

Strategies and Behaviors of End-User Programmers with Interactive Fault Localization

S. Prabhakararao, C. Cook, J. Ruthruff, E. Creswick, M. Main, M. Durham, and M. Burnett
Oregon State University, Corvallis, Oregon, 97331, USA
{prabhash, cook, ruthruff, creswick, mainma, durhammi, burnett}@cs.orst.edu

Abstract

End-user programmers are writing an unprecedented number of programs, due in large part to the significant effort put forth to bring programming power to end users. Unfortunately, this effort has not been supplemented by a comparable effort to increase the correctness of these often faulty programs. To address this need, we have been working towards bringing fault localization techniques to end users. In order to understand how end users are affected by and interact with such techniques, we conducted a think-aloud study, examining the interactive, human-centric ties between end-user debugging and a fault localization technique. Our results provide insights into the contributions such techniques can make to an interactive end-user debugging process.

1. Introduction

Recent years have seen the explosive growth of end-user programming. In fact, by the year 2005, it is estimated that there will be approximately 55 million end-user programmers in the U.S. alone, as compared to an estimated 2.75 million professional programmers [3]. Real-world examples of end-user programming environments include educational simulation builders, web authoring systems, multimedia authoring systems, e-mail filtering rule systems, CAD systems, and spreadsheets.

But, how reliable are the programs end users write using such systems? One of the most widely used real-world end-user programming paradigms is the spreadsheet. Despite its perceived simplicity, evidence from this paradigm reveals that end-user programs often contain an alarming number of faults [15]. (Following standard terminology, in this paper we use the term *failure* to mean an incorrect output value given the inputs, and the term *fault* to mean the incorrect part of the program (formula) that caused the failure.)

To help solve this reliability problem, we have been working on a vision we call “end-user software engineering,” prototyping our ideas 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 end users to have training, or even interest, in traditional software engineering concepts or techniques.

As part of this work, we previously introduced the “What You See Is What You Test” (WYSIWYT) testing methodology for spreadsheets [17]. We are working to tightly integrate WYSIWYT with visual fault localization techniques in an effort to explicitly support debugging by end users [16, 18].

In this paper, we consider how such techniques affect and interact with end-user programmers’ debugging efforts. To explore this issue, we conducted a think-aloud study to investigate the following research questions:

- RQ1:** How much perceived value do end users see in the interactive fault localization feedback over time?
- RQ2:** How thoroughly do end users understand the interactive fault localization feedback?
- RQ3:** What debugging strategies do end users use to find faults?
- RQ4:** How does fault localization feedback influence an end user’s interactive debugging strategy?
- RQ5:** How do wrong testing decisions affect fault localization feedback?

This paper reports the results of our study.

2. Background: WYSIWYT

Our approach to fault localization is integrated into the WYSIWYT testing methodology. All testing-related communication is incorporated into the spreadsheet itself, and is performed through visualization devices. WYSIWYT has been implemented in Forms/3 [5], a research spreadsheet language that allows “free-floating” cells in addition to cells in grids.

Figure 1 presents an example of WYSIWYT in Forms/3. In WYSIWYT, untested spreadsheet output (i.e. non-constant) cells are given a red border (light gray in this paper), indicating that the cell is untested. For example, the EC_Award cell has never been tested; hence, its border is red (light gray). The borders of such cells remain red until they become more “tested”.

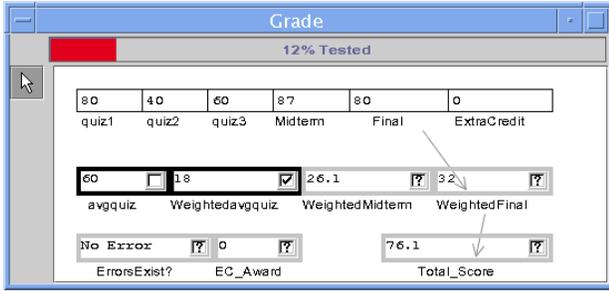


Figure 1. An example of WYSIWYT in the Forms/3 spreadsheet language.

