End-user software engineering

Christopher Scaffidi and Margaret Burnett

Oregon State University

End-user software engineering (EUSE) refers to processes and technologies that enable ordinary end users to create, perfect, share, and otherwise work with their own software to achieve quality high enough for its intended use (Ko 2010; Nardi 1993). Thus, EUSE combines the goal of end-user programming (EUP), which focuses on enabling end users to create software, with the concern for quality of that software across its *entire lifecycle* (not just the "create" stage).

The distinction between EUSE and EUP became increasingly important in the late 1990's, as people became more reliant on software they created for themselves using spreadsheets, database systems, scripting languages, and other environments. Today, research based on U.S. Bureau of Census and Bureau of Labor data indicates that at least a sixth of the American population now use these tools at work (Scaffidi et al. 2005), highlighting the potentially enormous impact of EUSE.

Challenges of EUSE

The software qualities relevant to EUSE are the same as those of interest to professional software engineers who sell their products. These qualities include attention to functional correctness, reliability, performance, maintainability, reusability, privacy, and security.

However, EUSE is inherently different from traditional software engineering, because simply mimicking traditional approaches is often insufficient to produce successful results.

One reason is that end users usually do not have the kind of training and background that professional programmers have. In particular, end users are not likely to know about quality control mechanisms, formal development processes, modeling diagrams, or test adequacy criteria, and are not likely to invest time learning about such things.

Even more important, end users face fundamentally different motivations and work constraints than professional programmers do. Specifically, end users' goal is not to optimize the quality of their software per se. Rather, their real goal is to complete some immediate task in some non-programming domain such as accounting, teaching, managing safety, understanding financial data, or authoring new media-based experiences. Because of their focus on an immediate goal other than programming, end users do not approach programming with the same attention or awareness of quality issues as professional software engineers often do. EUSE research is therefore largely concerned with providing processes and technologies that enable end-user programmers to achieve adequate levels of quality without forcing end users to lose their focus on non-programming goals.

EUSE research on software reliability

EUSE research to date has largely been concentrated on helping users to detect and fix bugs. From an HCI standpoint, this can be viewed in terms of helping users to cross the Gulf of Evaluation (Norman 1988) to determine whether they have put their programs into the desired state, and to cross the Gulf of Execution to determine how to fix programs so that they end up in the desired state.

One of the first directions toward this goal was to help users evaluate whether their programs contained bugs by encouraging end users to test strategically. Perhaps the most developed end-user testing approach is "What You See Is What You Test" (WYSIWYT),

which guides users through the process of systematically testing spreadsheets (Fisher et al. 2006). (The approach has also been investigated with other end-user programming platforms, but is most thoroughly investigated in the spreadsheet paradigm.) The WYSIWYT approach entices users through a "Surprise-Explain-Reward" strategy (Wilson et al. 2003), in which surprises such as colored borders attract users' attention to areas of the spreadsheet that need testing, tool tips explain the colors' meaning and the potential reward in using the testing devices. Behind the scenes, WYSIWYT uses a formal test adequacy criterion to reason about elements of the formulas that have been covered by tests so far. WYSIWYT's benefit is that it helps users to strategically select cells to test in order to systematically exercise data dependencies in formulas, and empirical studies have shown that the approach does help improve end users' ability to find and fix spreadsheet errors.

Another approach to error detection is specification-based: assessing whether initial or intermediate computations of a spreadsheet or other program satisfy an assertion or other specification of correct behavior (Burnett et al. 2003; Koesnandar et al. 2008; Scaffidi et al. 2008). For example, a "web macro" is a type of script that reads data from a website, computes with it, and then generates outputs that might be posted to another website (Little et al. 2007). When the program is initially created, it might perform properly; however, upon later execution, invalid outputs might arise either because of a bug in the macro itself or because of changes in the structure and content of websites (the macro inputs). An assertion can catch such errors that arise, halt execution, and bring them to the user's attention to prevent the macro from running awry. In order to help programmers construct assertions, researchers have developed techniques for automatically or semi-automatically generating assertions. For example, tools can infer potential assertions based on prior executions, or they

can generate assertions on outputs based on how they are computed from other outputs that already have assertions.

A third, related approach for finding errors in programs is for the programming tool to automatically look for errors on the basis of types, dimensions, or units (Erwig and Burnett 2002; Abraham and Erwig 2004; Coblenz et al. 2005; Chambers and Erwig 2009). This approach can be regarded as specific kinds of assertions. For example, one system associates types with spreadsheet cells (based on the placement of labels at the top of columns and at the left end of rows) and specifies how these types propagate through the spreadsheet. If two cells with different types are combined, then their type is generalized if an applicable type is available (e.g.: “3 apples + 3 oranges = 6 fruit”), or else an error message is shown.

