming technique could you use to prevent the program from crashing in this situation?

   Part C: Edit the script as necessary so that it runs correctly in this scenario. When a bad file is encountered, you should display a message using the code below and then ignore the file.

   ```
   alert("Skipping a bad file");
   ```

6. **Part A:** Sometimes photos are organized in sub-folders to make them easier to find. Our script should be able to create a gallery containing all the photos in a collection, regardless of how they are organized. To test this scenario, run the program using *imageSet4* as the input folder. Did the script perform as intended? If not, describe what happened.

   Part B: What combination of programming strategies and/or techniques could you use to make the script behave as intended in this situation?

Accompanying these instructions were a collection of files for participants to use during session 1. The three code files used are shown in their entirety in Appendix D. In addition, participants were provided with 4 input directories described as follows:

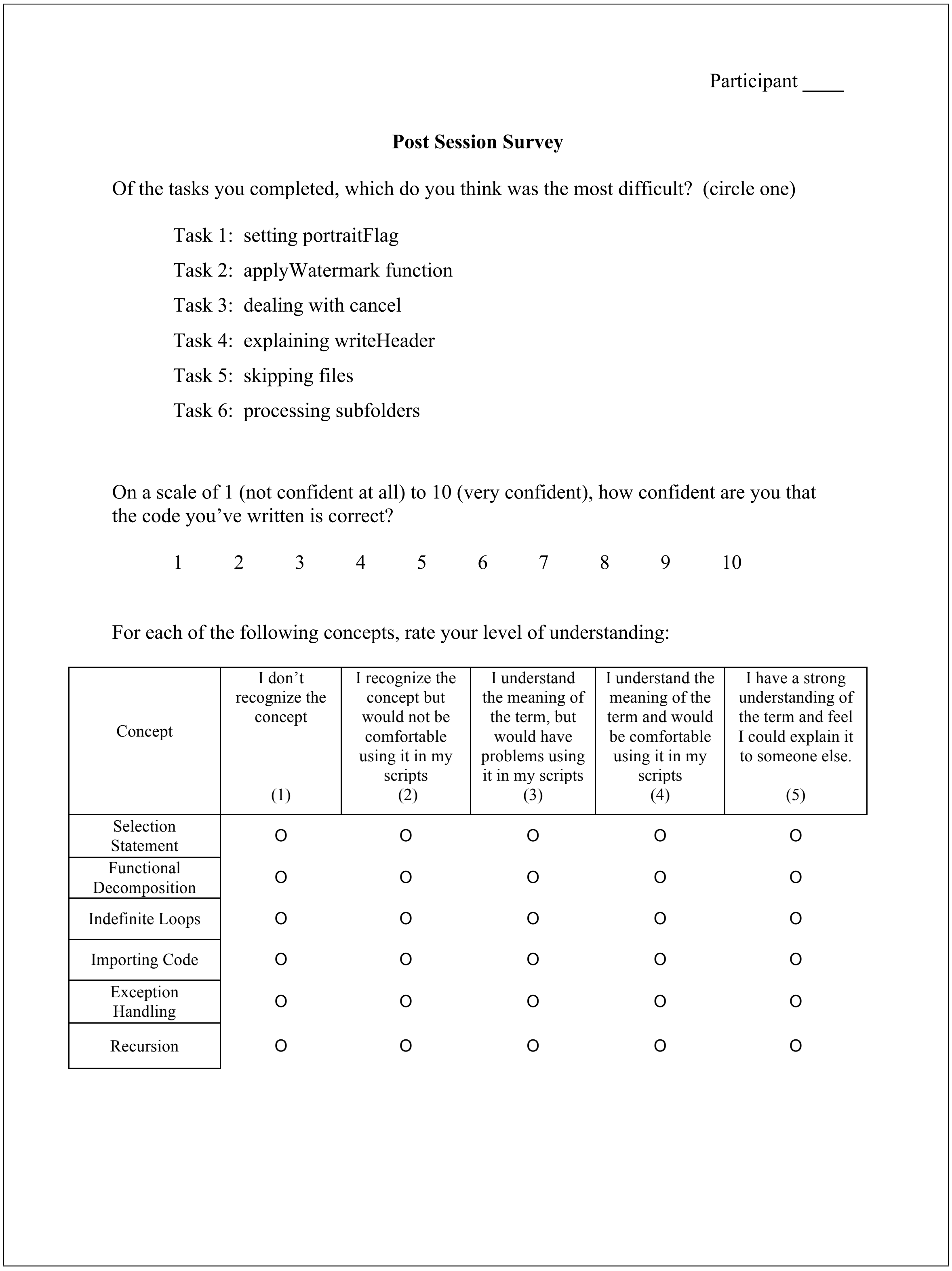
imageSet1 is a folder of 7 JPEG images, all having landscape orientation (i.e., having a greater width than height).

