

Graphic Designers Who Program as Informal Computer Science Learners

Brian Dorn and Mark Guzdial
College of Computing
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332-0280

{brian.dorn, mark.guzdial}@cc.gatech.edu

ABSTRACT

We introduce end-user programmers as a group of persons engaged in informal Computer Science education. Results of a small-scale survey for a previously unstudied population of end-users, users of graphics manipulation software, are presented. We find that graphic designers are taking part in significant programming activities, despite little to no formal training in programming. We discuss what draws them to programming, what they know about Computer Science, and where they seek help. We also consider ways in which we might further support the Computer Science learning that takes place in end-user settings.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education—*computer science education*;
D.2.6 [Software Engineering]: Programming Environments—*interactive environments, integrated environments*;
K.8.1 [Personal Computing]: Application Packages—*graphics*

General Terms

Design, Documentation

Keywords

End-user programming, end-user software engineering, graphic design, informal education

1. INTRODUCTION

This paper frames *end-user programmers* as a set of informal Computer Science learners. They learn about computing largely in an ad hoc fashion, and our study indicates that their activities might be enhanced with practices taught in formal Computer Science education (e.g., software engineering techniques). Broadly defined, end-user programmers are

individuals making use of a class of applications that incorporate features like textual scripting, high-level declarative specification, programming by example, and automation or customization via wizards [13]. Covered by this definition are languages and tools like Lisp in AutoCAD, Visual Basic for Applications in Excel, and JavaScript in Photoshop. The code written by many such end-users closely resembles coding done in traditional programming languages and is subject to many of the same challenges.

Recent estimates based on projections from the Bureau of Labor Statistics report that over 90 million Americans will use a computer at work by 2012, including 55 million users of spreadsheets and databases [18]. The number of self-described programmers is expected to exceed 13 million, while there will be fewer than 3 million professional programmers [18]. The study presented here and other sources (e.g., [16]) suggest that the actual number of people engaged in end-user programming activities will be even greater given the ubiquity of computing and increasingly available software that provides scripting and customization functionality (e.g., computer game extension languages, scripting in media manipulation tools, etc.). The vast majority of these users will have no formal coursework in Computer Science, and this lack of knowledge could prove costly given the reliance on the software programs they create. For example, Panko illustrates the pervasiveness of spreadsheet errors in the absence of systematic testing practices [15] and recounts the story of a Texas oil and gas company that lost millions of dollars due to spreadsheet errors [14]. The significant size and impact of this population warrant a closer look at how Computer Science learning takes place in these contexts and at how we might be able to develop resources in support of these learners.

1.1 Informal Learning

Before discussing the details of our study, it is important to take a general look at what is meant by the terms *informal education* and *informal learning environment*. *Informal education* can be viewed as:

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. [21, p. 547]

This is in contrast to the education that takes place in *for-*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICER'06, September 9–10, 2006, Canterbury, United Kingdom.
Copyright 2006 ACM 1-59593-494-4/06/0009 ...\$5.00.

mal learning environments, which have been characterized as those featuring a highly structured and chronologically ordered curriculum (e.g., that typically found in primary, secondary, and higher education) [21].

Computer Science education research has tended to favor the study of formal learning environments, but there are many opportunities to study informal computing education more systematically. Bruckman studied children learning to program in a text-based virtual world [2, 3] and others have assessed after-school programs like the Intel Computer Clubhouse [5]. However, these existing studies only target children—a small number of the total informal computer users. There are few studies of informal learning among professionals and other adults.

The study of informal Computer Science education is important for helping us understand life-long CS learning. The pervasiveness of computers and other computational devices has led to a techno-centric society. The global marketplace values those who can combine technical knowledge with other domain expertise [9]. Knowing how professionals discover and learn about computing can inform us how to teach our own undergraduate Computer Science majors. Even professionally trained computer scientists learn new languages, development environments, and software systems informally on the job. Research and development of supports for informal computing education are needed to foster a technically literate society and to scaffold life-long learning within the field.

1.2 End-User Programmers

We recognize that end-user programmers must be involved in informal Computer Science learning in order to create the artifacts that they do. They clearly must at least learn about language syntax and semantics. To investigate aspects of informal learning among end-user programmers we needed to choose an audience. Historically speaking, spreadsheet users probably have been studied most. Users of other software like computer-aided design tools and web development suites have also been examined. However, current software applications in a wide variety of domains now support scripting. We chose to study users of image manipulation packages to gain a sense for what kinds of programming activities take place in these new domains.