After a programming error is detected, the next step is to figure out how to remove it by debugging. A new class of debugging tools based on *question asking* has recently emerged and has proven effective in EUSE. The first tool to take this approach was the Whyline, which was prototyped for the Alice programming environment that enables users to program animations (Ko and Myers 2004). Users execute their program, and when they see a behavior they have a question about, they press a “Why” button. This brings up a menu of “why did” and “why didn’t” questions, organized according to the structure of the visible 3D objects manipulated by the program. Once the user selects a question, the system analyzes the program’s execution history and generates an answer explaining the error in terms of the events that occurred during execution. The Whyline approach has also been applied to debugging other kinds of programs (e.g., Ko 2008; Kulesza et al. 2009).

EUSE research on software reusability

Reusability is a crucial quality attribute in professional software development, as reusing programs can greatly improve productivity, but it has not received as much attention in EUSES research to date as reliability. Supporting reuse of end-user programs is challenging because end-user programmers rarely have the opportunity or training required to design highly reusable programs. Therefore, even though repository systems can make it easy for end-user programmers to post programs for others to reuse, it can be extremely time-consuming for other programmers to evaluate the reusability of these programs.

To help reduce the difficulty of reusing programs, models of what makes end-user programs reusable are now being developed in the hopes of helping users to search repositories for reusable programs related to the user's particular interests (Scaffidi 2010). Outside of repositories, other work has begun to explore how to help users to extract reusable pieces of software from existing applications (e.g., Oney 2009). Yet even after all or part of a program is retrieved from a repository or extracted from another program, there still is often the need to modify the program for a new use, and this aspect of end-user program reuse has not yet received significant attention.

The future and implications of EUSE

Beyond reliability and reusability, further EUSE research is also needed to help end-user programmers produce software with better attention to security, maintainability and other software quality attributes. For example, end-user programmers currently have no support for evaluating whether their programs create security holes, or whether their programs have enough scalability to handle the amounts of data that are likely to be encountered after the program goes into use. In order to help end-user programmers evaluate their software and fix any problems identified, it will be necessary for researchers to develop

new approaches, since approaches used by professional software developers, such as buffer-overrun analysis or Big-O analysis, may be irrelevant or too complex for the needs of end-user programmers. Other traditional approaches such as the use of design patterns might be relevant but need adaptation to meet the needs of end-user programmers (e.g., Diaz et al. 2008). Overall, research aimed at supporting qualities beyond reliability will continue to increase in importance as end-user programming plays a larger and more vital role in work and society.

More broadly, EUSE's continuing development as a social phenomenon has important implications for the relationship between end users and professional software developers (Fischer and Giaccardi 2006, Costabile et al. 2009). The rise of EUP to date enables end users to respond to professional developers' backlog of software work, and to the reality that professional software developers are not likely to understand and plan for every user requirement when developing software for them. With continuing advances in EUSE, end users will not only be able to create a variety of software on their own, but they will also to be able to assess and improve that software's quality on their own—so that they know to what extent to rely upon it, and what to do to increase the software's quality if needed. As a result, the fit between software's form and individual users' needs might be closer than has been possible before, vastly increasing the usefulness of software in peoples' lives.

How to learn more

The following resources provide an overview of EUSE and EUP. The first of these resources is a survey paper that includes a more detailed overview of the approaches we have summarized here, as well as pointers to primary references on these approaches. The other

resources listed below are books that discuss particular EUSE and EUP techniques and technologies that have been developed over the past two decades.

- Andrew Ko, Robin Abraham, Laura Beckwith, A. Blackwell, Margaret Burnett, Martin Erwig, Christopher Scaffidi, Joseph Lawrence, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. The State of the Art in End-User Software Engineering, ACM Computing Surveys, to appear.
- Allen Cypher, editor. Watch What I Do: Programming by Demonstration, MIT Press, 1993.
- Allen Cypher, Mira Dontcheva, Tessa Lau, and Jeffrey Nichols, editors. No Code Required: Giving Users Tools to Transform the Web, Elsevier Science, 2010.
- Henry Lieberman, editor. Your Wish is My Command: Programming by Example, MIT Press, 2001.
- Henry Lieberman, F. Paterno, and V. Wulf, editors. End User Development, Springer, 2006.
- Bonnie Nardi. A Small Matter of Programming: Perspectives on End User Computing, MIT Press, 1993.

Much of the research on EUSE has appeared in the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), with some also in the ACM Conference on Human Factors in Computing Systems (CHI) and the ACM/IEEE International Conference on Software Engineering (ICSE). A newer related conference series is the International Symposium on End User Development (ISEUD). The Workshop on End-User Software Engineering (WEUSE) is focused entirely on EUSE, and three of the workshops in the series were open meetings that generated online proceedings (WEUSE I,

WEUSE II, and WEUSE IV). Many EUSE researchers are members of the End-Users Shaping Effective Software (EUSES) consortium, which has a comprehensive listing of EUSE-related publications and resources on their web site (http://eusesconsortium.org/pubs/publications.php).

For estimates on the overall numbers of end-user programmers and kinds of EUP in America, see:

- Christopher Scaffidi, Mary Shaw, and Brad Myers. Estimating the Numbers of End Users and End User Programmers. Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2005), Dallas, TX, September 2005, 207-214.

