Software Engineering for End-User Programmers

Dr. Curtis Cook, Shreenivasarao Prabhakararao, Martin Main, Mike Durham, Dr. Margaret Burnett, and Dr. Gregg Rothermel
Oregon State University

It is estimated that by 2005, there will be 55 million end-user programmers compared to 2.75 million professional programmers. Even though end-user programs have the same reliability problems, software engineering research has largely ignored the end-user community. Because end users are different from professional programmers in motivation, background, and interests, the end-user community cannot be served by repacking tools and techniques developed for professional programmers. This article describes our work in developing software engineering devices for spreadsheet developers, one of the largest classes of end-user programmers.

Software engineering research has focused on aiding programmers throughout the software development and maintenance process. However, this focus has been on professional programmers and has largely ignored the sizeable end-user programmer community. It is predicted that by 2005 in the United States alone there will be 55 million end-user programmers compared to 2.75 million professional programmers [1]. The programming systems used by these end users include spreadsheets, web authoring tools, scientific visualization languages, and graphical languages for creating educational simulations.

It should not be surprising that a high percentage of end-user programs contain errors that can have significant economic impact. For example, a Texas oil and gas company lost millions of dollars in an acquisition deal because of spreadsheet errors [2]. In error data collected from field audit reports of real-world spreadsheets, Panko [2] reported that 20 percent to 40 percent of the spreadsheets contained errors, and errors were as high as 90 percent in some of the financial models reviewed. In empirical studies involving both experienced and inexperienced spreadsheet developers, he found that over 60 percent of the spreadsheets created by the participants contained errors. Compounding the reliability problem is the unwarranted confidence of end users that their spreadsheets do not contain errors [3].

What is surprising is that software engineering research has paid little attention to spreadsheet programmers and other end-user programmers. Our research has focused on the spreadsheet paradigm, the most widely used and studied end-user programming paradigm. Our intent is to bring some of the advances in software engineering research to these end users without requiring that they first learn the underlying software engineering theory and principles. We call this concept *end-user software engineering*.

In this article, we first point out some of the unique characteristics of spreadsheet end users. This serves two purposes. First, it shows that traditional software engineering techniques must be modified for end users; second, it provides a context for understanding the methodologies and tools we have developed as part of end-user software engineering. These include the *What You See Is What You Test* (WYSIWYT) methodology that provides visual feedback to end users about how much of their spreadsheets have been tested (e.g., degree of testing of their spreadsheets), a *Help Me Test* device that automatically generates test cases, and finally an approach for supporting assertions in end-user software. We present the devices and briefly describe a series of empirical studies that validate our efforts and conclude with a suggested follow-up.

End-User Characteristics

The most obvious difference between professional programmers and end-user programmers is programming experience and background. A high percentage of spreadsheet programmers have little or no programming experience. They view a spreadsheet as a tool to help them solve their problems and regard computers "as a means to an end rather than objects of intrinsic interest" [4].

Hence in adapting a software engineering technique for spreadsheet end users, it is unreasonable to expect them to have the time or interest to learn the underlying theory. Spreadsheet end users are accustomed to working in an incremental fashion in a highly interactive and visual environment with immediate feedback. Further, spreadsheets are usually created in an ad-hoc manner without a clear design plan or formal specification [5]. Even though the spreadsheet creator has a mental model of how it should work, most often it is not explicitly specified, and the actual spreadsheet is only an approximation of the model. Thus any technique developed should require a minimum of training, not assume a programming background or formal problem specifications, and be compatible with the incremental working style.

What We Have Done

Our work has been guided by the above end-user characteristics. We have prototyped our methodology and tools in the spreadsheet research language Forms/3 [6] because we have access to the implementation of Forms/3, and thus we can implement and experiment within that environment. Further, by working with Forms/3 we can investigate not only language features common in commercial spreadsheet languages but also advanced language features found in research spreadsheet languages.

In Forms/3, as in other spreadsheet languages, spreadsheets are a collection of cells and each cell's value is defined by the cell's formula. A programmer receives immediate feedback about a cell's value after the cell formula is entered. Figure 1 shows a Forms/3 spreadsheet that computes student grades based on quiz and extra credit scores. Three differences between Forms/3 and commercial spreadsheets such as Excel are that cells can have meaningful names, more than one cell formula can be displayed at a time, and the cells do not have to be laid out in a grid and can be positioned anywhere on the screen. None of these differences are required for or affect the end-user software engineering devices presented here.