Graphic 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. Through scripting, users 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. This paper discusses results from a small-scale survey targeted toward exploring the make-up of this population, with special attention given to areas of Computer Science knowledge and how users typically seek out such knowledge.

The remainder of this paper proceeds as follows: Section 2 describes the survey methods used. Section 3 sketches a picture of our survey respondents’ background characteristics. Reasons for beginning to program are explored in Section 4, and specific Computer Science content knowledge exhibited by users is discussed in Section 5. Reported learning strategies are given in Section 6, and we conclude with a

high-level discussion of the results and possible avenues for future work.

2. METHOD

There have been a number of attempts to characterize and classify the population of end-user programmers. Recent studies have cast a broad net and have looked at very general uses of automation and scripting in a number of domains.

Rosson, Ballin, and Rode surveyed over 300 informal web developers with a wide variety of backgrounds and uses for technology [16]. Their analysis distinguished between “programmers” and “non-programmers” (based on self-report), and found that both groups of users seem to place value in systematic design, review, and testing practices. They also concluded that non-programmers need additional tool support so that they can enact these and other coding skills (e.g., designing for reuse).

Similarly, Scaffidi et al. surveyed *Information Week* readers in an attempt to classify end-users based on the type of abstractions they create in common business applications like databases, spreadsheets, and web development packages [17]. Their respondents had significant technical backgrounds; approximately two-thirds had a college major in a computing, engineering, or other math-related field. Seventy-nine percent of the participants were familiar with the concepts of variables, subroutines, conditionals, and loops, with about 35% reporting use of all of these in the past year. Further, they noted significant clusters of tool feature usage corresponding to macro features, linked structure features, and imperative features.

These two studies provide some insight about end-user programmers; however it is difficult to tease out the specific Computer Science knowledge present in the populations because of the generality of the study designs. The nature of web development and its related tools make it troublesome to define what constitutes a programming activity in the Rosson study. The business-specific audience of Scaffidi’s work limits the applicability of their results, and the number of participants with formal training in computing clouds the role that informal learning might be playing.

Our study replicated many of the questions asked in the Rosson and Scaffidi studies, but extended them toward a graphic design population. The aim was to look more closely at a specific group of end-users, who were engaged in a fairly well-defined form of programming. We also sought to learn about the Computer Science knowledge these users exhibit. The remainder of this section provides more details about the survey design and recruitment strategies used.

2.1 Survey

Using the previously mentioned studies ([16, 17]) as a starting point, we created a 39-question survey directed at users of image manipulation packages like Adobe Photoshop, Illustrator, and GIMP¹. These software programs have specific support for built-in scripting languages. For example, Photoshop CS2 can be extended using ExtendScript (a JavaScript variant) and GIMP can be programmed using either Scheme or Python. We asked questions about several different aspects of end-user programming: tool use habits, motivation for scripting, script development behav-

¹A copy of the survey is available at: <http://home.cc.gatech.edu/dorn/19/>.

iors, programming concept familiarity, and general background. The prompts consisted of selection from a pre-defined list of choices (e.g., Check all languages you’ve used while scripting.)², open-ended responses (e.g., “Describe the general process you use as you write your scripts.”), scale ratings (e.g., Rate your tendency to create sketches or diagrams prior to coding on a 5-point scale from never to always), and simple yes/no probes (e.g., “Have you personally defined and referenced variables?”). Some questions were asked conditionally based on user responses to previous questions. For example, if a respondent indicated that he/she was not familiar with the concept of subroutines, we did not ask about subroutine creation and use. The survey was implemented online using a third-party provider and required approximately 20 minutes to complete.

2.2 Recruitment

As mentioned previously, we were interested in reaching those who were already engaged in scripting for image manipulation tools. We anticipated that our typical respondent would be from the graphic design profession, but would not have formal training in Computer Science. While the survey was open to users of all skill levels, we intentionally sought out the group of end-users Nardi refers to as *local developers*, “domain experts who happen to have an intrinsic interest in computers and have more advanced knowledge of a particular program” [13, p. 104]. This was somewhat different from the more general recruitment strategies of other similar studies, but our rationale for pursuing the most advanced of end-users was two-fold: with no previous literature about this audience, we wanted to gain a sense for the upper-boundaries of applications for scripting in this domain, and we wanted to determine what Computer Science knowledge had been acquired along the way to becoming a local developer.