For statistics on spreadsheet errors and stories of how errors impacted businesses, see:

- European Spreadsheet Risks Interest Group website (http://www.eusprig.org/stories.htm)
- Ray Panko. What We Know About Spreadsheet Errors, Journal of End User Computing (10), No. 2, 1998, 15-21.

The Wikipedia article on Quality Attributes provides an excellent list of the many qualities from the software engineering community that are also relevant to end-user software engineering (http://en.wikipedia.org/wiki/List_of_system_quality_attributes).

In addition to resources above, the following papers were cited in the body of this article:

- Robin Abraham and Martin Erwig, Header and Unit Inference for Spreadsheets through Spatial Analyses. IEEE Symposium on Visual Languages and Human-Centric Computing, September 2004, 165–172.
- Margaret Burnett, Curtis Cook, Omkar Pendse, Gregg Rothermel, Jay Summet, and Chris Wallace. End-User Software Engineering with Assertions in the Spreadsheet Paradigm. International Conference on Software Engineering, 2003, 93-103.
- Chris Chambers and Martin Erwig, Automatic Detection of Dimension Errors in Spreadsheets, Journal of Visual Languages and Computing, Vol. 20, No. 3, 2009
- Michael Coblenz, Andrew Ko, and Brad Myers. Using Objects of Measurement to Detect Spreadsheet Errors. IEEE Symposium on Visual Languages and Human-Centric Computing, 2005, 314-316.
- Maria Francesca Costabile, Piero Mussio, Loredana Provenza, and Antonio Piccinno. Supporting End Users to be Co-Designers of their Tools. 2nd International Symposium on End-User Development (LNCS 5435), Siegen, Germany, Springer-Verlag, March 2-4, 2009, 70-85.
- Paloma Diaz, Ignacio Aedo, and Mary Beth Rosson, "Visual representation of design patterns for end users", Proceedings of Advanced Visual Interfaces: Advanced Visual Interfaces 2008, Naples, Italy, May 28-30, 2008.
- Martin Erwig and Margaret Burnett. Adding Apples and Oranges. 4th International Symposium on Practical Aspects of Declarative Languages, 2002, 173-191.
- Gerhard Fischer and Elisa Giaccardi. Meta-Design: A Framework for the Future of End User Development. In Henry Lieberman, Fabio Paternò, & Volker Wulf (Eds.), End User Development — Empowering People to Flexibly Employ Advanced

Information and Communication Technology, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2006, 427-457.

- Marc Fisher, Mingming Cao, Gregg Rothermel, Darren Brown, Curtis Cook, Margaret Burnett, Integrating Automated Test Generation into the WYSIWYT Spreadsheet Testing Methodology, ACM Transactions on Software Engineering and Methodology, 2006.
- Andrew Ko and Brad Myers. Designing the Whyline: A Debugging Interface for Asking Questions About Program Failures. ACM Conference on Human Factors in Computing Systems, Vienna, Austria, April 24-29, 151-58, 2004.
- Andrew Ko. Asking and Answering Questions about the Causes of Software Behaviors, Human-Computer Interaction Institute Technical Report CMU-CS-08-122, May 2008.
- Andhy Koesnandar, Sebastian Elbaum, Gregg Rothermel, Lorin Hochstein, Kathryn Thomasset, and Chris Scaffidi. Using Assertions to Help End-User Programmers Create Dependable Web Macros. Proc. 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2008), Atlanta, GA, November 2008, 124-134.
- Todd Kulesza, Weng-Keen Wong, Simone Stumpf, Stephen Perona, Rachel White, Margaret Burnett, Ian Oberst, Andrew J. Ko. Fixing the Program My Computer Learned: Barriers for End Users, Challenges for the Machine, ACM Conference on Intelligent User Interfaces, Sanibel Island, Florida, pp. 187-196, Feb. 8-11, 2009.
- Greg Little, Tessa Lau, Allen Cypher, James Lin, Eben Haber, Eser Kandogan. Koala: Capture, Share, Automate, and Personalize Business Processes on the Web, ACM Conference on Human Factors in Computing Systems, 2007, 943-946.

- Don Norman. Psychology of Everyday Things, Basic Books, 1988.
- Stephen Oney and Brad Myers. "FireCrystal: Understanding Interactive Behaviors in Dynamic Web Pages". 2009 IEEE Symposium on Visual Languages and Human-Centric Computing, Sept. 20-24, 2009.
- Christopher Scaffidi, Christopher Bogart, Margaret Burnett, Allen Cypher, Brad Myers, and Mary Shaw. Using Traits of Web Macro Scripts to Predict Reuse, Journal of Visual Languages and Computing, 2010.
- Christopher Scaffidi, Brad Myers, and Mary Shaw. Topes: Reusable Abstractions for Validating Data, International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 2008, 1-10.
- Aaron Wilson, Margaret Burnett, Laura Beckwith, Orion Granatir, Ledah Casburn, Curtis Cook, Mike Durham, and Gregg Rothermel, Harnessing Curiosity to Increase Correctness in End-User Programming, ACM Conference on Human Factors in Computing Systems, Ft. Lauderdale, Florida, April 5-10 2003, 305-312.