Workshop Report: From End-User Programming to End-User Software Engineering (a CHI'06 Workshop)

Abstract: End users create software when they use spreadsheet systems, web authoring tools and graphical languages, and when they create educational simulations, macros-by-demonstration, and dynamic e-business web applications and mash-ups. Some end-user developers, such as accountants or teachers, may have no formal training at all in programming. Others, such as scientists or sysadmins, may have significant background in programming, but they still do not see their goals as producing software: rather, they see software as a means to some other goal, such as visualizing a scientific phenomenon or getting a computer set up so that it can talk to the printers in the building. It has become well known that errors are pervasive in end-user software, and the resulting impact is sometimes enormous. A growing number of researchers and developers are working on ways to make the software created by end users more reliable, by looking beyond the "create" aspect of the software lifecycle.

The WEUSE workshop (Workshop on End-User Software Engineering), organized by Margaret Burnett, Brad Myers, Mary Beth Rosson, and Susan Wiedenbeck for CHI'06, brought together academic and industrial researchers who are addressing this topic with industry representatives who are deploying end-user programming systems. The objective was to facilitate sharing of real-world problems and solutions. Four real-world end-user software engineering issues formed the focus of the CHI'06 workshop: extending an open source paradigm for end-user developers of MATLAB programs, helping disparate audiences learn to use the Flash end-user programming product, dealing with mission-critical aspects of end-user programming situations that occur in customization of healthcare systems, and supporting the high-pressure development done by sysadmins.

Workshop Report: From End-User Programming to End-User Software Engineering (a CHI'06 Workshop)

Margaret Burnett[1], Brad Myers[2], Mary Beth Rosson[3], Susan Wiedenbeck[4], and Adam Leibel[1]

[1]Oregon State University, [2]Carnegie Mellon University, [3]Pennsylvania State University, [4]Drexel University

Introduction

End users create software when they use spreadsheet systems, web authoring tools and graphical languages, and when they create educational simulations, macros-by-demonstration, and dynamic e-business web applications and mash-ups. Some end-user developers, such as accountants or teachers, may have no formal training at all in programming.

It has become well known that errors are pervasive in end-user software, and the resulting impact is sometimes enormous. A growing number of researchers and developers are developing ways to make the software created by end users more reliable. In general, their research question is:

Is it possible to bring the benefits of rigorous software engineering methodologies to end users?

The WEUSE workshop (Workshop on End-User Software Engineering), organized by Margaret Burnett, Brad Myers, Mary Beth Rosson, and Susan Wiedenbeck for CHI'06, brought together academic and industrial researchers who are addressing this topic with industry representatives developing end-user programming systems. The objective was to facilitate sharing of real-world problems and solutions. Four real-world end-user software engineering issues formed the focus of the CHI'06 workshop: extending an open source paradigm for end-user developers of MATLAB programs, helping disparate audiences learn to use the Flash end-user programming product, dealing with mission-critical aspects of end-user programming situations that occur in customization of healthcare systems, and supporting the high-pressure development done by sysadmins. A fifth scenario also occupied some discussion, relating to business people's customization in the context of web services, but as this was not part of the central focus, we refer the reader to the complete notes and proceedings for this aspect (http://eusesconsortium.org/weuse/).

Issue #1: Open Source for End Users

MathWorks, the maker of the MATLAB product, is working to connect the open source paradigm with the end-user programmers of MATLAB. Some of their users write software to solve a single problem without considering whether their code could be useful to other users. The company has been working to develop an open source community for their end-user programmers to share code—both to help users learn to use MATLAB more effectively and to provide code for problems already tackled by other users. The idea is to devise an open source paradigm that is viable for end-user developers. Issues they are dealing with include how to help ensure reasonable quality of the contributions, and how to encourage MATLAB users to invest their time contributing to an open source repository. MATLAB has an installed base of users numbering in the hundreds of thousands, so if the open source paradigm catches on in that community, each contribution could be of potential assistance to a large number of users.

Getting users interested in an open source paradigm was seen to be the largest hurdle—far more significant than how to build mechanisms that would allow end users to, for example, upload or search for contributed code. Many end-user developers do not have experience with the concept of code reuse or code review. Further, they may not have confidence that the code they have written is "good enough" to offer to others for viewing or editing. As the discussants put it, there is a need to develop not just tools, but clear *processes,* for an end-user open source paradigm to succeed.

Lowering the barriers to entry and thinking about what might seem rewarding about contributing code was a common theme throughout the discussion. The company is already succeeding at this to some extent, but more could be done. One idea put forward was to provide users a straightforward way to broadcast or post requests for code they needed, so that another user might become inspired to contribute a program from hearing about a specific need for it by some other user. Another possibility was to provide a MATLAB community discussion space in which more knowledgeable users could talk with users in need of help with MATLAB problems. These shared discussions would be likely to generate some example code, which might then help to encourage a code-sharing mentality.