To reach our target audience, we posted recruitment messages in six online communities devoted to scripting and script-related activities within specific graphics programs. Three groups were tied to Adobe Photoshop; two were for users of GIMP; and one pertained to Adobe Illustrator. Most often the other messages in the forums were users requesting help with some advanced program feature. Our message invited readers to help us learn how real people use scripting, outlined the nature of our study, and addressed the requisite data privacy issues. Participation was wholly voluntary and no incentives were given to respondents. Since there was no reward for participation, it is likely that many of those who elected to respond had a high intrinsic interest in the scripting features of these applications. Rather than serving as a limitation, we felt this minor bias would actually be helpful in reaching the most advanced end-users.

2.3 Analysis

Interpreting the survey data was complicated by the nature of our study design. Participants were permitted to skip any questions for which they did not want to provide a response, resulting in some partially completed surveys. In some cases, the questions omitted were those intended to create mutually exclusive groups for comparison. Due to the small number of total responses we chose to include as many data points as possible on the more general portions

²Questions that required choice from a pre-defined list also incorporated an option to specify an “other” response.

of the analysis, rather than completely eliminating partially completed surveys. For example, questions of demographics (Section 3) and one’s desire to program (Section 4) were investigated using the maximum number of responses in order to depict this population of end-users as completely as possible. In the later sections that explore Computer Science concept knowledge, we were able to divide all responses into groups based on prior computing coursework and have done so accordingly. In general, since some questions were not answered by all respondents, we indicate sample sizes when reporting all means and percentages throughout the paper.

3. DEMOGRAPHICS

Responses on the demographic questions indicated that we were largely successful in reaching our target population. There was some variation in occupation and skill level, but the majority of the responses came from advanced computer users in a graphics related field. All of the respondents were male. 22% were 30 years old or younger ($n=18$ for all data presented in this section unless otherwise noted); 33% were in the range 31-40; and about 44% were over the age of 40. Most of the responses came from users in the United States, but about one-fifth were from European countries. The mean rating of computer use skill on a scale from one (beginner) to five (advanced) was 4.33 and the majority of users had worked with computers for 20 or more years.

Respondent occupation was gathered in a free-form input box and then categories were created to aggregate responses. Three high-level categories were used to describe occupation: those related to art and/or media, those related to programming and/or technology, and other. Typical careers in the art/media category were graphic design, photography, and web development. Those mentioning programming or software engineering were classified under the second category, and miscellaneous responses like construction sales and bookseller were classified as other. The majority of participants (56%, $n=16$) were in the art/media category. Figure 1 illustrates the professions in our sample population in more detail.

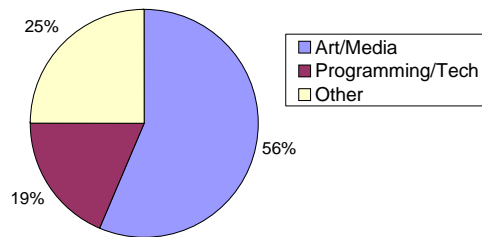


Figure 1: Area of Occupation ($n=16$)

With respect to formal education, there was relatively broad representation. As shown in Figure 2, highest education level completed spanned from high school diploma (or equivalent) to Master’s degree. Over half the responses came from those who had not earned a Bachelor’s degree (i.e., high school, college but no degree, or an Associate’s). Among those with some or more college coursework, about 70% ($n=13$) had a primary area of study in photography, art, or a media-related area. The remainder had majors in Computer Science, engineering, or another science field.

Beyond specifying college major, participants were asked

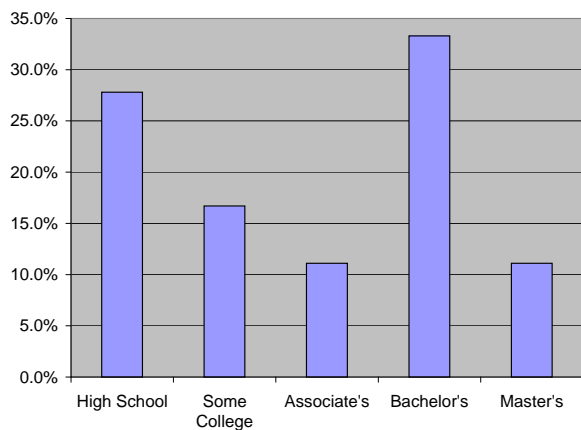


Figure 2: Highest Level of Education (n=18)

to indicate any prior formal training in programming (including classes, degrees, or certificates). These results are given in Table 1. Almost two-thirds had no experience with formal coursework in Computer Science, and when asked whether or not one self-identifies as a “programmer,” 83.3% responded negatively. It should be noted that, under this classification, formal training is broadly defined—a person having one high school programming class would answer identically to someone with an undergraduate degree in Computer Science.

