

Assertions in End-User Software Engineering: A Think-Aloud Study

Christine Wallace, Curtis Cook, Jay Summet, Margaret Burnett
Computer Science Department, Oregon State University
{wallacch, cook, summet, burnett}@cs.orst.edu

ABSTRACT

There has been little research on end-user program development beyond the programming phase. Devising ways to address additional phases may be critical, because research shows that over one-half of the programs written by end users, at least in the widely used spreadsheet paradigm, contain errors. In this paper, we investigate whether providing end users with integrated support for requirement specifications in the form of assertions can help them reason about, recognize, and remove errors in their spreadsheets. Our think-aloud study revealed that end users can indeed use assertions to find and correct errors as they work with their spreadsheets, and also revealed some surprising tendencies and biases about testing.

1. INTRODUCTION

End-user programming has become prevalent in our society. The number of end-user programmers in the United States alone is expected to reach 55 million by 2005, as compared to only 2.75 million professional programmers [2]. Despite its accessibility to wide audiences, end-user programming is still programming, and programming is subject to errors. Research about spreadsheet errors reveals that this problem is widespread. Panko's survey [3] of spreadsheet error research reports that 91% of the surveyed field audits since 1997 (54 spreadsheets) contained errors. Further, users in these and other studies demonstrated overconfidence as to the correctness of their spreadsheets [5].

To help address this problem, we have been working on a concept we call end-user software engineering. Since the spreadsheet paradigm is so common among end users, we are prototyping our research in that paradigm. Our goal is to create an end-user environment that seamlessly supports non-coding phases of software development.

That the software engineering research community has virtually ignored end-user programming is surprising given the huge number of end-user programmers and their penchant for errors. One of the few recent efforts toward supporting non-coding phases of software development is

the highly interactive WYSIWYT ("What You See Is What You Test") methodology we developed for testing and debugging spreadsheets [4]. The WYSIWYT methodology uses colored borders to communicate testedness of individual cells to the user. See Figure 1. Red means untested, shades of purple mean partially tested, and blue means fully tested. There is also a "percent tested" indicator at the top right of the spreadsheet that displays information about the spreadsheet as a whole. These devices are always kept up-to-date. Thus, when a formula is entered or modified, the cell border turns red because the cell is untested; cells that reference the modified cell also turn red.

In this paper, we investigate whether providing end users with integrated support for requirement specifications in the form of assertions, called "guards," can help them reason about, recognize, and remove errors in their spreadsheets. The goal of our guard mechanism is to capture information from the end user about specifications and to make it explicit. Once captured, the system and the user can together use the information to find and correct errors. Eventually, we plan a number of ways to capture requirement specifications (guards) from user interactions with the environment. However, before proceeding with specific strategies for capturing assertions, it is important to gain more understanding into fundamental issues such as whether end users can understand or reason about formal requirements and whether doing so helps them find errors. To learn more about such issues, we conducted a think-aloud study.

2. BACKGROUND

Traditionally, assertions in the form of preconditions, post-conditions, and invariants, have provided a method for professional programmers to reason about the integrity of their logic, document their assumptions, and catch exceptions. In the realm of end-user programming, there has been little attention directed toward increasing support for achieving correctness in end-user programs. Microsoft Excel, a popular commercial spreadsheet application, has a simple device termed "data validation" that can be thought of as a version of assertions. Excel's device is not automatically updated and the assertions do not propagate

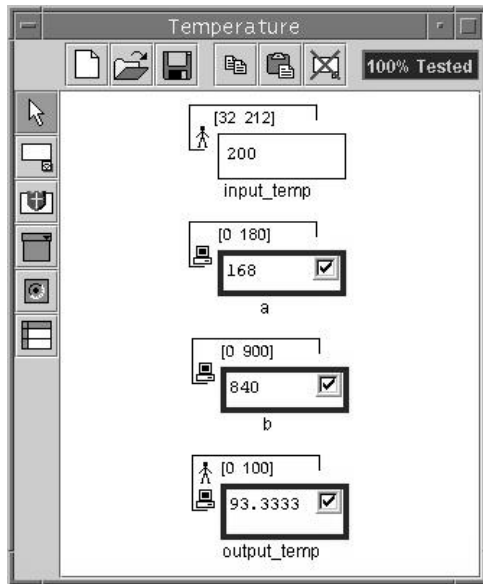


Figure 1: Temperature Spreadsheet with Guards

to output cells. However, our system continuously updates and propagates assertions to output cells by reasoning about their interactions with the formulas.