Whenever a user notices that her real or experimental input values are resulting in correct outputs, she can place a checkmark (✓) in the decision box at the corner of the cells she observes to be correct: this constitutes a successful “test”. These checkmarks increase the “testedness” of a cell, which is reflected by adding more blue to the cell’s border (more black in this paper). For example, the Weightedavgquiz cell has been given a checkmark, which is enough to fully test this cell, thereby changing its border from red to blue (light gray to black). 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. Consequently, in this example the border of cell avgquiz also turns blue (black).

Although users may not realize it, the “testedness” colors that result from placing checkmarks reflect the use of a dataflow test adequacy criterion that measures the interrelationships in the source code that have been covered by the users’ tests. A cell is fully tested if all its interrelationships have been covered; if only some have been covered the cell is partially tested. These partially tested cells would have borders in varying shades of purple (shades of gray). (Details of the test adequacy criterion are given in [17].)

To facilitate understanding the structure of a spreadsheet, Forms/3 also allows the user to pop up dataflow arrows between cells—in Figure 1 the user has triggered the dataflow arrows for the WeightedFinal cell—and even between subexpressions within cell formulas (not shown). The dataflow arrows use the same testedness colorization as the cell borders do. Dataflow arrows for any cell can be displayed and hidden at will by the user.

3. Related Work

Most fault localization research has been based on slicing and dicing techniques. Tip surveyed these techniques in [21]. In general a program slice relative to variable V at program point P is the set of all statements in the program that affect the value of V at P . If the value of V is incorrect at point P for a given test case, then, under certain assumptions, it can be guaranteed that the fault that caused the incorrect value is in the slice. A dice is similar to a slice [7]. The difference is that it utilizes

information about correct values of V at other program points to further reduce the set of statements that could be responsible for the incorrect value of V .

Agrawal et al. have built upon these techniques for traditional languages in xSlice [20]. xSlice is based upon displaying dices of the program relative to one failing test and a set of passing tests. Tarantula utilizes testing information from all passing and failing tests to visually highlight possible locations of faults [11]. Both these techniques are targeted at the professional programmer and return results after running a batch of tests. Besides being explicitly targeted at end users, our methods differ these approaches because our methods are interactive and incremental.

There are several interactive visual approaches related to program comprehension for debugging purposes. ZStep [12] provides visualizations of the correspondences between static program code and dynamic program execution. ZStep also offers a navigable visualization execution history that is similar to features found in some visual programming languages such as Kid-Sim/Cocoa/Stagecast [9] and Forms/3 [5]. S2 [19] provides a visual auditing feature in Excel 7.0: similar groups of cells are recognized and shaded based upon formula similarity, and are then connected with arrows to show dataflow. This technique builds upon the Arrow Tool, a dataflow visualization device proposed by Davis [8].

Work aimed particularly at aiding end-user programmers with debugging and other software engineering tasks is beginning to emerge. Myers and Ko recently proposed research in assisting users in the construction and debugging of code for event-based languages [14]. Igarashi et al. present tools to aid spreadsheet users in dataflow visualization and editing tasks [10]. Carr proposes reMIND+ [6], a visual end-user programming language with support for reusable code and type checking. reMIND+ also provides a hierarchical flow diagram for increased program understanding. Outlier finding [13] is a method of using statistical analysis and interactive techniques to direct end-user programmers’ attention to potentially problematic areas during automation tasks. Miller and Myers use visual cues to indicate abnormal situations while performing search and replace or simultaneous editing tasks. Because not all outliers are incorrect, the approach uses a heuristic to determine the outliers to highlight for the user. Finally, the assertions approach in Forms/3 has been shown empirically to help end-user programmers correct errors in spreadsheets [4, 22].

4. Study

To obtain the qualitative information necessary to investigate the research questions enumerated earlier (Section 1), we conducted a think-aloud study, using ten end users as subjects. A think-aloud study allows subjects to verbalize the reasoning for their actions. Traditional

experiments based on statistical methods do not provide the qualitative information we sought for this work. For example, a traditional experiment cannot explain user behaviors or reactions to fault localization feedback, or provide insights into the cognitive thought process of a user; rather, such experiments provide only indirect clues about the human-centric issues we sought to investigate.