Table 1: Coursework and Self-Affiliation (n=18)

Statement	Yes	No
Have you had formal training in programming (e.g., classes, degrees, certificates)?	38.9%	61.1%
Do you consider yourself a programmer?	16.7%	83.3%

To summarize, the demographic data collected indicated a close match to our intended audience. Survey respondents considered themselves to be advanced computer users, and many engaged in image manipulation activities professionally. Most had formal training in fields other than Computer Science and did not identify as programmers. However, other data gathered suggested that these users were very much engaged in what we—as Computer Scientists—consider programming. For example, one user stated that he used scripting to “build database-populated pages for print and CD catalog distribution.” It is natural to ask, what made these users turn to programming in the first place, how much programming do they really know, and how did they learn everything they needed to know? We turn to these questions in the sections that follow.

4. PROGRAMMING’S ATTRACTION

We initially imagined that the path to scripting began with use of pre-made automation features common to modern graphics packages. Scripting would then be a natural follow-on to achieve more advanced tasks that are not otherwise possible. Participants were asked whether or not they had ever made use of a number of different automation tools as part of their image editing applications. These results ap-

pear in Table 2, with features appearing roughly in order of increasing task complexity.

Table 2: Use of Automation Features (n=22)

Tool Feature	% Reporting Use
Used batch processing to apply an action to multiple files	86.4%
Created a “droplet” of an action for repeat use later	54.5%
Used a predefined automation to perform a complex task	68.2%
Executed a script written by someone else	77.3%
Edited a script or plugin written by someone else	68.2%
Personally written a script or plugin from scratch	72.7%

The first three features listed in the table warrant a brief explanation. The first item refers to use of a software wizard that aids users in creating batch automations of particular tasks to be applied across all files in a directory. Droplets, as mentioned in the second row, can be thought of a executable macros that reside on one’s desktop and accept input files through dragging-and-dropping onto the droplet’s icon. Thirdly, some image editors, like Photoshop, incorporate pre-built automations that ease complicated tasks like creating a web photo gallery or merging multiple frames of a panorama. Though there was fairly high usage of these three features, there is some need for caution. We noticed greater variation on these elements corresponding to the specific software used than the rest. It is believed that this was caused by differences in support of and nomenclature for these features in Adobe products and other packages like GIMP. Nonetheless, users did exhibit use of the highly-structured automation facilities.

Most respondents indicated that they made use of scripting either as a client or as a developer. They also provided insight into the rationale behind their initial decision to start creating scripts (see Figure 3). The most commonly mentioned motivating factor for learning to script was its ability to save time (84.6%, n=13). Enabling a custom, non-standard effect was the next greatest impetus (69.2%), followed by curiosity about programming, creation of a redistributable artifact (e.g., a plug-in), and suggestion from friends or colleagues.

After taking the initial plunge into scripting for whatever reason, respondents developed a sense for situations in which automation could be applied. Their responses, shown in Table 3, were somewhat expected based on the importance placed on saving time. Users found value in scripting to simplify repetitive tasks with iteration and to conditionally apply actions. To a lesser degree, respondents used scripts to specify tasks that controlled multiple related programs or generated dynamic media with features specified by user input (e.g., a customized greeting card).

These applications of automation clearly hinted at certain aspects of Computer Science knowledge. Users were commonly employing iteration and selection in their activities. Coordinating tasks between multiple software applications via scripts required at least a basic notion of application programmer interfaces (APIs). Another section of the survey

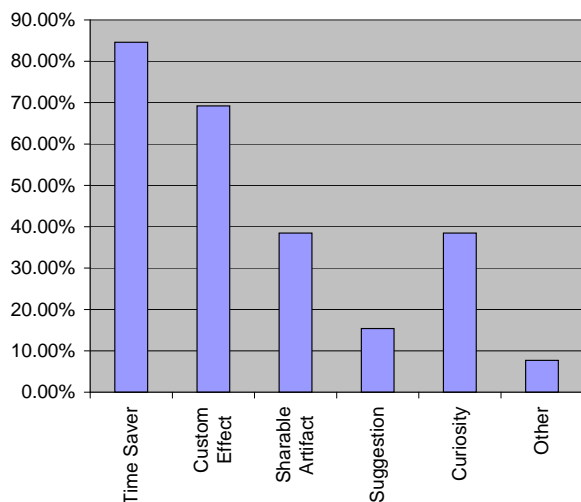


Figure 3: Why Users Start Scripting (n=13)