imageSet2 is a folder of 7 JPEG images, three of which have portrait orientation (i.e., having a greater height than width).

imageSet3 is a folder of 6 files. The first four files alphabetically are JPEG images, the fifth file is a plain text file, and the last file is a image saved in the TIFF format.

imageSet4 is a folder containing 2 JPEG images and two subfolders, each with 4 more JPEG images inside

E.5 Session 2 Completion Survey



Participant ____

Post Session Survey

Of the tasks you completed, which do you think was the most difficult? (circle one)

Task 1: setting portraitFlag

Task 2: applyWatermark function

Task 3: dealing with cancel

Task 4: explaining writeHeader

Task 5: skipping files

Task 6: processing subfolders

On a scale of 1 (not confident at all) to 10 (very confident), how confident are you that the code you've written is correct?

1 2 3 4 5 6 7 8 9 10

For each of the following concepts, rate your level of understanding:

Concept	I don't recognize the concept (1)	I recognize the concept but would not be comfortable using it in my scripts (2)	I understand the meaning of the term, but would have problems using it in my scripts (3)	I understand the meaning of the term and would be comfortable using it in my scripts (4)	I have a strong understanding of the term and feel I could explain it to someone else. (5)
Selection Statement	O	O	O	O	O
Functional Decomposition	O	O	O	O	O
Indefinite Loops	O	O	O	O	O
Importing Code	O	O	O	O	O
Exception Handling	O	O	O	O	O
Recursion	O	O	O	O	O

Participant ____

For each of the following concepts, rate your current understanding compared to what you knew before today:

Concept	Significantly Worse (1)	Worse (2)	About the same (3)	Better (4)	Significantly better (5)
Selection Statement	O	O	O	O	O
Functional Decomposition	O	O	O	O	O
Indefinite Loops	O	O	O	O	O
Importing Code	O	O	O	O	O
Exception Handling	O	O	O	O	O
Recursion	O	O	O	O	O

Rate your experience with ScriptABLE:

	Strongly disagree (1)	Disagree (2)	Neutral (3)	Agree (4)	Strongly Agree (5)
ScriptABLE helped me complete the tasks today.	O	O	O	O	O
ScriptABLE is a good resource for me to learn new things.	O	O	O	O	O
ScriptABLE is a good resource for other people to learn new things.	O	O	O	O	O
I would use ScriptABLE in my daily scripting projects.	O	O	O	O	O

What did you like best about ScriptABLE?

What did you like least about ScriptABLE?

E.6 Session 2 Post-Task Interview Questions

- What did you struggle with today?
- Did you learn anything today about Photoshop? If so, what?
- Did you learn anything today about scripting? If so, what?
- What resources did you find most useful in solving these problems?
- Is there anything you wish you would have had to help you solve these problems?
- Compare and contrast your experiences in the two sessions.
- (follow up on any survey questions)

REFERENCES

Adobe Photoshop CS2 JavaScript scripting reference. (2005). San Jose, CA. (Retrieved October 24, 2006, from `http://partners.adobe.com/public/developer/en/photoshop/sdk/JavaScriptReferenceGuide.pdf`)

Anderson, L. W., et al. (Eds.). (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives* (abridged ed.). New York, NY: Longman.

Beckwith, L., Kissinger, C., Burnett, M., Wiedenbeck, S., Lawrance, J., Blackwell, A., et al. (2006). Tinkering and gender in end-user programmers' debugging. In *CHI '06: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 231–240).

Beringer, J. (2004). Reducing expertise tension. *Communications of the ACM*, *47*(9), 39–40.

Bhat, G. P., & Kolodner, J. L. (2009). A case-based system to aid cognition and meta-cognition in a design-based learning environment. In *Association for the advancement of artificial intelligence fall symposium* (pp. 26–31).

