

Six Challenges in Supporting End-User Debugging

Joseph R. Ruthruff
Department of Computer Science
and Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska, USA
ruthruff@cse.unl.edu

Margaret Burnett
School of Electrical Engineering
and Computer Science
Oregon State University
Corvallis, Oregon, USA
burnett@cs.orst.edu

ABSTRACT

This paper summarizes six challenges in end-user programming that can impact the debugging efforts of end users. These challenges have been derived through our experiences and empirical investigation of interactive fault localization techniques in the spreadsheet paradigm. Our contributions reveal several insights into debugging techniques for end-user programmers, particularly fault localization techniques, that can help guide the direction of future end-user software engineering research.

Categories and Subject Descriptors

D.2.5 [Software Engineering]: Testing and Debugging—*debugging aids, testing tools*; D.2.6 [Software Engineering]: Programming Environments—*interactive environments*; H.4.1 [Information Systems Applications]: Office Automation—*spreadsheets*

General Terms

Experimentation, Verification

Keywords

end-user software engineering, end-user programming, debugging, fault localization

1. INTRODUCTION

End-user programming has become the most common form of programming today: it is estimated that, in 2005 in the United States alone, 55 million end users, compared to only 2.75 million “professional” programmers [4], will be creating software using diverse software environments such as educational simulation builders, web authoring systems, multimedia authoring systems, e-mail filtering rules, CAD systems, and spreadsheet environments. (Although new research [20] has revealed flaws with the methodology used to produce this estimate, this new research indicates that in fact the number of end-user programmers is even higher than reported by Boehm et al. [4].)

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.

First Workshop on End-User Software Engineering (WEUSE I), May 21 2005, Saint Louis, Missouri, USA.

Copyright ACM 1-59593-131-7/05/0005 \$5.00.

Yet despite this trend, evidence suggests that end-user programmers do not have adequate support for their software development efforts. Boehm and Basili [5] observe that 40–50% of the software created by end users contains non-trivial faults. These faults can be serious, costing millions of dollars in some cases (e.g., [10, 14, 16]).

To help provide needed software development support to end users, we have been working on a vision we call *end-user software engineering* [7], which we have prototyped in the spreadsheet paradigm because it is so widespread in practice. The concept of end-user software engineering is a holistic approach to the facets of software development in which end users engage. Its goal is to bring some of the gains from the software engineering community to end-user programming environments — *without* requiring training, knowledge, or even interest in traditional software engineering theory or practices. The aspect of end-user software engineering that is of interest in this paper is debugging.

This paper outlines six challenges that we have encountered in supporting end-user debugging through fault localization techniques. Some of these challenges are brought about due to inherent characteristics of most end-user programming environments, while other challenges are directly tied to the end users themselves. While our challenges are most directly applicable to the area of end-user debugging, all have potential ramifications to research bringing any software development support to end-user programmers.

The remainder of this paper is organized as follows: Section 2 briefly describes the interactive fault localization support in our own end-user software engineering environment; Section 3 discusses the challenges that we have encountered in bringing such debugging support to end users, including the ramifications of these challenges to other researchers; and Section 4 concludes the paper.

2. BACKGROUND

Our debugging support through fault localization is prototyped in the spreadsheet paradigm in conjunction with our “What You See Is What You Test” (WYSIWYT) testing methodology [17], so we briefly describe that methodology first in order to provide a context for our experiences.

2.1 End-User Testing via WYSIWYT

Figure 1 presents an example of WYSIWYT in Forms/3 [6], a spreadsheet language utilizing “free-floating” cells in addition to traditional spreadsheet grids. In WYSIWYT, untested spreadsheet cells that have non-constant formulas are given a red border (light gray in this paper). The borders of such cells remain red until they become more “tested”.

For cells to become more tested, tests must occur. These tests can occur at *any time* — intermingled with formula edits, formula additions, and so on. The process is as follows. Whenever a user notices a correct value, he or she can place a checkmark (✓) in the decision box at the corner of the cell observed to be correct: this *testing decision* completes a successful “test”.

Checkmarks can increase the “testedness” of cells, which is reflected by adding more blue to cell borders (more black in this paper). Further, because a correct value in a cell *c* depends on the correctness of the cells contributing to *c*, these contributing cells participate in *c*’s test. These tests increase testedness according to a test adequacy criterion that has been reported elsewhere [17]. Testedness feedback is also provided at two other granularities: a “percent testedness” indicator provides testedness feedback at the spreadsheet granularity, and colored dataflow arrows can provide feedback at the subexpression granularity (in addition to the cell granularity).

