Learning on the Web: A Case Study of Graphic Design End-User Programmers

Brian Dorn and Mark Guzdial
School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA 30332-0760
{dorn, guzdial}@cc.gatech.edu

ABSTRACT

An increasing number of users are exploring scripting and programming to enhance their Web experiences. As a result, the processes by which these non-professional programmers acquire knowledge of computing deserve our attention. This position paper describes our formative work in understanding graphic design end-user programming communities and also introduces our current efforts to support them better. As users facing analogous concerns to those engaged in programming for the Web, we discuss how specific lessons learned from graphic designers may be more broadly applicable to other types of Web designers. We also highlight the importance of framing end-user development issues, at least in part, as educational challenges.

INTRODUCTION

The Web is packed with user-generated applications. For example, 8 million members of DeviantART have uploaded over 62 million submissions [1], and ProgrammableWeb has indexed 999 APIs and over 3400 mashups [2]. Web users can both consume and create applications, and an increasing movement towards openness is facilitating novice entry into the end-user developer community. Ironically, in such an information overloaded space, it has become increasingly difficult to find relevant, high-quality, transparent code examples from which to learn online. Our research is centered around supporting end-user programmers locate, access, and make sense of resources on the Web.

Many studies of end-user development (EUD) on the Web have focused on the design and use of tools to support end-user programmers—what tools will support their needs, and how do we increase usability of these tools? Approaching EUD problems from an alternative angle, our research seeks to understand the nature of end-user programmers' knowledge of main-stream programming and computing concepts—what computing techniques have been appropriated, where do end-users lack computing skills, and how can we support them in developing the necessary computing knowledge needed in order to better use the tools they currently have? This perspective helps augment our understandings about what makes end-user programming hard, and how we might bring the power of computation to a wider audience.

The context of our research is with graphic design end-user programmers. This is a particularly interesting set of users because they have a wide set of needs and interests, and their activities often directly impact the Web. They are engaged in a variety of development tasks: creating and maintaining websites, building Flash applications, and scripting custom extensions in tools like Adobe Photoshop. As with other groups of end-user programmers on the Web, few have been trained in formal computer science concepts, yet they need to appropriate sufficient skills and knowledge from computer science concepts in order to do their work. In this position paper, we describe some of our early formative work in understanding graphic design end-user programmer communities, and we outline our current studies in helping them to further develop computing expertise.

INFORMAL EDUCATION AND EUD

We recognize that software development is no longer confined to professionally trained computer programmers. Broadly defined, end-user programmers (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 [10]. 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.

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 [9, 14, 6]. Even in situations where EUPers can easily master a programming language, other challenges arise. Segal notes "depending on the context in

Submitted to CHI'09 Workshop: End User Programming for the Web

which the software is going to be used, issues such as code comprehensibility, software robustness and performance become important" [13, 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 security of third-party Web servers, to make important business decisions, or even to develop scientific theories. Gaps in end-user knowledge about software development are problematic and have real world consequences (e.g., [11]). Potential risks for graphic designers and Web developers may be different, but they are no less important.

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 as they go about solving tasks in the real world (e.g., [3, 12]). 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. [15, p. 547]

Re-envisioning end-user programmers as informal computer science learners opens numerous research directions. In addition to building new technologies that enable end-user development, we can 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? Specifically, we are focused on pedagogical, rather than technological, solutions to overcome barriers in end-user programming. 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 already takes place within end-user contexts.

FORMATIVE WORK

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. Those employed in the graphic design field also encounter scripting languages in Web-related job responsibilities. In some instances their scripting activities on the desktop are designed to generate Web content (e.g., automatically transforming a collection of wedding photos into a digital album for the Web).

Having identified this sub-group of end-user developers, we sought to gain a better understanding of their demographics and how they engage with scripting. In addition, we aimed to explicitly query their familiarity with computing concepts and how they go about learning new skills.