Blackwell, A. F. (2002). First steps in programming: a rationale for attention investment models. In *Proceedings of the 2002 IEEE symposium on human centric computing languages and environments* (pp. 2–10).

Bloom, B. S., Englehart, M. D., Furst, E. J., Hill, W. H., & Krathwohl, D. R. (Eds.). (1956). *The taxonomy of educational objectives: The classification of educational goals; Handbook I: Cognitive domain.* New York, NY: David McKay Company, Inc.

Brandt, J., Dontcheva, M., Weskamp, M., & Klemmer, S. R. (2010). Example-centric programming: Integrating web search into the development environment. In *CHI '10: Proceedings of the 28th international conference on human factors in computing systems* (pp. 513–522).

Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., & Klemmer, S. R. (2009). Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *CHI '09: Proceedings of 27th international conference on human factors in computing systems* (pp. 1589–1598).

Bransford, J. D., Brown, A. L., & Cocking, R. R. (2000). *How people learn: Brain, mind, experience, and school* (Expanded ed.). Washington, D.C.: National Academy Press.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Pyschology*, *3*(2), 77–101.

Bruckman, A., & Edwards, E. (1999). Should we leverage natural-language knowledge? An analysis of user errors in a natural-language-style programming language. In *CHI '99: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 207–214).

Burnett, M., Atwood, J., Djang, R. W., Gottfried, H., Reichwein, J., & Yang, S. (2001). Forms/3: a first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of Functional Programming*, *11*(2), 155–206.

Carroll, J. M. (1990). *The Nurnberg funnel: Designing minimalist instruction for practical computer skill.* Cambridge, MA: MIT Press.

Carroll, J. M. (Ed.). (1998). *Minimalism beyond the Nurnberg funnel.* Cambridge, MA: MIT Press.

Carroll, J. M., & Rosson, M. B. (2005). Cases as minimalist information. In *Proceedings of the 38th Hawaii international conference on system sciences.*

Caspersen, M. E., & Bennedsen, J. (2007). Instructional design of a programming course—a learning theoretic approach. In *ICER '07: Proceedings of the 2007 international computing education research workshop* (pp. 111–122).

Clancy, M. J., & Linn, M. (1995). *Designing Pascal solutions: A case study approach.* New York, NY: W.H. Freeman & Co.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Pyschological Measurement*, *20*(1), 37–46.

Computational media. (2008). Georgia Institute of Technology. (Available at `http://lcc.gatech.edu/compumedia/`, Accessed June 22, 2010)

Cypher, A. (Ed.). (1993). *Watch what I do: Programming by demonstration.* Cambridge, MA: MIT Press.

Deitel, H., & Deitel, P. (2005). *C++: How to program* (5th ed.). Upper Saddle River, NJ: Prentice Hall.

Dorn, B., & Guzdial, M. (2006). Graphic designers who program as informal computer science learners. In *ICER '06: Proceedings of the 2nd International Workshop on Computing Education Research* (pp. 127–134).

Dorn, B., & Guzdial, M. (2010). Learning on the job: Characterizing the programming knowledge and learning strategies of web designers. In *CHI '10: Proceedings of the 28th international conference on human factors in computing* *systems* (pp. 703–712).

Dorn, B., Tew, A. E., & Guzdial, M. (2007). Introductory computing construct use in an end-user programming community. In *VL/HCC'07: Proceedings of the 2007 IEEE symposium on visual languages and human-centric computing* (pp. 27–30).

Du Boulay, B. (1989). Some difficulties of learning to program. In E. Soloway & J. Spohrer (Eds.), *Studying the novice programmer* (pp. 283–299). Hillsdale, NJ: Lawrence Erlbaum Associates.

Felleisen, M., Findler, R. B., Flatt, M., & Krishnamurthi, S. (2001). *How to design programs: An introduction to programming and computing.* Cambridge, MA: MIT Press.

Felleisen, M., Findler, R. B., Flatt, M., & Krishnamurthi, S. (2004). The Teach-Scheme! project: Computing and programming for every student. *Computer Science Education*, *14*(1), 55–77.

Fincher, S., & Tenenberg, J. (2005). Making sense of card sorting data. *Expert Systems*, *22*(3), 89–93.