2.2 Adding Interactive Fault Localization

In our prototype, WYSIWYT serves as a springboard for fault localization: instead of noticing that a cell’s value is correct and placing a checkmark, a user might notice that a cell’s value is incorrect (a failure) and place an “X-mark”. In Figure 2, the user notices an incorrect value in *Exam_Avg* — the value is too high — and places an X-mark in the cell’s decision box. X-marks trigger a *fault likelihood* calculation for each cell (with a non-constant formula) that might have contributed to the failure.

Most fault localization support attempts to help programmers locate the causes of failures in two ways: (1) by indicating the ar-

reas that should be searched for faults, thereby *reducing the search space*; and (2) by indicating the areas most likely to contain faults, thereby *prioritizing the sequence of the search* through this space.

In our prototype, fault likelihood, updated for each appropriate cell after any testing decision or formula edit, is represented by coloring the interior of suspect cells in shades of yellow and orange (gray in this paper). This serves our first goal of reducing the user’s search space. As the fault likelihood of a cell increases, the suspect cell is colored in increasingly darker shades of orange (gray). The darkest cells are estimated to be the most likely to contain the fault, and are the best candidates for the user to consider in debugging; this serves our second goal of helping end users prioritize their search. (This approach is generalizable to paradigms other than spreadsheets [19].)

3. SIX CHALLENGES IN END-USER DEBUGGING

Software engineering researchers have long recognized the importance of fault localization strategies (e.g., [1, 9, 11, 13]), investing considerable effort into bringing fault localization techniques to professional programmers. Similar efforts, directed at the needs of end users, could help to improve the quality of the software developed by end-user programmers. However, significant differences exist between professional and end-user software development, and these differences have ramifications for any such efforts by acting as constraints on the types of strategies suitable for end users.

In this section we show that these differences lead to (at least) six challenges in bringing interactive fault localization support to end

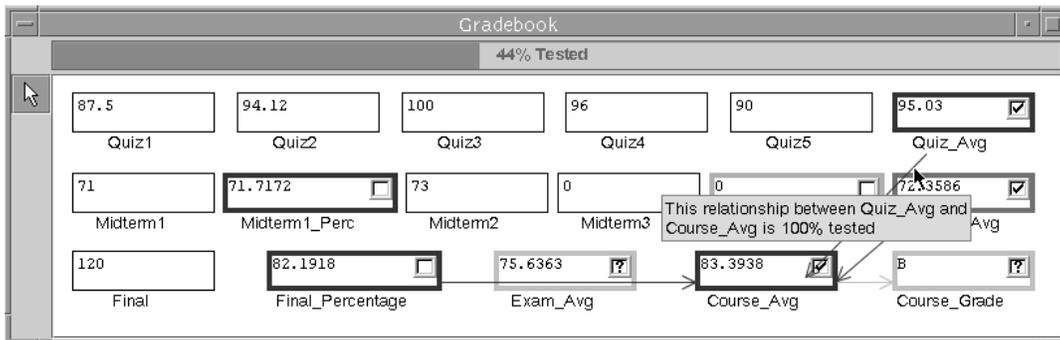


Figure 1: An example of WYSIWYT in Forms/3.

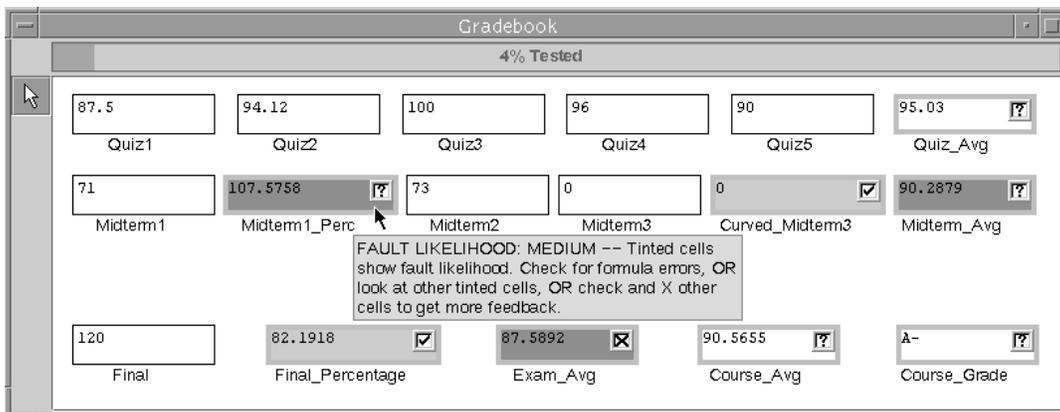


Figure 2: An example of fault localization in the Forms/3 spreadsheet environment.

users in order to support end-user debugging activities. These challenges are most directly applicable to end-user debugging; however, all have potential ramifications to any end-user programming research.