Table 3: Uses for Automation (n=20)

Use	% Reporting Use
Iterative application of an action within one project	85.0%
Batch processing multiple files	75.0%
Conditional application of an action	60.0%
Duplicate object creation in one project	45.0%
Control of multiple programs	40.0%
Dynamic media generation	25.0%
Other	10.0%

attempted to elicit a clearer understanding of their computing awareness.

5. COMPUTER SCIENCE KNOWLEDGE

Before delving into the specific content knowledge evidenced in the survey results, it is useful to get a better sense for what “scripting” in this context means. In general, end-user programming encompasses a broad range of activities, but the results that follow stem from automation activities that resemble traditional high-level language programming. An example script, based on actual reported usage from our survey, is given in the program listing in Figure 4. This code is written in ExtendScript, Adobe’s implementation of JavaScript, and enables a user to automatically generate 10 frames for an animation that rotates an image 360 degrees in Photoshop CS2.

Experienced programmers immediately recognize many common aspects in this 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 [1] that is not unlike documentation found in the Java API.

With this frame of reference, we sought out to investigate Computer Science knowledge among this population. One

```
//Setup units and document references
references.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 4: ExtendScript for Generating Frames

section of the survey focused specifically on elementary programming concepts, while higher-level concepts like design were explored through a series of more general questions.

5.1 Elementary Programming

Part two of our survey was devoted to a sequence of questions regarding specific programming terms. We asked about participants’ familiarity with each term, and for those topics that were known, we asked whether or not the participants had used those programmatic constructs. Each prompt contained the term, a brief definition, and a list of common synonyms. We replicated questions about the terms *variable*, *subroutine*, *conditional*, and *loop* that were originally presented in the Scaffidi study [17]. We also added a question for the term *compound data structure* to explore possible use of more advanced data types like arrays, structures, and trees. An example prompt appears below:

Are you personally familiar with the concept of a “loop”—a sequence of instructions that are repeated until a condition is met (also called “for/next”, “while/do”, “repeat/until”)? [17]

A follow-on question asked whether or not the individual had used that particular construct while scripting in the past year. We assumed that those who had some formal coursework in computing would be familiar with many of these basic terms; this was confirmed with 100% (n=7) reported familiarity for all five terms among this sub-group. However it was unclear what we could expect from those who had simply learned on their own. We were surprised to note a high rate of term recognition (over 80%) among those who had not taken a course in programming (n=11). Figure 5 depicts the term familiarity and reported-use of the “no-coursework” sub-group³.

In general, respondents with no formal training in Computer Science appeared to recognize basic programming terminology, and they also reported use of these concepts. For all terms other than compound data structures, we saw over 80% recognition. We were further surprised that a majority of people were familiar with (and used) data structures, given that it has been commented elsewhere [17] that end-users are not likely to make use of abstractions like ADTs and inheritance hierarchies. While preliminary, our results

³Use of programming constructs was only requested from those familiar with the terms. Data in this chart presumes that non-familiarity implies non-use.

suggest that these previous characterizations may have been premature.

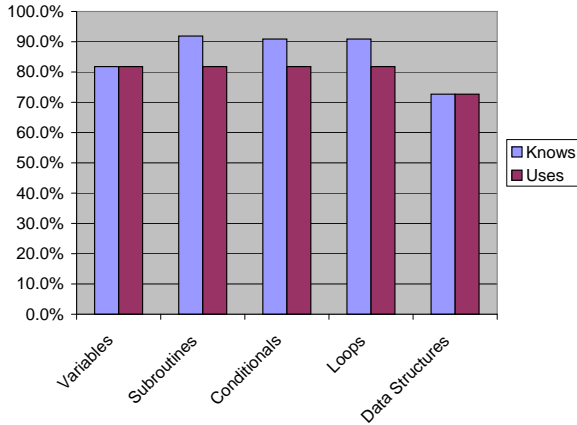


Figure 5: Programming Knowledge for those w/o Coursework (n=11)

5.2 Software Development Practices

Exploration of the elementary programming terms was encouraging, but we were also interested in more sophisticated forms of Computer Science knowledge. End-user programming researchers have acknowledged that lessons of software engineering could be valuable to informal developers. For example, Burnett and her colleagues implemented various techniques for scaffolding more formal testing behaviors among spreadsheet users [4]. We wondered what behaviors related to testing and other software development practices end-users in the graphic arts domain currently possess. In the survey, respondents were asked about their propensity to engage in particular actions with respect to their scripts. Questions pertained roughly to design, reuse, and testing behaviors. Results from a selection of these questions are shown in Table 4, separated based on prior programming coursework. Responses were given on a five-point scale where one meant never, three meant regularly, and five meant always. Small sample sizes precluded meaningful statistical analysis, but these two groups did appear to exhibit similar tendencies in many software development processes.