The WYSIWYT Methodology

The WYSIWYT [7] methodology gives end users visual feedback about the degree of testing of individual cells and the entire spreadsheet. The WYSIWYT methodology is based on definition-use associations (du-associations) in a spreadsheet that link a defining expression in a

cell formula (definition) with expressions in other cell formulas that reference (use) the defined cell. See [7] for more details.

The WYSIWYT methodology provides visual feedback about the extent to which du-associations have been covered by tests by means of cell border colors. A percent-tested indicator at the upper right of the spreadsheet gives the percent of du-associations that have been covered. A red cell border (*Total_Score*, *LetterGrade, ErrorsExist?*) means none of the du-associations for the cell have been covered. A blue border (*avg*) means all of the du-associations have been covered, and shades of purple (*EC_Award*) mean some of the du-associations have been covered. Via tool tips, the end users can learn that a red cell border means that a cell is untested, blue means fully tested, and shades of purple mean partially tested.

An end user can also display arrows that indicate dependencies (du-associations) between cells and cell formulas. The arrows follow the same color scheme as cell borders. The arrows reveal the degree of testing at the du-association level, but they are optional; users do not have to think about testing at the du-association level unless they prefer it. Arrows for cell *ErrorsExist?* displayed in Figure 1 indicate a partial degree of testing. Since the formula for this cell is displayed, the arrows point to the cell references in the formula and from the formula to uses of the cell.

The WYSIWYT visual devices keep the user continually informed about the degree of testing of the spreadsheet, draw attention to untested parts of the evolving spreadsheet, and suggest where testing will cover new situations. As cell formulas are modified or new cells added, du-associations are added, deleted, or modified; these changes to du-associations are immediately reflected in the cell border and arrow colors and the percent-tested indicator (upper right indicator).

Help Me Test

As described to this point, the WYSIWYT relies solely on the skill of the end user to develop test cases for his or her spreadsheets. Sometimes the end user will know from the WYSIWYT feedback that a spreadsheet is not fully tested, but will be unable to find a set of inputs for a new situation. To aid end users in finding appropriate input values for these situations, we have integrated a Help Me Test device that the user can invoke to find a test case. When Help Me Test succeeds, it stops and highlights the input cells that have been changed and the cells that now cover new situations. Figure 2 shows only the output in the Help Me Test window when invoked for cell *EC_Award* in the Grades spreadsheet and not the cells in the spreadsheet that have been changed. The user can then make testing decisions about some or all of these cells. A user can invoke Help Me Test for the entire spreadsheet, a single cell, or a particular arrow.