Fleury, A. (1997). Code reuse through the eyes of students and professionals. In *Proceedings of the national educational computing conference* (pp. 136–142).

Gantt, M., & Nardi, B. A. (1992). Gardeners and gurus: Patterns of cooperation among cad users. In *CHI '92: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 107–117).

Gay, L. R., & Airasian, P. W. (2000). *Educational research: Competencies for analysis and application* (6th ed.). Upper Saddle River, N.J.: Merrill.

Ginat, D. (2004). On novice loop boundaries and range conceptions. *Computer Science Education*, *14*(3), 165–181.

Goel, A. K., Kolodner, J. L., Pearce, M., Billington, R., & Zimring, C. (1991). Towards a case-based tool for aiding conceptual design problem solving. In *Proceedings of the case-based reasoning workshop* (pp. 109–120). Washington, D.C.

Green, T. R. G., & Payne, S. J. (1984). Organization and learnability in computer languages. *International Journal of Man-Machine Studies*, *21*, 7–18.

Greeno, J. G., Collins, A. M., & Resnick, L. B. (1996). Cognition and learning. In D. C. Berliner & R. C. Calfee (Eds.), *Handbook of educational psychology* (pp. 15–46). New York, NY: Simon and Schuster Macmillan.

Guzdial, M. (1999). Supporting learners as users. *Journal of Computer Documentation, 23*(2), 3–13.

Guzdial, M., & Ericson, B. (2010a). *Introduction to computing and programming in Python, a multimedia approach* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Guzdial, M., & Ericson, B. (2010b). *Problem solving with data structures using Java: a multimedia approach.* Upper Saddle River, NJ: Pearson.

Guzdial, M., & Kehoe, C. (1998). Apprenticeship-based learning environments: A principled approach to providing software-realized scaffolding through hypermedia. *Journal of Interactive Learning Research, 9*(3/4), 289–336.

Guzdial, M., & Tew, A. E. (2006). Imagineering inauthentic legitimate peripheral participation: an instructional design approach for motivating computing education. In *ICER '06: Proceedings of the 2nd International Workshop on Computing Education Research* (pp. 51–58).

Hoadley, C. M., Linn, M. C., Mann, L. M., & Clancy, M. J. (1996). When, why, and how do novice programmers reuse code? In W. D. Gray & D. A. Boehm-Davis (Eds.), *Empirical studies of programmers: 6th workshop* (pp. 109–129). Norwood, NJ: Ablex.

Horstmann, C. (2006). *Big Java* (2nd ed.). Hoboken, NJ: John Wiley and Sons.

Hutchins, E. (1995). How a cockpit remembers its speeds. *Cognitive Science, 19*, 265–288.

Katsanos, C., Tselios, N., & Avouris, N. (2008). AutoCardSorter: Designing the information architecture of a web site using latent semantic analysis. In *CHI '08: Proceedings of the 26th international conference on human factors in computing systems* (pp. 875–878).

Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist, 41*(2).

Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems. In *VL/HCC '04: Proceedings of the 2004 IEEE symposium on visual languages and human-centric computing* (pp. 199–206).

Kolodner, J. L. (1993). *Case-based reasoning.* San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Kolodner, J. L. (1997). Educational implications of analogy. *American Psychologist*, *52*(1), 57–66.

Kolodner, J. L., Owensby, J. N., & Guzdial, M. (2004). Theory and practice of case-based learning aids. In D. H. Jonassen (Ed.), *Theoretical foundations of learning environments* (2nd ed., pp. 215–242). Mahwah, N.J.: Lawrence Erlbaum Associates.

Kulesza, T., Wong, W.-K., Stumpf, S., Perona, S., White, R., Burnett, M. M., et al. (2009). Fixing the program my computer learned: Barriers for end users, challenges for the machine. In *IUI '09: Proceedings of the 13th international conference on intelligent user interfaces* (pp. 187–196).

Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, *33*(1), 159–174.

Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation.* New York, NY: Cambridge University Press.

Leshed, G., Haber, E. M., Matthews, T., & Lau, T. (2008). Coscripter: automating & sharing how-to knowledge in the enterprise. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on human factors in computing systems* (pp. 1719–1728).