An idea MathWorks is already working on is a reputation incentive, enabling users who contribute especially useful and usable MATLAB programs to receive recognition. Quality is a major issue at this point. The company has been experimenting with voting or quick reviews by others who have used the contributed code, similar to internet e-business community approaches used by EBay, Amazon, travel sites, movie sites, and so on. Small sub-communities were also considered as a way to help build quality, as in the CSCW community. Small communities could self-identify based on the particular topic or domain of interest to the users. Trust may arise more easily in such small communities where everyone is known, making a more comfortable forum for discussing quality issues in contributed code.

Issue #2: First, Do No Harm

GE Healthcare produces an electronic medical record (EMR) system to assist small medical offices with digitizing their patient records. Although, these offices are typically too small to support a technical staff member. One potential obstacle to adoption of any such system can be

an office's resistance to changing procedures, and their use of record formats that do not fit well with the procedures and screens/reports built into the software. To help address this obstacle, the software includes a structured editor that enables office staff to drag and drop input and output fields onto their screens, thereby facilitating customization of the software's behavior to better match their existing procedures.

Given such customization, how can the company help its customers avoid customization design oversights or simple slip-ups by the medical staff who are creating the customization? Customization bugs, whether logic errors or usability flaws, could lead to serious enough problems in data entry to cause harm to a patient. The problem is providing a robust enough customization system while protecting patients from faulty setups.

This is a classic end-user software engineering problem, but it is exacerbated by the "mission critical" nature of its potential impact on a patient's health. Most end-user programmers in this setting do not have much background in designing system changes, programming them, and debugging them. One proposed idea was that the makers of the EMR system might encourage users to work together with a community site, as in the MATLAB discussion. Such a site might contain common templates, examples, discussions among the users, tutorials, and perhaps an easy connection to experts of the system. Several approaches to end-user software engineering from other domains could be brought to bear on these issues, such as the approaches to end-user debugging, automatic fault localization, and systematic testing summarized in [WEUSE 2006]. Discussions arose about natural language inputs and semantic data categories, incorporating some of the recent advances from AI research being brought to bear on end-user programming (see [WEUSE 2006] for examples). Error mining, best practices, and patterns also arose as ways users could help each other. Finding ways to leverage existing XML technology that has already been applied to the medical community was suggested as an enabling technology that could help with some of these efforts.

Issue #3: One Product, Disparate Audiences

Adobe brought a developing concern about their Flash product to the workshop: over the years, their product has been adopted by at least two very different user populations. Graphic designers and animators use Flash as a creativity medium for creating and working with animations. Software developers, on the other hand, use Flash to develop animated applications instead of using traditional web programming languages to develop such applications. These different user communities often need different features. In particular, the sizable software development constituency requires features that are too complex for most animators, and the product has become more and more technical over the years. The continuing addition of such features makes the system increasingly difficult for designers and animators to use.

Because the issue is situated in the realities of the software business, the discussion stayed grounded within these realities. Thus, since the product is not likely to become simpler or to diverge into two versions, the discussion focused primarily on explicit mechanisms to support both audiences in understanding the features and in debugging their Flash programs.

Flash already has a substantial amount of learning/understanding support available, including thousands of pages of documentation. Even so, many people feel it necessary to start with

courses. Discussions on how better to support the different audiences included making the available documentation more task-oriented (as per the principles of Minimalist Learning Theory [Carroll 1990]) and making Flash responsive enough to teach users "on demand", such as allowing them to click on an object to see what it does.

There are a number of approaches that have been developed by researchers that might be pertinent to the debugging aspect. Some of these ideas included interactive viewing of forward and backward execution (e.g., [Burnett et al. 2000, Lieberman and Fry 1995]), visual displays of variables and values causing changes (e.g., [Lieberman and Fry 1995]), making use of end-user assertions in a manner similar to spreadsheet research [Burnett et al. 2003], and asking "why" and "why not" questions about the behavior of their programs [Ko and Myers 2004]. Community-based mutual support via discussion forums and wikis also emerged as an idea for this issue, as it had for the other issues. Finally, a number of suggestions centered around increasing the availability of visual/textual alternatives, such as generating more of the code by demonstration (e.g., [Lieberman 2001]), and viewing the animation visually and code textually side by side in a synchronized fashion (e.g., [Reiss 1985]).

Issue #4: End-User Development Under Pressure

The system administrator (sysadmin) is an example of an end-user programmer. Some sysadmins are formally trained in programming and some are not, but regardless of their backgrounds, writing code in the form of scripts is an almost daily job. Like other end-user programmers, their deliverable at the end of the day is not software per se—it is a solution to a problem that has arisen for some set of users. These problems often arise with no notice and an urgent need for a solution. Further, every shop is different, and sysadmins manage installations of diverse equipment from different vendors; thus off-the-shelf tools can rarely be found for the solution required. IBM reported that in their survey of sysadmins, they found many examples of custom written programs and scripts for tasks such as monitoring systems, converting data, presentation of data to users, and maintenance. In effect, every system administrator has a private toolbox of custom tools for keeping their systems up. IBM would like to find ways to help sysadmins share and reuse their solutions and tools with each other. Another issue in this situation is that debugging is very difficult, since errors have persistent side effects on systems that are in use. Finally, sysadmins' jobs consist of crisis responses almost every day, and the associated time pressure discourages extra reflection and quality control measures.