Design activities were the least likely of all development practices surveyed. Sketching and diagramming were used on a slightly less than regular basis, and over 60% of respondents reported never pre-planning code on paper. While these behaviors were less utilized than we had hoped, they were not entirely unexpected given the difficulty of teaching design and planning to Computer Science students (e.g., [20, 11]). Free form question responses about design revealed use of a number of strategies ranging from top-down planning to writing pseudocode first to ad-hoc development. Two of the more common responses were reuse of an existing script as a template and what might best be called experiment-and-record. Some users noted that they started from a basic goal for their script and proceeded to work out the sub-steps manually using the application. Along the way, they would note the actions that they were performing with the tool, which later became the outline for their script. Work by DiGiano noted similar interaction habits and used this

Table 4: Reported Software Development Practices

How often do you:		Course (n=5)	No Course (n=8)
Design	Create sketches or diagrams prior to coding	2.60	2.25
	Write code fragments on paper prior to coding	1.80	1.88
Reuse	Share your scripts and/or plugins with others	3.20	4.25
	Borrow pieces of code from others' scripts	2.80	3.25
	Borrow pieces of code from of your own	4.00	3.75
Testing	Test your scripts on multiple files/inputs	4.40	3.63
	Invite others to test your scripts	4.00	3.25

Average ratings from 1 to 5 where 1=never, 3=regularly, and 5=always.

as a motivation for *self-disclosing tools*⁴ to help users learn scripting from tool use [6].

Code reuse seemed to play a substantial role for our survey respondents. Borrowing code or code fragments from others (and oneself) was shown to be a regular activity within this community. Further, users indicated that they almost always share their code with others. Interestingly, those who had formal training in Computer Science appeared to be less likely to make their scripts available to others. However, when asked how often one writes scripts intentionally for other people's use we noted a "less than regular" (2.62, n=13) response consistent across both sub-groups. We believe this potential mismatch between intention during development and script usage in practice could lead to difficulties in reading and understanding programs written by others.

Script testing was also a regular practice for these users. Those with formal education in computing were slightly more likely to engage in testing practices, but even the no-coursework group showed testing on a more than regular basis. Of the testing habits investigated, respondents were least likely to test on multiple platforms (2.46 mean response, n=13). Free form responses about testing showed a variety of testing processes, but most described an incremental test-as-you-go approach. It was not possible to infer any more specific information about actual testing habits from our data (e.g., code-coverage, test case generation, etc), but a more detailed look at testing in practice could lead to additional insights in the future.

Overall, survey responses provided evidence that graphic design end-users have adopted practices that resemble parts of a more formal software engineering process. Pre-planning and design were the least likely activities, but they did engage in testing and code reuse frequently. There was also evidence that these end-users might benefit from additional knowledge about software development. For example, all but one user reported that their largest program was over

⁴Essentially, self-disclosing tools are applications that associate a script function to each possible user interface interaction and make the code for all function invocations visible to the user.

46 lines of code. While this may not seem like a large program, we know that code fragments as short as five lines can be candidates for the *Extract Method* refactoring technique [8]. Additionally, issues of *cohesion* and *coupling* can greatly impact the understandability, and therefore share-ability, of a program [19]. These concerns might be of minimal importance for an end-user who programs in isolation, but all indications are that these users thrive on collaboration and community support.

6. LEARNING STRATEGIES

As mentioned earlier, we were curious as to how these end-user programmers went about learning the programming knowledge they needed to achieve their goals. Respondents indicated that they most often learn about application features by experimentation on their own, and 78% reported turning to software tutorials (n=18).