We adapted Rosson, Ballin, and Rode's survey of Web developers [12] to create a 39-question survey directed at users of image manipulation packages like Adobe Photoshop, Illustrator, and GIMP [3]. These software programs have specific support for built-in scripting languages. We asked questions about several different aspects of end-user programming: tool use habits, motivation for scripting, script development behaviors, programming concept familiarity, and general background. The survey was distributed in online communities where users seek information about scripting in these tools.

The majority of the responses came from advanced computer users in an art or media related field. Typical careers named were graphic design, photography, and Web development. Most had formal training in fields other than computer science and did not self-identify as programmers.

However, other data gathered suggested that these users were very much engaged in sophisticated programming tasks. For example, one user stated that he used scripting to "build database-populated pages for print and CD catalog distribution." The most common applications for scripting were the automation of repetitive tasks and creating custom image-editing effects that are not standard components of the host application. These applications of automation 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).

Code reuse seemed to play a substantial role for our survey respondents. Borrowing code or code fragments from others (and oneself) was a typical activity within this community. Further, users indicated that they almost always shared their code with others. Though we did not inquire about the specific means by which this sharing took place, it is reasonable to assume that they rely on Web-based communities in order to connect with non-collocated users. Paradoxically, they also reported that they rarely write scripts intending for other people to use them. This potential mismatch between intention during development and script usage in practice could lead to difficulties in reading and understanding programs at a later time.

In order to probe graphic designers' learning behaviors, we replicated a series of questions about learning from the Rosson et al. study [12]. We asked participants about their likelihood to consult a variety of resources in order to learn something new. Responses overwhelmingly favored the use of existing program code that is similar to the current task and the use of FAQs and tutorials. These results closely mirror the study of Web developers which also indicates that

educational resources of use to graphic designers may also be useful to the larger Web EUD community.

CONTENT ANALYSIS FROM THE WEB

From these early findings, we wanted next to understand in what ways online communities provide examples from which other end users learn. We conducted a second study with the goal of closely examining a repository of example code with respect to the type of introductory computing constructs embodied therein. We analyzed the content of a popular Photoshop scripting repository to uncover the breadth of support EUPers would be likely to find when searching through online examples for assistance [4, 5]. Such a depiction is an important step in understanding the computing constructs end-user programmers are likely to encounter when seeking help, as well as an indication of those concepts that may need additional attention if we wish users to engage with them.

We developed a coding scheme that considered both general introductory computing constructs, informed by the CS education literature, as well as EUP domain-specific constructs, suggested by EUP studies and derived in a data-driven manner by the scripts. Using this scheme, we then conducted an artifact analysis of all publicly available scripts hosted in the Photoshop section of the Adobe Exchange repository, an end-user programming community for graphic designers.

The most frequent programming constructs were: variable, assignment, relational operators, selection, number datatypes, and string. The least common were: indefinite loop, nested loops, recursion, type conversion, user-defined objects, and exporting/importing code. Some of these observations could be tied to language influences (e.g., creation of instantiable objects in JavaScript is awkward). Others, like the step-down in frequency of definite loops to nested loops to recursion, seem indicative of conceptual difficulties noted in the novice programming literature.

Projects in the repository contained a number of interesting observations with respect to code reuse and sharing in the community. Given that our earlier study reported a high degree of sharing, it was significant that there was a lack of examples where code was modularized into reusable entities. However, script authors clearly exhibited concerns about sharing, code ownership, and intellectual property. A majority of scripts contained copyright notices, nearly half contained some form of end user license agreement, and over one-fifth gave explicit credit to other people who had helped in script development.

These findings are important for both graphic design end-user programmers and those engaged in development on the Web more broadly. They provide insight into the types computing concepts that are being introduced while a user searches for relevant examples. They also indicate that what the user finds may not always be ideal. Further, they highlight the importance of creative credit in Web-based, end-user development environments, particularly those that leverage the sharing of code artifacts.