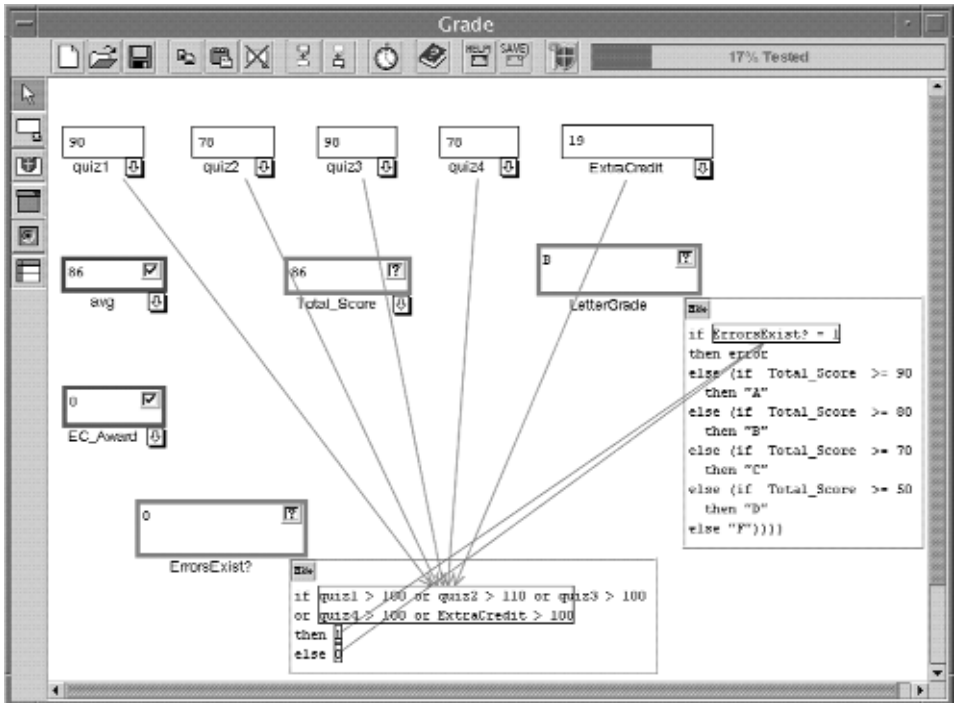


Figure 1: *A Forms/3 Grades Spreadsheet*

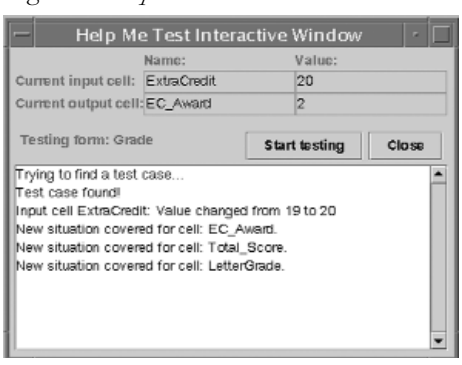
Assertions

Assertions – statements about the properties of a program – are used by professional programmers to prove their programs are correct and to help detect errors. When creating a spreadsheet, the user has a mental model of properties it should have and how it should operate. One approximation of this model is the formulas they enter, but unfortunately these formulas may contain inconsistencies or faults. These formulas, however, are only one representation of the user's model of the problem and its solution: They contain information on how to generate the desired result, but do not provide ways for the user to communicate other properties. Traditionally, assertions in the form of preconditions, post conditions, and invariants have fulfilled this need for professional programmers, providing a method of making explicit the properties the programmers expect of their program logic, providing a reason about integrity of their logic and providing a way to catch exceptions.

While these forms of assertions may aid professional programmers, their syntax and Boolean expressions are inappropriate for most end users. Our approach attempts to provide the same advantages to end-user programmers, but is different from traditional approaches in that ours is a component of our integrated set of software engineering features specifically designed for end users. As part of the incremental end-user spreadsheet development, the user can enter a few assertions and see the effects. Our assertions look like simple ranges, but because they include open and closed ranges, *and*, *or*, and references to cells, this syntax allows a fairly powerful set of assertion types [8].

There are two types of assertions: user-entered and system-generated. User-entered assertions are those explicitly entered by the user while the generated assertions result from propagating assertions through formulas in the direction of dataflow using logic and interval arithmetic. User-entered and system-generated assertions are stacked on the top of the cells in Figure 3 (see Page 22). The