Assertions in our guard mechanism allow users to communicate aspects of their mental model not represented by formulas. Thus, the system gains the ability to crosscheck the user's mental model as represented by the guards against the way that it is represented by the actual formulas.

Users place guards on cells to ensure specific properties, e.g. "this cell will contain a number between zero and one hundred." Guards appear above the cells. See Figure 1. A computer icon in a guard indicates a propagated guard and a stick figure a user-entered guard. A value violation occurs when cell values are out of a range. A guard conflict occurs when the user and propagated guards disagree. Both violations and conflicts are indicated automatically by red ovals.

3. EXPERIMENT

We designed our study to reveal information about the following primary research questions:

- RQ1: Can end users understand guard propagation?
- RQ2: Are end users distracted from their current tasks by the guard conflicts or value violations [1]?
- RQ3: Do guards help end users modify their spreadsheets correctly?
- RQ4: How do guards affect end users' testing?

Subjects for our think-aloud study were ten sophomore-level Business majors, with little or no formal programming training. We evenly split them into a Control group (without guards) and a Guard group. We chose the think-aloud analysis method because it lent itself to the heart of our research questions. We chose

maintenance over creation tasks because our research questions deal with how end users reason about existing guards. We selected range assertions for our guards because end users were most likely to encounter and understand ranges. See [6] for a more complete description of the experiment and results.

We conducted our study on a Sun workstation running Forms/3, our research spreadsheet language. After completing a background questionnaire, the subjects received a hands-on tutorial that gave them a basic Forms/3 and WYSIWYT skill set. The Guard group received additional training on guard feedback and editing. The tutorial was paced for each individual subject and concluded only when the skills had been mastered.

The subjects performed two spreadsheet modification tasks. In the first task, the original spreadsheet converted a temperature between 32 and 212 in degrees Fahrenheit to Celsius. The task for both groups was to modify the existing spreadsheet so that it converted a temperature between 0 and 100 degrees Celsius to Fahrenheit.

The second task (Grades) involved modifying a spreadsheet that calculated a student's course grade. As given, the weighting was implemented by giving all assignments and test scores different point values. For example: hw1, hw2, and midterm had maximum point values of 10, 20, and 15 respectively. The task was to implement a scheme in which all assignments and tests have the same point value of 100 and the formulas take care of the weighting, i.e., $0.1*hw1 + 0.2*hw2 + \dots$

For each task, the subjects received a problem description handout and a fully tested (all cells were blue), working spreadsheet. Both spreadsheet tasks were designed so that value and guard conflicts would arise no matter what order the subjects did the modification.

The subjects worked individually with the examiner and were encouraged to think aloud. We made audio recordings and electronic transcripts of their keystrokes as they performed the two spreadsheet maintenance tasks. The examiner used prompting questions such as "What is your strategy?" and "What are you thinking?" to remind the subjects to think aloud. The examiner answered questions about skills learned in the tutorial, but for strategy questions, the subjects were referred to the written problem description.

After the tasks, all subjects answered post session questions about the correctness of their spreadsheets, strategies used, and test case selection. The Guard subjects answered additional questions that tested their understanding of guards, value violations, and guard conflicts.

4. RESULTS

We investigated the answers to our research questions in two ways: by observations of subjects during the modification tasks and through post session questions.

This data demonstrated that users with guards were able to understand guard propagation, and were rarely drawn off task by value violations and/or guard conflicts. As an example of understanding guard propagation, subject G2, after editing cell *b*'s formula, saw the propagated guard on cell *b* change to [32 212] and then remarked:

"And now we have found that the guard value has changed on *b* from 32 to 212, representing the possible range we could get for Fahrenheit - which is pretty ingenious for this program. I like that."