3.1 Lack of Software Engineering Knowledge

A first challenge pertains to knowledge of software engineering theory and practices. Unlike professional programmers, end users rarely have such knowledge, and are unlikely to take the time to acquire it. This impacts fault localization techniques because, traditionally, such techniques often require at least partial knowledge of such theory to (1) properly employ the technique, (2) comprehend the technique’s feedback, or (3) understand why the technique is producing particular fault localization feedback. (As research [2, 8] explains, understanding is critical to trust, which in turn is critical to users actually believing a system’s output and acting on it.)

For example, critical slicing [9] uses mutation-based testing, a strategy of which end-user programmers are unlikely to have any prior knowledge. This lack of knowledge could hinder end users’ ability to understand why critical slicing is producing particular fault localization feedback, which in turn can result in a loss of trust in the feedback.

We believe that researchers should be particularly concerned about whether end users require previous software engineering knowledge to understand why techniques produce particular feedback. End users are unlikely to blindly follow *any* feedback during debugging *unless they are comfortable doing so*, and this comfort level can be hindered by prerequisites placed upon end users by debugging techniques.

A first challenge for researchers, then, is to develop techniques that support debugging without unrealistic prerequisites such as knowledge of software engineering practices.

3.2 Modeless and Interactive Environments

A second challenge pertains to the manner of interaction between the software developer and the programming environment. End-user programming environments are usually modeless and highly interactive: users incrementally experiment with their software and see how the results seem to be working out after every change, using techniques such as the automatic recalculation feature of spreadsheet environments. Most professional programming environments, however, are modal — featuring separate code, compile, link, and execute modes, and separate techniques for tasks such as fault localization. The lack of interaction in these environments has allowed many fault localization techniques to perform a batch processing of information before displaying feedback.

For example, χ slice [1] uses preexisting execution traces from predetermined passed and failed test cases to produce feedback. Unfortunately, if either the suite of test cases or the program source code changes, the testing and execution slicing information would have to be recreated. This process would have to be performed in batch — before any fault localization feedback could be provided.

Because end users tend to interactively debug in parallel with incremental software development, they are likely to expect debugging techniques, such as fault localization support, to be available *at any point* in the development or debugging process, resulting in immediate feedback.

A second challenge for researchers, then, is to develop interactive debugging techniques for end users, as techniques that perform batch processing are at best unsuited, and at worst incompatible, with interactive end-user environments.

3.3 Lack of Organized Testing Infrastructure

A third challenge pertains to the amount of testing information available in professional versus end-user software development environments. End users do not usually have suites of organized test cases, so large bases of testing information, which are commonly used by debugging techniques, are rarely available. However, large bases of testing information are precisely what is required for some debugging techniques to operate effectively.

For example, TARANTULA [11] uses a set of failed and passed tests, and coverage information indicating the program points contributing to each test, to calculate (1) a color representing the participation of every statement (in the source code) in testing, and (2) the technique’s confidence in the correctness of each color. However, in order to calculate accurate colorizations for every statement in the program, the technique not only requires a test for every statement (i.e., a test suite with 100% coverage), but enough tests covering each statement to serve as data points for the confidence of each colorization.

Complicating this type of situation for researchers is the previously described interactive nature of end-user debugging: end users may observe a failure and start the debugging process early — not just after some long batch of tests — at which time the system may have very little information with which to provide feedback. Debugging techniques, therefore, must be able to report feedback at points other than those of “maximal system reasoning potential” — when the technique has the greatest amount of available information with which to provide feedback.

A third challenge for researchers, then, is to develop techniques that do not require large bases of testing information, as such techniques may be inappropriate in end-user programming settings.

3.4 Unreliability of Testing Information

A fourth challenge pertains to a common assumption in software engineering techniques created for professional programmers: that the accuracy of the information provided to the techniques is reliable. Researchers have rarely considered the possibility that information, such as the results of certain test cases in a test suite, may be inaccurate — yet evidence [18, 19] suggests that this dilemma is likely to face researchers seeking to bring debugging techniques such as fault localization support to end users. (Professional programmers err too, of course, but their understanding of testing processes may render them less error-prone than end users.)