Figure 2: *Help Me Test Window*



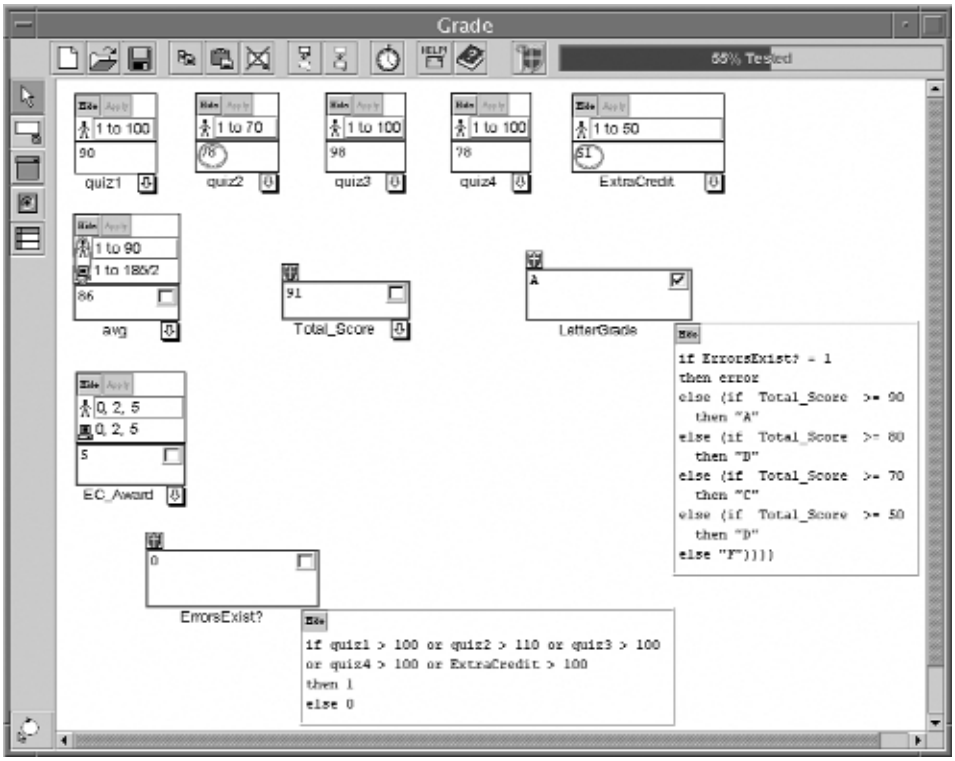


Figure 3: *User-Entered and System-Generated Assertions in Grades Spreadsheet*

top row of cells is simply input cells with constant values as their formulas. Cell *ExtraCredit* has a user-entered assertion (stick figure icon) from one to 50 while the user-entered and system-generated (computer icon) assertions for cell *EC_Award* are the three integer values zero, two, or five. Assertions help users detect errors through assertion conflicts (user and system assertions disagree) and value violations (cell value outside of range). To draw the user's attention to possible errors, red ovals circle assertion conflicts and value violations. Cells *quiz2* and *ExtraCredit* have value violations and the cell *avg* has an assertion conflict in Figure 3.

To introduce users to the idea of entering assertions, Help Me Test provides suggested assertions on some cells that do not yet have them. When users run Help Me Test to get new test inputs, our empirical work showed that these suggested assertions were effective in inducing them to use assertions while debugging [9].

Commercial spreadsheets such as Microsoft Excel have a data validation feature that bears a surface similarity to assertions in our environment. However, these commercial spreadsheets do not propagate assertions, do not automatically display assertions, and do not update the display of assertion violations when changes are made. In short, their assertions are data entry checks, whereas ours form an ever-present reasoning mechanism that watches over all the cells at all times.

Validation

We have used empirical studies both to demonstrate that our methodology and tools do indeed aid end users in testing, debugging, and maintaining their spreadsheets and to gain a better understanding of how end users work and how our devices help them. In nearly all of these studies we have used sophomore and junior business majors as subjects.

Two controlled experiments [10, 11] showed that subjects using the WYSIWYT methodology tested significantly better (higher coverage, fewer redundant tests) and were significantly more successful in a maintenance task (more correct modifications, more testing) than subjects without the WYSIWYT methodology. In a debugging study [12], we found that WYSIWYT subjects using assertions found significantly more bugs and found them faster than WYSIWYT subjects without assertions. A follow-up study [9] showed that end users elected to enter assertions of the type described in this article, and did so quite accurately.

We have also conducted several think-aloud studies during which we observe subject behavior and record subject verbalizations as they perform the experimental task. These studies provide insight into their thought processes and strategies. Our think-aloud studies have found that end-users understood assertions and could effectively use them in a maintenance task [8], and that end-users with WYSIWYT and Help Me Test were more effective and more efficient in a modification task than end-users with only WYSIWYT [8]. In all of our experiments, the subjects using our end-user software engineering devices showed a more appropriate level of confidence about whether their spreadsheets contained errors.

Conclusions

Software engineering research has largely ignored the end-user community in spite of the fact that there will soon be 20 times as many end-user programmers as professional programmers. Yet, it should not be a surprise that end-user programs have the same correctness problems. Because end users are different from professional programmers in background, motivation, and interests, the end-user community cannot be served by simply repackaging techniques and tools developed for professional programmers. Instead, the methodologies, tools, and techniques developed for end users must take these differences into account.

In this article we have described our approach in developing software engineering devices for spreadsheet users, which has been met with considerable success. We advocate that spreadsheet languages contain some of the devices we have developed, and we believe our approach holds promise for those developing tools and techniques for other types of end-user software. We welcome the opportunity to collaborate with others interested in this work. If you are interested in either theoretical or practical follow-up, please contact author Dr. Curtis Cook.◆

References

1. Boehm, B., E. Horowitz, R. Madachy, D. Riefer, B. Clark, B. Steece, A.W. Brown, S. Chulani, and C. Abts. Software Cost Estimation With COCOMO II. Englewood Cliffs, N.J.: Prentice-Hall, 2000.
2. Panko, R. "What We Know About Spreadsheet Errors." Journal of End User Computing Spring 1998: 15-21.
3. Brown, P., and J. Gould. "Experimental Study of People Creating Spreadsheets." ACM Transactions on Office Information Systems 5.3 (July 1987): 258-272.
4. Nardi, B., and J. Miller. "Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development." Int. J. Man-Machine