One way we addressed RQ2, was to compare the orders in which subtasks were performed by the Control group with those of the Guard group. We observed ten instances of guard conflicts: five Guard subjects, two tasks each. In all cases, the guard conflict arose immediately after the first edit of the spreadsheet. Only one of these instances gave evidence of distraction; in the other nine instances of guard conflicts, the subjects continued working in a left-to-right, top-down (dataflow) fashion, editing intermediate formulas before working on the cell that contained the conflict. In the Control group, as well, the predominant editing order was with the dataflow.

With regard to RQ3, we found that the assertions brought errors to users' attention, and that the users found the crosschecking provided by the assertions reassuring. Among users who made errors, the guard subjects were more successful in finding and correcting the errors as a direct result of the guards. For instance in the Temperature conversion task, C1, a control subject, turned in an incorrect spreadsheet because she checked the mathematics in a formula instead of checking against the specifications. In doing so she failed to realize that the output for 100 degrees Celsius was incorrect. Two guard subjects created spreadsheet similar to C1's but they noticed the guard conflict between the range they entered and Forms/3's propagated range. The quote below from one of the guard subjects illustrates how the conflict helped him find the error.

"Now I have a problem down here because the ranges disagree. So I say it should go from 0 to 100. And <the system> says it should go from something else totally different. So that means that somewhere my formula is wrong. But, I know overall that it should be between 32 and 212, because it is supposed to be in Fahrenheit. So I'll put that in."

Finally, the guards did not cause users to do less testing. There was no statistical difference between the two groups in the amount of testing.

In testing their modified programs, the Guard and Control subjects exhibited several behaviors that were a surprise to us. One such surprise was the use of what might be called a "local testing" strategy, in which subjects tested a cell using only information in that cell.

Checking the math is an instance of local testing. Most intermediate cells were tested by local testing. Predictive testing was used, primarily for testing output cells, on less than half of the spreadsheets.

Another surprise was that subjects chose to avoid "easy" inputs, because they did not view these inputs as being realistic tests or they felt the computer could do these by default. Easy inputs would have allowed them to use their domain knowledge to predict and easily determine whether the program output was correct. Since their realistic (non-easy) inputs did not lead to predictable outputs, subjects then resorted to local testing. Many of the subjects also avoided extreme values (maximums and minimums) during their testing. In fact, two subjects explicitly expressed a bias against extreme value testing.

Interestingly, the guards partially compensated for some of these problematic behaviors, and this too was unanticipated. That is, Guard subjects who did not choose to test extreme values still got the same feedback as subjects who did choose them, because the system-generated guards reflected the result of pushing the original range of inputs through the formulas. Further, the guards seemed to serve as reminders to some of the subjects to think about the big picture. For instance, one subject explicitly described using predictive testing and the guards together. This suggests that the tight integration of guards and testing is better than either feedback alone.

5. REFERENCES

- [1] Blackwell, A. and Green, T. R. G., "Investment of Attention as an Analytic Approach to Cognitive Dimensions" In T. R. G. Green, R. H. Abdullah & P. Brna (Eds.) *Collected Papers of the 11th Annual Workshop of the Psychology of Programming Interest Group (PPIG-11)*, 1999, 24-35.
- [2] Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. J. and Steece B. *Software Cost Estimation with COCOMO II*, Prentice Hall PTR, Upper Saddle River, NJ, 2000.
- [3] Panko, R., "Spreadsheet Errors: What We Know. What We Think We Can Do", *Proceedings of the Spreadsheet Risk Symposium European Spreadsheet Risks Interest Group (EuSprIG)* (July 2000).
- [4] Rothermel, G., Burnett, M., Li, L., DuPuis, C., and Sheretov, A., "A Methodology for Testing Spreadsheets", *ACM Transactions on Software Engineering and Methodology*, (January 2001), 110-147.
- [5] Wilcox, E., Atwood, J., Burnett, M., Cadiz, J., and Cook, C., "Does continuous visual feedback aid debugging in direct-manipulation programming systems", in *ACM CHI'97*, (Mar. 1997), 258-265.
- [6] Wallace, C. and Cook, C., "End-User Assertions in Forms/3: An Empirical Study", TR 01-60-11, Computer Science Department, Oregon State University, Sept. 2001.