For example, in one of our own formative studies [19], which was conducted to investigate end-user debugging strategies, we found that 4.3% of the testing decisions by the end-user participants were incorrect, and that these incorrect decisions affected 60% of the participants’ success rates when searching for faults. A more recent study [18] provided even more disturbing evidence: nearly 25% of the participants’ testing decisions were incorrect, and they affected 74% of the debugging efforts by participants. More evidence can be found in an observational study conducted by Ko and Myers [12], where 29 breakdowns — problems with knowledge, amount or direction of attention, or strategies [15] — occurred during debugging activities.

Unfortunately, many fault localization techniques have no safeguards to provide the technique with robustness in the presence of such unreliable information, and other strategies, such as program dicing [13], simply cannot operate reliably in these settings. It is clearly necessary to provide accurate debugging feedback to users even in these settings so that the technique fulfills its purpose.

A fourth challenge for researchers, then, is to provide end users with debugging techniques that do not require a high degree of reliability in the data in order to provide useful feedback.

3.5 Evaluation of Debugging Feedback

A fifth challenge pertains to evaluating interactive debugging techniques for end users. Many traditional techniques report feedback only at the end of a batch processing of information. This point of maximal system reasoning potential — when the system has its best (and only) chance of producing correct feedback — is therefore the appropriate point to measure these techniques. Given the interactive nature of end-user environments, however, debugging occurs not just at the end of testing, but *throughout* the testing process in incremental stages. Measuring debugging techniques' effectiveness only at the end of testing would thus ignore most of the reporting being done by the interactive technique.

It is therefore necessary for researchers to measure feedback at multiple stages throughout the multiple debugging “sessions” that end users are likely to engage in during their incremental software development. There are many ways in which a researcher could perform these measurements. For example, if the researcher is interested in the effectiveness of one technique's feedback, it may be appropriate to evaluate feedback immediately whenever the feedback changes (e.g., due to a change in the program's logic or the addition, or removal, of testing information). Likewise, if the researcher is interested in comparing the effectiveness of multiple techniques' feedback, then measurements should be taken at all incremental feedback points that are reached often enough to support statistical comparisons.

A fifth challenge for researchers, then, is to incorporate measures for evaluating debugging techniques that consider interactive feedback.

3.6 Attention Investment

A sixth challenge pertains to a question that is not often asked by researchers creating programming techniques for professional programmers: “If we build it, will they come?” A common assumption made by researchers is that software engineers will use a debugging technique because they are already familiar with such techniques, and the benefits of their use are clear to them (assuming that the technique provides accurate feedback). However, the benefits of using such techniques may not be immediately clear to end-user programmers, especially if the end users have little experience using such techniques.

In fact, Blackwell's model of attention investment [3] is one model of user problem-solving behavior predicting that users *will not* want to use *any* programming technique unless the benefits of doing so are clear to them. The model considers the costs, benefits, and risks that users weigh in deciding how to complete a task. For example, if the ultimate goal is to forecast a budget using a spreadsheet, then using a relatively unknown feature such as a fault localization technique has a cost, benefit, and risk. The costs are figuring out when and where to use the technique, and thinking about the resulting feedback. The benefit of finding faults may not be clear after only one use of the technique; in fact, the user may have to expend even more costs (i.e., use the technique more than once) for benefits to become clear. The risks are that going down this path will be a waste of time or worse, will mislead the user into looking for faults in the correct formulas instead of the incorrect ones.

The implications of this model to researchers are considerable, and we will not attempt to enumerate all of them in this paper. Two

implications, however, are (1) the need for mechanisms to convince end users to use debugging techniques *for the very first time* given the perceived costs, benefits, and risks of doing so; and (2) the need for mechanisms to regularly deliver to end users the promised benefits as they use the technique, thus demonstrating benefits that outweigh the costs and risks *if they continue to use the technique*.

In our own prototype, we address this problem using a “Surprise-Explain-Reward” strategy [21]. This strategy (1) entices users to use a technique for the first time by arousing their curiosity about the technique through the element of surprise, and (2) encourages them, through explanations and rewards, to continue using the technique. We believe that researchers need to incorporate similar strategies to address the attention investment considerations that end users will make at any point in their software development.

A sixth challenge for researchers, then, is to account for the attention investment considerations of end users by ensuring that users are enticed to use debugging techniques for the first time, and then continue to use the techniques.