4.1 The Study’s Fault Localization Technique

The fundamental assumption behind the WYSIWYT methodology is that as a user incrementally develops a spreadsheet, he or she is also incrementally testing that spreadsheet. At any time, users can place a checkmark (✓) in the decision box of a cell with a correct value. With the added integration of fault localization into WYSIWYT, users can also, at any time, indicate observations of a failure in a spreadsheet cell by placing an X-mark in that cell’s decision box instead of a checkmark.

For this study, to generate fault likelihood estimates for spreadsheet cells we chose a refinement of the technique introduced in [16]. This technique, which we term the “Blocking” Technique, records the ✓ and X-marks placed on each spreadsheet cell and notes when an X-mark blocks a checkmark from affecting a cell, or vice versa. This testing information is combined with an analysis of the dependencies between the spreadsheet’s cells to estimate the likelihood that each cell in the spreadsheet contributed to the observed failure(s) [16, 18]. The system communicates these fault likelihood estimates to the user by tinting the interior of every suspect (non-input) cell in varying shades of red, even if the user has placed a checkmark on a cell, with darker tints corresponding to increased fault likelihood.

We will term the cells to which C refers (directly or indirectly) C ’s *producers* and the cells referring to C (directly or indirectly) C ’s *consumers*. C ’s fault likelihood is None if none of its consumers have X-marks. C is given a Very Low fault likelihood if all X-marks in its consumers are *blocked* from C by checkmarks on the dataflow path between the X-mark and C . Otherwise, there are X-marks in C ’s consumers that *reach* C (are not blocked by checkmarks), and C ’s fault likelihood is estimated using the equation below, and then mapped to a colorization using the scheme in Table 1.

$$FL(C) = \max(1, 2 * \text{ReachingXMarks} - \text{ReachingCheckmarks})$$

4.2 Procedure

Our subjects were 10 business majors with spreadsheet experience. We divided these subjects into two groups: a control group having only the features of WYSIWYT and a treatment group having both WYSIWYT and the Blocking Technique. (A control group was needed for the debugging strategy comparisons of RQ3.) Each session was conducted one-on-one between an examiner and the

Intensity of Color	<i>fault likelihood(C)</i>
Low	1-2
Medium	3-4
High	5-9
Very High	10+

Table 1. Mapping fault likelihood calculations to color intensities.

subject. The subject was given training on thinking aloud, and a brief tutorial on the environment they would be working in. The tutorial consisted of hands-on instruction on the basic features, followed by a practice task. Also, the environment had an on-demand explanation system via tool tips, available for all objects, reducing or eliminating the need for memorization.

After familiarizing themselves with their environment, each subject worked on the tasks detailed in Section 4.3. The data collected for each session included audio transcripts, electronic transcripts capturing all user interactions with the system, post-session written questionnaires, and the examiner’s observations.

4.3 Tasks

Allwood classified faults in spreadsheets as mechanical, logical and omission faults [1], and this scheme is also used in Panko’s work [15]. Under Allwood’s categorization, mechanical faults include simple typographical errors or wrong cell references in the cell formulas. Mistakes in reasoning were classified as logical faults. Logical faults in spreadsheets are more difficult than mechanical faults to detect and correct, and omission faults are the most difficult [1]. An omission fault is information that has never been entered into the formula.

Drawing from this research, we seeded five faults into each of two Forms/3 spreadsheets. One of these spreadsheets is the Grades spreadsheet from Figure 1, which computes the total score for a course given input for three quizzes, extra credit, a midterm, and a final exam. There is also an output cell that indicates when an input cell is outside the valid range. Grades has three mechanical faults, one logical fault, and one omission fault. This spreadsheet was designed to be the “easier” of our two tasks based on its size, the complexity of its formulas, and our choice of seeded faults.

The other spreadsheet, Payroll, is presented in Figure 2. Into Payroll, we seeded one mechanical fault, three logical faults, and one omission fault. This spreadsheet was much larger, had more levels of data flow, and had a greater number of output cells in relation to input cells when compared to the Grades spreadsheet.

Subjects were given these two spreadsheets (tasks) in varying order, with instructions to test and correct any errors found in the spreadsheets. For each task, the subjects were provided the unverified spreadsheet and a description of the spreadsheet’s functionality. Furthermore,

the subjects were provided a single example of the expected output values given specified inputs for the Grades task, and two such examples for the Payroll task. Subjects had a time limit of 15 minutes to complete the Grades task and 30 minutes to complete the Payroll task.