We duplicated a series of questions from the Rosson web-developer study [16] that inquired about a number of different possible sources for learning. We asked, “If you were attempting to learn a new task with a piece of software, how likely would you be to consult the following resources for assistance?” Table 5 outlines their average responses on a scale of one (not likely) to five (very likely) for several resources. Similar to what was found with informal web developers, we noted the most highly ranked resources were related example code and FAQs/tutorials. Considering the small sample sizes, we noted mostly minor variations between those who did and those who did not have formal Computer Science training. However, one learning resource was ranked noticeably higher among the no-coursework group. These users were more likely to turn to interactive wizards for help. In general, the two groups ranked the various resources in order of reliance comparably.

Table 5: Resources for Learning and Support

Source	Course (n=5)	No Course (n=8)
Examples of similar tasks from which you can borrow ideas and/or copy code	4.20	4.50
FAQs, books, tutorials or other documentation	4.60	4.13
A friend or coworker	2.80	3.25
An interactive software wizard that takes you step-by-step through the new task	1.20	3.00
A class or seminar	2.20	2.50
A software agent that makes suggestions on an as-needed basis	2.00	2.50
Technical support/telephone hotline	1.00	1.75

Average ratings from 1 (not likely to consult) to 5 (very likely to consult)

Admittedly, these questions do not capture the full extent of learning strategies employed. It stands to reason that many of these users would have also indicated reliance on online forums given that they were recruited from those communities. More importantly, we need to understand deeper

questions like why they place value in certain resources and how they determine whether particular examples are relevant to a task at hand. Answers to these questions could provide additional rationale for designing new educational supports. Such questions were not addressed in this preliminary survey, but follow-on studies will explore these areas explicitly.

Nonetheless, these early results do suggest some possible avenues to support end-user programmers. Heavy use of pre-existing examples and online documentation (noted here and elsewhere) points to *case-based design aids* [10] as a particularly close match to the informal learning style of end-user programmers. Systems, like STABLE [12], that leverage case libraries of multiple example solutions to problems appear to be a promising way to introduce software design concerns to users seeking reusable code fragments. Secondly, incorporating self-disclosure and other in-situ opportunities for learning could entice end-users to learn more about Computer Science. Their personal interest and curiosity could provide the needed motivation for learning if these opportunities were presented in line with their desires. For example, software might recommend: “To save time, you might consider using a tree. Would you like to know more?”

7. DISCUSSION

Results of this small-scale study indicate that graphic designers and other people engaged in media manipulation are taking part in significant end-user programming activities. Their software artifacts share many features with programs written by professional software developers, yet few of them have formal training in Computer Science. For them, programming serves as a means to save time and add features to their applications. They have similar characteristics and behaviors to those noted in studies of other end-user domains. They borrow code from examples and rely on documentation like FAQs to accomplish their tasks.

These graphic designers who script appear to be discovering knowledge that is common in formal Computer Science learning environments. Our respondents knew a good deal about programming constructs, and some referred to processes similar to those espoused in software engineering courses. Given the scope of their activities and their sharing habits, we have reason to believe that they could benefit from knowing even more about formal software design and development practices.

Our results are by no means conclusive. Small sample sizes rule out a number of analysis techniques, and the true extent of this end-user community is unclear from the data given here. We still lack a complete picture of the typical user in this domain with respect to their Computer Science knowledge, how they learn, and their motivations to do so. However, this exploratory study indicates much potential. We hope to further investigate many of these issues in the future with a more widely distributed survey and through qualitative interviews with graphic design professionals. Eventually we aim to use our knowledge of this population to create tools that scaffold Computer Science education within the context of normal software interactions.

On a larger scale, further identification of end-user programmers could be beneficial to the Computer Science education community. In a time of declining enrollments [22] and the need to attract more non-CS majors to our courses,

end-users who take up programming could represent a set of possibilities. For end-users, like those studied here, learning to program is a natural progression. As a user has more and more sophisticated needs, she eventually outgrows the standard affordances of her tools. Scripting is a way for her to build the enhanced graphic effect or to save countless hours of manual work. Much of the motivation is intrinsic, and her real-life situation provides the context for her CS learning. Not only might the learn-as-you-go strategies enacted by these users help increase motivation in classrooms, but the settings in which end-user programming takes place might also suggest new contextualizations (like Media Computation [7]) to be leveraged by CS educators.