Workshop discussion pointed out that the theory of Attention Investment can help provide guidance on how to help sysadmins balance these constraints [Blackwell 2002]. Also, a fair amount of time was spent in thinking about "sanity checking," including systematic testing. Examples included tools for simulating input for scripts, with a key feature of allowing tests of extreme values, and tools to make it easy for sysadmins to get multiple sources of input for verification. Another idea was to promote a culture in which testing and sharing is encouraged. Language-independent approaches arose, such as natural language [Liu and Lieberman 2004] or search-based techniques [Little and Miller 2006]. Another key may lie in the notions of abstractions used by these users [Scaffidi et al. 2005] and how these could be identified, shared, and reused.

Conclusion

The workshop's goals were two-fold. The first was to generally share information and raise awareness among researchers already in this area with researchers in the related areas of Empirical Studies of Programming and Psychology of Programming, and with practitioners interested in current and future techniques that can be embodied in tools and development processes. The second was to concretely match end-user software engineering problems in industry with potential solutions drawn from new and emerging research findings, resulting in technology transfer in both directions.

These goals were achieved. Several attendees commented that it was one of the most valuable workshops they had attended. Several of the industrial participants left with specific follow-up items they wanted to try, and at least three follow-on collaboration efforts involving attendees at the workshop who had not worked together before emerged. A survey paper on the state of the art in end-user software engineering is also emerging, building upon the findings from this workshop.

Interested readers may consult the detailed notes and full proceedings at the workshop web site: http://eusesconsortium.org/weuse/. Also of interest may be the on-line site for end-user software engineering research, located at http://eusesconsortium.org/.

Acknowledgments

We thank the participants of the workshop for their contributions. The organizational work for this workshop was supported in part by the EUSES Consortium under NSF Grant ITR-0325273 / 0324861 / 0324770 / 0324844 / 0405612.

References

[Blackwell 2002] Alan Blackwell, "First steps in programming: a rationale for attention investment models," *Proc. IEEE Human-Centric Computing Languages and Environments,* Arlington, VA, Sept. 3-6, 2002, 2-10.

[Burnett et al. 2000] Margaret Burnett, Nanyu Cao, and John Atwood, "Time in Grid-Oriented VPLs: Just Another Dimension?" *2000 IEEE Symposium on Visual Languages*, Seattle, Washington, 137-144, Sept. 10-13, 2000.

[Burnett et al. 2003] Margaret Burnett, Curtis Cook, Omkar Pendse, Gregg Rothermel, Jay Summet, and Christine Wallace, "End-User Software Engineering with Assertions in the Spreadsheet Paradigm," *Internatinoal Conference on Software Engineering*, Portland, OR, May 3-10, 2003, 93-103.

[Carroll 1990] John Carroll, *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*, MIT Press, Cambridge, MA, 1990.

[Ko and Myers 2004] Andrew Ko and Brad Myers, "Designing the Whyline: A Debugging Interface for Asking Questions about Program Behvarior," *ACM Conference on Human Factors in Computing Systems*, Vienna, Austria, Apr 24-29, 2004, pp. 151-158.

[Lieberman and Fry 1995] Henry Lieberman and Christopher Fry, "Bridging the Gap Between Code and Behavior in Programming," *ACM Conference on Computers and Human Interface (CHI'95)*, Denver, Colorado, April 1995.

[Lieberman 2001] Henry Lieberman, editor, *Your Wish Is My Command: Programming By Example*, Academic Press, San Diego, CA, 2001.

[Liu and Lieberman 2004] Hugo Liu and Henry Lieberman. "Toward a Programmatic Semantics of Natural Language," *IEEE Symposium on Visual Languages and Human-Centric Computing*, Sept. 27-30, 2004, 281-282.

[Little and Miller 2006] Greg Little and Robert C. Miller. "Translating Keyword Commands into Executable Code." *ACM Conference on User Interface Software and Technology* (UIST), 2006

[Reiss 1985] Steven P. Reiss, "PECAN: Program Development Systems that Support Multiple Views", *IEEE Trans. on Software Engineering*, SE-11 (3), March 1985.

[Scaffidi et al. 2005] Chris Scaffidi, Andrew Ko, Brad Myers, and Mary Shaw, Identifying Categories of End Users Based on the Abstractions That They Create, Technical Report CMU-ISRI-05-110/CMU-HCII-05-101, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2005.

[WEUSE 2006] WEUSE II attendees, *Proceedings of The Next Step: From End-User Programming to End-User Software Engineering,* (Margaret Burnett, Brad Myers, Mary Beth Rosson, Susan Wiedenbeck, eds.), April 2006. http://eusesconsortium.org/weuse/proceedings.php