Studies 34 (1991): 161-184.
5. Ronen, B., R. Palley, and H. Lucas. "Spreadsheet Analysis and Design." Communications of the ACM 32.1 (Jan. 1989): 84-93.
6. Burnett, M., 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 (Mar. 2001): 155-206.
7. Rothermel, G., L. Li, C. DuPuis, and M. Burnett. What You See Is What You Test: A Methodology for Testing Form-Based Visual Programs. Proc. of 20th International Conference on Software Engineering. Kyoto, Japan. Apr. 1998: 198-207 <http://cs.oregon state.edu/~burnett/ITR2000>.
8. Rothermel, K., C. Cook, M. Burnett, J. Schonfeld, T.R.G. Green, and G. Rothermel. WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation. Proc. of the 22nd International Conference on Software Engineering. Kyoto, Japan. June 2000: 230-239.
9. Wallace, C., C. Cook, J. Summet, and M. Burnett. Assertions in End-User Software Engineering: A Think-Aloud Study. Proc. of IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC 2002). Arlington, VA, Sept. 3-6, 2002: 63-65.
10. Wilson, A., M. Burnett, L. Beckwith, O. Granatir, L. Casburn, C. Cook, M. Durham, and G. Rothermel. Harnessing Curiosity to Increase Correctness in End-User Programming. Proc. of ACM Conference on Human Factors in Computing Systems. Ft. Lauderdale, FL, Apr. 5-10, 2003.
11. Krishna, V., C. Cook, D. Keller, J. Cantrell, C. Wallace, M. Burnett, and G. Rothermel. Incorporating Incremental Validations and Impact Analysis Into Spreadsheet Maintenance: An Empirical Study. Proc. 25th IEEE International Conference on Software Maintenance. Florence, Italy, Nov. 2001: 72-78.
12. Burnett, M., C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace. End-User Software Engineering With Assertions in the Spreadsheet Paradigm. Proc. of the International Conference on Software Engineering. Portland, OR, May 2003.

About the Authors

Curtis Cook, Ph.D., is professor of computer science at Oregon State University. He has more than 20 years of research and experience in software complexity metrics, program understanding, and software quality. Cook is a member of the editorial board of the *Software Quality Journal*. He has a doctorate in computer science from the University of Iowa.

Computer Science Department
Oregon State University
Corvallis, OR 97331-3202
Phone: (541) 737-5564
Fax: (541) 737-3014
E-mail: cook@cs.orst.edu

Martin Main is a senior and undergraduate research assistant in Computer Science at Oregon State University. Prior to returning to college studies, Martin was a recording engineer assisting on major label recordings. The current shift in that industry into computer usage sparked Main's interest in how computers are used by society as a whole.

E-mail: mainma@cs.orst.edu

Margaret Burnett, Ph.D., is an associate professor in the Computer Science Department at Oregon State University. She previously worked for several years in industry. Her research interests are where programming languages, human-computer interaction, and software engineering meet, namely in visual programming languages and in how programming language and software engineering research can be applied to support end-user programming. Burnett received the National Science Foundation's Young Investigator Award for her work in visual programming languages. She has a doctorate in computer science from the University of Kansas.

E-mail: burnett@cs.orst.edu

Shreenivasarao Prabhakararao is a graduate student in the Computer Science Department at Oregon State University. He works as a research assistant in the Forms/3 research group. His research interests are software engineering, software testing, and empirical studies. Prabhakararao has a bachelor's degree from Osmania University, India, and a master's degree in computer applications from Birla Institute of Technology, Mesra, India.

E-mail: prabhash@cs.orst.edu

Mike Durham is a senior and undergraduate research assistant with Oregon State University's Computer Science department. He contributed to a recently published paper on the curiosity and behavior of end users. Durham's research interests include human-computer interaction, end-user software engineering, and psychology.

E-mail: durhammi@cs.orst.edu

Gregg Rothermel, Ph.D., is an associate professor in the Computer Science Department at Oregon State University. His research interests include software engineering and program analysis, with an emphasis on software maintenance and testing. Rothermel is a recipient of the National Science Foundation's Faculty Early Career Development Award and is associate editor for *IEEE Transactions on Software Engineering*. He has a doctorate in computer science from Clemson University.

E-mail: grother@cs.orst.edu