Lewandowski, G., Gutschow, A., McCartney, R., Sanders, K., & Shinners-Kennedy, D. (2005). What novice programmers don't know. In *ICER '05: Proceedings of the 1st international workshop on computing education research* (pp. 1–12).

Lewis, J., & Loftus, W. (2005). *Java software solutions (Java 5.0 version): Foundations of program design* (4th ed.). Boston, MA: Addison Wesley.

Li, I., Nichols, J., Lau, T., Drews, C., & Cypher, A. (2010). Here's what I did: Sharing and reusing web activity with ActionShot. In *CHI '10: Proceedings of the 28th international conference on human factors in computing systems* (pp. 723–732).

Lieberman, H. (Ed.). (2001). *Your wish is my command: Programming by example.* San Francisco, CA: Morgan Kaufmann.

Lieberman, H., Paternó, F., & Wulf, V. (Eds.). (2006). *End user development.* Dordrecht, The Netherlands: Springer.

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

Little, G., Lau, T. A., Cypher, A., Lin, J., Haber, E. M., & Kandogan, E. (2007). Koala: capture, share, automate, personalize business processes on the web. In *CHI '07: Proceedings of the SIGCHI conference on human factors in computing* *systems* (pp. 943–946).

Little, G., & Miller, R. C. (2006). Translating keyword commands into executable code. In *UIST '06: Proceedings of the 19th annual ACM symposium on user interface software and technology* (pp. 135–144).

McKeithen, K. B., Reitman, J. S., Rueter, H. H., & Hirtle, S. C. (1981). Knowledge organization and skill differences in computer programmers. *Cognitive Psychology, 13*, 307–325.

Nardi, B. A. (1993). *A small matter of programming: Perspectives on end user computing.* Cambridge, MA: MIT Press.

Norman, D. A. (1988). *The psychology of everyday things.* New York, NY: Basic Books.

Pane, J. F., Myers, B. A., & Miller, L. B. (2002). Using HCI techniques to design a more usable programming system. In *Proceedings of the 2002 IEEE symposia on human centric computing languages and environments* (pp. 198–206).

Pane, J. F., Ratanamahatana, C., & Myers, B. A. (2001). Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies, 54*, 237–264.

Panko, R. R. (1995, May). Finding spreadsheet errors: Most spreadsheet models have design flaws that may lead to long-term miscalculation. *Information Week*(529), 100.

Pearce, M., Goel, A. K., Kolodner, J. L., Zimring, C., Sentosa, L., & Billington, R. (1992). Case-based design support: A case study in architectural design. *IEEE Expert, 7*(5), 14–20.

Pirolli, P. (1991). Effects of examples and their explanations in a lesson on recursion: A production system analysis. *Cognition and Instruction, 8*(3), 207–259.

Pirolli, P. (2007). *Information foraging theory: Adaptive interaction with information.* New York, NY: Oxford University Press.

Rogoff, B., & Lave, J. (Eds.). (1999). *Everyday cognition.* New York, NY: toExcel.

Rosson, M. B., Ballin, J., & Rode, J. (2005). Who, what, and how: A survey of informal and professional web developers. In *VL/HCC '05: Proceedings of the 2005 IEEE symposium on visual languages and human-centric computing* (pp. 199–206).

Roth, W.-M. (2005). Mathematical inscriptions and the reflexive elaboration of understanding: An ethnography of graphing and numeracy in a fish hatchery.

Mathematical Thinking and Learning, *7*(2), 75–110.

Rugg, G., & McGeorge, P. (1997). The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems*, *14*(2), 80–93.

Rugg, G., & Petre, M. (2007). *A gentle guide to research methods.* Berkshire, UK: Open University Press.

Sanders, K., Fincher, S., Bouvier, D., Lewandowski, G., Morrison, B., Murphy, L., et al. (2005). A multi-institutional, multinational study of programming concepts using card sort data. *Expert Systems*, *22*(3), 121–128.

Saunders, A. (2009). Baroque parameters. *Architectural Design*, *79*, 132–135.

Scaffidi, C., Ko, A., Myers, B., & Shaw, M. (2006). Dimensions characterizing programming feature usage by information workers. In *VL/HCC '06: Proceedings of the 2006 IEEE symposium on visual languages and human-centric computing* (pp. 59–62).