All indications are that the results given here are just the tip of a proverbial iceberg. In the space of image manipulation there are many packages outside the scope of this study. For example ImageJ is an open-source package used by scientists to process and view data, and it even boasts an academic-style conference⁵ for its users! Beyond that, there are scriptable 3D media (e.g., Maya, Blender) and digital video (e.g., Final Cut Pro) tools. All these only constitute one segment of the much larger end-user programming picture. This presents a unique opportunity to study Computer Science learning in the end-user's domain. Gaining a better understanding of how to support learning that takes place in these settings can only serve to enhance our understanding of computing education in general.

8. ACKNOWLEDGMENTS

We extend our gratitude to the online communities for allowing us to recruit from their membership and to the participants who volunteered to be part of our study. We also thank Allison Elliott Tew for her comments on earlier drafts of this paper.

This material is based upon work supported in part by the National Science Foundation under CISE Educational Innovations Grant No. 0306050.

9. REFERENCES

- [1] Adobe Systems Incorporated, San Jose, CA. *Adobe Photoshop CS2 JavaScript Scripting Reference*, 2005. Available online: <http://partners.adobe.com/public/developer/en/photoshop/sdk/JavaScriptReferenceGuide.pdf>.
- [2] A. Bruckman. Situated support for learning: Storm's weekend with Rachael. *The Journal of the Learning Sciences*, 9(3):329–372, 2000.
- [3] A. Bruckman, C. Jensen, and A. DeBonte. Gender and programming achievement in a cscl environment. In G. Stahl, editor, *Proceedings of CSCL 2002*, pages 119–127, 2002.
- [4] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace. End-user software engineering with assertions in the spreadsheet paradigm. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, pages 93–103, 2003.
- [5] Center for Children and Technology. Evaluation of the Intel Computer Clubhouse network: Year 2 report. Research Rep., New York, NY, September 2002.
- [6] C. DiGiano and M. Eisenberg. Self-disclosing design tools: a gentle introduction to end-user programming. In *DIS '95: Proceedings of the Conference on Designing Interactive Systems*, pages 189–197, New York, NY, USA, 1995. ACM Press.
- [7] A. Forte and M. Guzdial. Motivation and non-majors in computer science: Identifying discrete audiences for introductory courses. *IEEE Transactions on Education*, 48(2):248–253, 2005.
- [8] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Aug. 1999.
- [9] T. L. Friedman. *The World is Flat: A Brief History of the Twenty-First Century*. Farrar, Straus, and Giroux, New York, NY, 2005.
- [10] A. K. Goel, J. L. Kolodner, M. Pearce, R. Billington, and C. Zimring. Towards a case-based tool for aiding conceptual design problem solving. In *Proc. of the Case-Based Reasoning Workshop*, pages 109–120, Washington, D.C., 1991.
- [11] M. Guzdial, L. Hohmann, M. Konneman, C. Walton, and E. Soloway. Supporting programming and learning-to-program with an integrated CAD and scaffolding workbench. *Interactive Learning Environments*, 6(1/2):143–179, 1998.
- [12] M. Guzdial and C. Kehoe. Apprenticeship-based learning environments: A principled approach to providing software-realized scaffolding through hypermedia. *Journal of Interactive Learning Research*, 9(3/4):289–336, 1998.
- [13] B. A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, Cambridge, MA, 1993.
- [14] R. R. Panko. Finding spreadsheet errors: Most spreadsheet models have design flaws that may lead to long-term miscalculation. *Information Week*, (529):100, May 1995.
- [15] R. R. Panko. What we know about spreadsheet errors. *Journal of End User Computing*, 10(2):15–21, 1998.
- [16] M. B. Rosson, J. Ballin, and J. Rode. Who, what, and how: A survey of informal and professional web developers. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 199–206, 2005.
- [17] C. Scaffidi, A. Ko, B. Myers, and M. Shaw. Identifying categories of end users based on the abstractions that they create. Technical Report CMU-ISRI-05-110, Institute for Software Research, International, Carnegie Mellon University, Pittsburgh, PA, December 2005.
- [18] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 207–214, 2005.
- [19] I. Sommerville. *Software Engineering*. Addison-Wesley, Harlow, UK, fifth edition, 1996.
- [20] J. Spohrer and E. Soloway. Putting it all together is hard for novice programmers. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, November 1985.
- [21] C. Titmus. *Lifelong Education for Adults: An International Handbook*. Pergamon, Oxford, UK, 1989.
- [22] J. Vegso. Interest in CS as a major drops among incoming freshmen. *Computing Research News*, 17(3), 2005.

⁵The second ImageJ User and Developer Conference was held in May 2006.