5. Results

5.1 RQ1: How much perceived value do end users see in the feedback over time?

Blackwell’s model of attention investment [2] is one model of user problem-solving behavior predicting that users will not want to enter an X-mark unless the benefits of doing so are clear to them. The model considers the costs, benefits, and risks 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 an X-mark has cost, benefit, and risk. The costs are figuring out when and where to place the X-mark and thinking about the resulting feedback. The benefit of finding faults may not be clear after only one X-mark; in fact, the user may have to expend even more costs (place more X-marks) 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 in the incorrect ones.

First, we consider whether users, having briefly seen X-marks in the tutorial, were willing to enter even one X-mark to help their debugging efforts. The answer was that they were: four of the five treatment subjects placed at least one X-mark, especially when they needed assistance debugging a failure (discussed further in Section 5.4). The subject who did not place an X-mark (treatment subject TS3) explained during a post-session interview that she had forgotten about them, and wished she had used them:

TS3: I wish I could redo the problem with the X-mark. If I would have done that, it would have been lot more easier.

In our interactive fault localization system, the first

interaction about a failure (X-mark) leads to feedback, and this feedback may or may not provide enough benefits to lead to a second X-mark. In general, a difference in any *interactive* fault localization approach from traditional approaches is that the accuracy of feedback about fault locations must be considered at every step of the way, especially in early steps, not just at the end of some long batch of tests. As the attention investment model explains, if the early feedback is not seen as providing information that is truly of practical help, there may never be any more interactions with the system! This was exactly the case for subject TS5, who placed only one X-mark in the Grades task, and explained after the session:

TS5: To me, putting an X-mark just means I have to go back to do more work.

In his case, the X-mark he placed was in a cell whose only contributors were input cells; consequently, because our technique does not tint input cells (which do not have formulas), the only cell tinted was the cell with the X-mark. Since this feedback did not add to the information he already had, it is not surprising that he found no benefit from placing an X-mark. This indicates the importance of the feedback (reward), even in the early stages of use; if the reward is not deemed sufficient for further attention, a user may not pursue further use.

However, the other three treatment subjects who placed an X-mark went on to place a second X-mark later in the session. Further, after placing this second X-mark, all three subjects then went on to place a third X-mark. Clearly, the successes that rewarded these subjects (detailed in Section 5.3) outweighed their perceived costs of testing and marking failures with X-marks.

5.2 RQ2: How thoroughly do end users understand the interactive feedback?

To what extent did the subjects understand the message the interactive feedback was trying to communicate? We investigated two levels of understanding: the deeper level being the ability to predict feedback under various

Payroll										
0% Tested										
5	Married	8,000	84,000	0	50,000	8,000				
Allowances	MStatus	Salary	OldYTDGrossPay	Pre_TaxChild	LifeInsurAmount	TotalGrossPay				
1,250		6,750	675	650.2	650.2	92,000				
FedWithHoldAllow	AdjustMinusWithhold	SingleWithHold	MarriedWithHold	FedWithHold	NewYTDGrossPay					
5,000	310	116	25	480	39	8,000				
GrossOver87K	SocSec	Medicare	LifeInsurPremium	HealthInsur	Dental	AdjustGrossPay				
505	520	0			1,581.2	6,418.8				
InsurCost	InsurContrib	ExcessInsurCost			EmployeeTaxes	NetPay				

Figure 2. The Payroll spreadsheet.

circumstances, and the more shallow level of being able to interpret feedback received.

To investigate these two levels of understanding, the post-session questionnaire for our treatment subjects had 11 questions, approximately evenly divided between the two levels, regarding the effects of X-marks on the interior colorings of a cell. The subjects’ ability to *predict* behavior, as measured by 6 questions, was mixed. Again using producer-consumer terminology, all subjects were able to correctly predict the impacts on producer cells of placing a single X-mark (2 questions). About half the subjects were able to predict the impacts on consumer cells of placing a single X-mark (1 question) and to predict the impacts when multiple X- and checkmarks were involved (3 questions). However, the ability to *interpret* behaviors was uniformly good: all four of the subjects who actually used X-marks during the session were able to explain the meanings of the colorings and marks, and to say what those meanings implied about faults (5 questions). For example, some responses to questions about what it means when the interior of cells get darker or get lighter were:

TS1: If the color becomes lighter, the cells have less of a chance of to be wrong.