CURRENT DIRECTIONS

Our current research involves developing and deploying a Web-based learning tool to scaffold graphic design end-user programmers. The primary motivation is to aid them in developing more expert knowledge about programming as they go about their normal activities, like searching for example code. The results of our initial studies suggest that there is potential to use case-based learning aids to scaffold learning about computer science topics among this community. 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 [8, 7]. Thus, case-based learning aids give learners a set of worked examples that highlight particular pieces information relevant to the solution. Our goals are to develop an educational resource which is easily integrated into current informal learning practices and to show that such a resource can promote learning of mainstream computer science concepts more deeply than other currently available information sources.

Our driving research questions are:

1. What is the nature of graphic design end-user programmers' knowledge of normative computing concepts?
2. How does the presentation context of the information in a library of cases influence the way it is consumed by end users?
3. To what extent do the case-based materials support appropriation of computing knowledge?

To address these questions, we are conducting studies with graphic design end-user programmers from the Atlanta area. After semi-structured design studies with these participants, we will develop a Web based collection (i.e., a library) of example Photoshop scripting projects to support this class of users. To illustrate with an example from such a library, 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, a code listing of the final solution, and links to other relevant cases. The narrative would also introduce the computing concepts which are necessary to solve the problem in general (e.g., recursion in this particular case).

We intend to compare this experimental library to a controlled collection of examples that are written in a more general context (i.e., without specific reference to Photoshop). We hypothesize that in addition to providing useful examples, these well-formed cases can increase end-user programmer knowledge of computing concepts that will further their programming abilities. Further, we hypothesize that users will identify the Photoshop cases as a particularly valuable way to present new information.

CONCLUDING REMARKS

The overarching goal of our research is to explore ways of supporting the education of end-user programmers. In seek-

ing to better understand these informal learning practices, we hope to provide insights to both the computer science education and EUD research communities that will help advance the state of the art in end-user software creation.

In our research community's efforts to empower users on the Web with additional computational tools, we must take a multifaceted approach. New programming systems, new computational metaphors, and new online communities that foster end-user development are all necessary. So too are research efforts rooted in educational traditions which help uncover the ways in which users make sense of this new array of tools.

ACKNOWLEDGMENTS

This research is supported in part by NSF ITR-SoD #0613738.

REFERENCES

1. DeviantArt. Wikipedia. Retrieved Oct 31, 2008 from `http://en.wikipedia.org/wiki/DeviantArt`.

2. ProgrammableWeb - mashups, APIs, and the web as platform. Retrieved Oct 31, 2008 from `http://www.programmableweb.com/`.

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

4. B. Dorn, A. E. Tew, and M. Guzdial. 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*, pages 27–30, 2007.

5. B. Dorn, A. E. Tew, and M. Guzdial. Computer science construct use, learning, and creative credit in a graphic design community. Technical Report GT-IC-08-01, Georgia Insitute of Technology, School of Interactive Computing, Atlanta, GA, 2008.

6. B. Du Boulay. Some difficulties of learning to program. In E. Soloway and J. Spohrer, editors, *Studying the Novice Programmer*, pages 283–299. LEA, Hillsdale, NJ, 1989.

7. 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.

8. 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.

9. A. J. Ko, B. A. Myers, and H. H. Aung. Six learning barriers in end-user programming systems. In *Proceedings of the 2004 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'04)*, pages 199–206, 2004.

10. B. A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, Cambridge, MA, 1993.

11. 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.

12. M. B. Rosson, J. Ballin, and J. Rode. Who, what, and how: A survey of informal and professional web developers. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 199–206, 2005.

13. J. Segal. Some problems of professional end user developers. In *VL/HCC'07: Proceedings of the 2007 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 111–118, 2007.

14. 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.

15. C. Titmus. *Lifelong Education for Adults: An International Handbook*. Pergamon, Oxford, UK, 1989.