Scaffidi, C., Myers, B., & Shaw, M. (2008). Tool support for data validation by end-user programmers. In *ICSE '08: Proceedings of the 30th international conference on software engineering* (pp. 867–870).

Scaffidi, C., Shaw, M., & Myers, B. (2005). Estimating the numbers of end users and end user programmers. In *VL/HCC '05: Proceedings of the 2005 IEEE symposium on visual languages and human-centric computing* (pp. 207–214).

Segal, J. (2007). Some problems of professional end user developers. In *VL/HCC'07: Proceedings of the 2007 IEEE symposium on visual languages and human-centric computing* (pp. 111–118).

Seidman, I. E. (1991). *Interviewing as qualitative research: A guide for researchers in education and the social sciences.* New York, NY: Teachers College Press.

Shaffer, D. W., & Resnick, M. (1999). "Thick" authenticity: New media and authentic learning. *Journal of Interactive Learning Research*, *10*(2), 195–215.

Soloway, E., Bonar, J., & Ehrlich, K. (1989). Cognitive strategies and looping constructs: An empirical study. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 191–207). Hillsdale, NJ: Lawrence Erlbaum Associates.

Soloway, E., Guzdial, M., & Hay, K. E. (1994). Learner-centered design: The challenge for HCI in the 21st century. *interactions*, *1*(2), 36–48.

Spohrer, J., & Soloway, E. (1985, November). Putting it all together is hard for novice programmers. In *Proceedings of the IEEE international conference on systems, man, and cybernetics.*

Subrahmaniyan, N., Kissinger, C., Rector, K., Inman, D., Kaplan, J., Beckwith, L., et al. (2007). Explaining debugging strategies to end-user programmers. In *VL/HCC '07: Proceedings of the 2007 IEEE symposium on visual languages and human-centric computing* (pp. 127–134).

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, *12*, 257–285.

Sweller, J., & Cooper, G. A. (1985). The use of worked examples as a substitue for problem solving in learning algebra. *Cognition and Instruction*, *2*(1), 59–89.

Tew, A. E. (2010). *Assessing fundamental introductory computing concept knowledge in a language independent manner.* Unpublished doctoral dissertation, Georgia Institute of Technology.

Tew, A. E., & Guzdial, M. (2010). Developing a validated assessment of fundamental CS1 concepts. In *SIGCSE '10: Proceedings of the 41st ACM technical symposium on computer science education* (pp. 97–101).

Tew, A. E., McCracken, W. M., & Guzdial, M. (2005). Impact of alternative introductory courses on programming concept understanding. In *ICER '05: Proceedings of the 2005 international workshop on computing education research* (pp. 25–35).

The Joint Task Force on Computing Curricula (Ed.). (2001). Computing curricula 2001. *Journal on Educational Resources in Computing*, *1*(3es), 1–240.

Titmus, C. (1989). *Lifelong education for adults: An international handbook.* Oxford, UK: Pergamon.

Turkle, S., & Papert, S. (1991). Epistemological pluralism and the revaluation of the concrete. In I. Harel & S. Papert (Eds.), *Constructionism: Research reports and essays, 1985-1990* (pp. 161–192). Norwood, N.J.: Ablex.

Wiedenbeck, S. (1988). Learning recursion as a concept and as a programming technique. In *SIGCSE '88: Proceedings of the nineteenth SIGCSE technical symposium on computer science education* (pp. 275–278).

Wiedenbeck, S. (2005). Facilitators and inhibitors of end-user development by teachers in a school environment. In *VL/HCC '05: Proceedings of the 2005 IEEE symposium on visual languages and human-centric computing* (pp. 215–222).

Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., et al. (2003). Harnessing curiosity to increase correctness in end-user programming. In *CHI '03: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 305–312).

Yardi, S., Dorn, B., Bruckman, A., & Guzdial, M. (2008). *Deconstructing Facebook: social support for avocational developers.* (Unpublished Manuscript)

Zelle, J. M. (2004). *Python programming: An introduction to computer science.* Wilsonville, OR: Franklin Beedle.

Zimring, C., Do, E., Domeshek, E., & Kolodner, J. L. (1995). Supporting case-study use in design education: A computational case-based design aid for architecture. In *Proceedings of the second congress on computing in civil engineering* (pp. 1635–1642).