TS2: The chance of an error in the darker cells is greater than in the lighter cells.

TS4 (referring to a darker cell): Higher chance of error in that cell.

These post-session questionnaire results are corroborated by the actions of the users themselves, as we will discuss in the next two sections.

5.3 RQ3: What debugging strategies do end users use to find faults?

Because this work is about fault *localization*, we focus on users’ abilities to *identify* the location of faults, as defined by either an explicit verbal statement or by the fact that they edited the cell’s formula. (Once identified, corrections usually followed; 60 of the 69 faults were corrected once identified.)

Once a failure was spotted, users exhibited two kinds of strategies to find the fault causing the failure: an ad hoc strategy, in which they examined cell formulas randomly in no particular order, and a dataflow strategy, in which they followed the failure’s dependencies back through cell references until they found the fault. A dataflow strategy can be accomplished through mental effort alone, but subjects rarely did this: mostly they used either arrows, the fault localization feedback, or a combination of both.

Table 2 enumerates the subjects’ strategy choices and corresponding success rates. Comparing the first two columns’ percentages column-wise shows that, for both subject groups, dataflow debugging tended to be more

successful than ad hoc. Within dataflow, the treatment subjects’ success rates with X-marks exceeded the dataflow total success rates. A row-wise comparison of the denominators in the table also shows that treatment subjects tended to move to dataflow strategies nearly twice as frequently as the control subjects.

These differences in strategy choices lead to the following question: In what situations did the strategies matter?

Easy faults: The subjects’ strategy choices did not matter with the easiest faults: The easiest are mechanical faults, according to Allwood [1], and were usually found regardless of strategy used. Over all tasks and all subjects, 35 of the 40 mechanical faults were identified.

Local faults: Strategy also did not matter much with the “local” faults (those in which the failed value spotted by the subject was in the same cell as the faulty formula). This is often the case in smaller spreadsheets, where there are fewer cells to reference and the likelihood of a local fault is greater, and probably contributed to both groups’ greater success in the Grades task.

Non-local faults: Strategy mattered a great deal for the non-local faults. Over all of the subjects and tasks, 16 non-local faults were identified—all using dataflow. *Not a single non-local fault was identified using the ad hoc strategy.* In fact, for 7 of these non-local fault identifications (by 6 different subjects), the subjects began their search for the fault using an ad hoc strategy and, when unable to succeed, switched to a dataflow strategy, with which they succeeded in finding the fault.

The fault localization technique augments the dataflow strategy, which is illustrated by treatment subjects TS4 and TS5. Both subjects found all faults in the smaller Grades task. Both subjects also found the mechanical

	Ad hoc	Dataflow		Total
		dataflow total	using X-marks	
<i>Grades:</i>				
Control	13 / 20 (65%)	6 / 6 (100%)	n/a	19 / 26 (73%)
Treatment	13 / 16 (81%)	9 / 10 (90%)	5 / 5 (100%)	22 / 26 (85%)
Total	26 / 36 (72%)	15 / 16 (94%)		41 / 52 (79%)
<i>Payroll:</i>				
Control	6 / 17 (35%)	3 / 6 (50%)	n/a	9 / 23 (39%)
Treatment	9 / 21 (43%)	6 / 12 (50%)	3 / 5 (60%)	15 / 33 (45%)
Total	15 / 38 (39%)	9 / 18 (50%)		24 / 56 (43%)

Table 2. The success rates of identifying a fault contributing to an observed failure (faults identified / failures observed), for each debugging strategy.

fault and one of the logical faults in the large Payroll task in short order. But then, they both got stuck on where to go next. At this critical juncture, TS4 decided to place an X-mark (carried out with a right-click) on a failure. Once he saw the feedback, he rapidly progressed through the rest of the task, placing 5 X's and correcting the final 3 faults in only 7 minutes. The transcripts show that the initial X-mark, which initiated the (dataflow-oriented) fault localization feedback, was a key turning point for him:

TS4 (thinking aloud): Right click that 'cause I know it's incorrect, highlights everything that's possible error... employee taxes is also incorrect. My net pay is incorrect. Adjusted gross pay is incorrect, so click those wrong.