4. CONCLUSIONS

Research is gradually emerging to bring debugging techniques directly to end-user programmers. However, many challenges face researchers attempting to bring this debugging support to this software engineering domain. Using our experiences with fault localization techniques in an end-user software engineering approach, we have outlined six particular challenges that are likely to face researchers attempting to support end-user debugging, and even other software development tasks, in end-user programming environments. We hope that our contributions can help guide the direction of future end-user software engineering research by assisting in the creation of techniques that are better-equipped to support the needs of end-user programmers.

5. ACKNOWLEDGMENTS

This work was supported in part by the EUSES Consortium via NSF grant ITR-0325273. The opinions and conclusions in this paper are those of the authors and do not necessarily represent those of the National Science Foundation.

6. REFERENCES

- [1] H. Agrawal, J. Horgan, S. London, and W. Wong. Fault localization using execution slices and dataflow tests. In *Proceedings of the Sixth IEEE International Symposium on Software Reliability Engineering*, pages 143–151, Toulouse, France, October 1995.
- [2] N. Belkin. Helping people find what they don't know. *Communications of the ACM*, 41(8):58–61, August 2000.
- [3] A. Blackwell. First steps in programming: A rationale for attention investment models. In *Proceedings of the IEEE Symposium on Human-Centric Computing Languages and Environments*, pages 2–10, Arlington, Virginia, USA, September 2002.
- [4] B. Boehm, C. Abts, A. Brown, and S. Chulani. *Software Cost Estimation with COCOMO II*. Prentice Hall PTR, Upper Saddle River, New Jersey, USA, 2000.
- [5] B. Boehm and V. Basili. Software defect reduction Top 10 list. *Computer*, 34(1):135–137, January 2001.
- [6] M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang. Forms/3: A first-order visual language to

- explore the boundaries of the spreadsheet paradigm. *Journal of Functional Programming*, 11(2):155–206, March 2001.
- [7] M. Burnett, C. Cook, and G. Rothermel. End-user software engineering. *Communications of the ACM*, 47(9):53–58, September 2004.
- [8] C. Corritore, B. Kracher, and S. Wiedenbeck. Trust in the online environment. In *HCI International*, volume 1, pages 1548–1552, New Orleans, Louisiana, USA, August 2001.
- [9] R. DeMillo, H. Pan, and E. Spafford. Critical slicing for software fault localization. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 121–134, San Diego, California, USA, January 1996.
- [10] D. Hilzenrath. Finding errors a plus, fannie says; mortgage giant tries to soften effect of \$1 billion in mistakes. *The Washington Post*, October 31, 2003.
- [11] J. Jones, M. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering*, pages 467–477, Orlando, Florida, USA, May 2002.
- [12] A. Ko and B. Myers. Development and evaluation of a model of programming errors. In *Proceedings of the IEEE Symposium on Human-Centric Computing Languages and Environments*, pages 7–14, Auckland, New Zealand, October 2003.
- [13] J. Lyle and M. Weiser. Automatic program bug location by program slicing. In *Proceedings of the 2nd International Conference on Computers and Applications*, pages 877–883, 1987.
- [14] R. Panko. Finding spreadsheet errors: Most spreadsheet errors have design flaws that may lead to long-term miscalculation. *Information Week*, page 100, May 1995.
- [15] J. Reason. *Human Error*. Cambridge University Press, Cambridge, England, 1990.
- [16] G. Robertson. Officials red-faced by \$24m gaffe: Error in contract bid hits bottom line of TransAlta Corp. *Ottawa Citizen*, June 5, 2003.
- [17] G. Rothermel, M. Burnett, L. Li, C. Dupuis, and A. Sheretov. A methodology for testing spreadsheets. *ACM Transactions on Software Engineering and Methodology*, 10(1):110–147, January 2001.
- [18] J. Ruthruff, M. Burnett, and G. Rothermel. An empirical study of fault localization for end-user programmers. In *Proceedings of the 27th International Conference on Software Engineering*, St. Louis, Missouri, USA, May 2005 (to appear).
- [19] J. Ruthruff, S. Prabhakararao, J. Reichwein, C. Cook, E. Creswick, and M. Burnett. Interactive, visual fault localization support for end-user programmers. *Journal of Visual Languages and Computing*, 16(1–2):3–40, February/April 2005.
- [20] C. Scaffidi, M. Shaw, and B. Myers. The ‘55m end-user programmers’ estimate revisited. Technical Report CMU-ISRI-05-100, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, February 2005.
- [21] A. Wilson, M. Burnett, L. Beckwith, O. Granatir, L. Casburn, C. Cook, M. Durham, and G. Rothermel. Harnessing curiosity to increase correctness in end-user programming. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 305–312, Fort Lauderdale, Florida, USA, April 2003.