Whereas TS4 made the defining decision to use the X-mark, TS5 did not. TS5's pattern of looking at cells gradually became ad hoc. He started randomly looking at formulas. He made decisions about the values in various cells and eventually happened upon a local fault, bringing his total to 3. He said "I'm getting confused here" numerous times, but did not change his approach.

5.4 RQ4: How does this feedback influence an interactive debugging strategy?

We had initially expected that treatment subjects would always place X-marks whenever they observed a failure and use the subsequent visual feedback to guide their debugging, but this was not the case. Instead, they seemed to view the X-marks as a device to be called upon when they were in need of assistance. For example, only late in the session, when treatment subject TS1 got stuck debugging the failures, did he turn to the fault localization technique:

TS1 (thinking aloud): I don't know how to check the kind of error it is. I'll mark it wrong and see what happens.

When subjects did place an X-mark, the visual feedback often had an immediate impact: regardless of what their previous strategy had been, as soon as the feedback appeared, the subjects switched to a dataflow strategy by limiting their search to those cells with estimated fault likelihood and ignoring cells with no assigned fault likelihood.

TS1 (thinking aloud): I'm going to right-click on the total score. See that the weighted average, the weighted quiz, the weighted midterm, and the weighted final, and the error box all turn pink.

The fault localization device beckons the user toward a dataflow strategy, but it has attributes dataflow arrows do not have. First, it produces a smaller search space than the dataflow arrows, because it highlights only the producers that actually *did* contribute to a failure (the dynamic slice), rather than including the producers that could contribute to failures in other circumstances (the static

slice). Second, it prioritizes the order in which the users should consider the cells, so that the faulty ones are likely to be considered earliest. The above shows that TS1 took advantage of the reduction in search space brought about by the tinting of the producers of a cell with a failure. But did the subjects also take advantage of this prioritization, indicated by some cells being darker than others?

Our electronic transcripts indicate that the answer to this question is yes. When the subjects searched cell formulas for a fault after placing an X-mark, 77% of these searches initially began at the cell with the darkest interior shading. As an example, here is a continuation of the above quote from TS1 after placing an X-mark:

TS1 (thinking aloud): See that the weighted average, the weighted quiz, the weighted midterm, and the weighted final, and the error box all turn pink. The total score box is darker though.

When the fault was not in the darkest cell, subjects' searches would gradually progress to the next darkest cell and so on. Some subjects realized that the colorings' differentiations could be enhanced if they made further testing decisions by placing \surd and X-marks, carried out by left- or right-clicking a cell's decision box.

TS4 (thinking aloud): Right click that 'cause I know it's incorrect, highlights everything that's possible errors. Now, I know my total gross pay is correct. I'll left click that one and simplify it.

From the results of this and the previous sections, it is clear that fault localization's ability to draw the user into a suitable strategy (dataflow) was important, particularly when subjects had not figured out a strategy that would help them succeed better than ad hoc approaches. Further, it is clear that subjects were influenced by the feedback's prioritization information when more than one color was present—in that they looked first to the darkest cells, and then to the next darkest, and so on—and that their doing so increased success.

5.5 RQ5: How do wrong testing decisions affect fault localization feedback?

Being human, the end-user subjects in our study made some mistakes in their testing decisions. Here we consider the types of mistakes they made, and the impact of these mistakes on the users' successful use of the fault localization feedback. (Because the control subjects did not have fault localization feedback, we consider only the treatment subjects.)

In total, the five treatment subjects placed 241 checkmarks, of which 11 (4.56%) were wrong—that is, the user pronounced a value correct when in fact it was incorrect. Surprisingly, however, no subjects made incorrect X-marks.

A possible reason for this difference may be a perceived seriousness of contradicting a computer's calcu-

lations, meaning subjects were only willing to place X-marks when they were really sure their decision was correct. For example, at one point, subject TS1 placed an X-mark in a cell, then reconsidered the mark because he was unsure the X-mark was really warranted.

TS1 (thinking aloud): So, I'll right click on that one. I'm not sure if this is right. Eh, I'll leave it as a question mark.

In contrast, checkmarks were often placed even if the user was unsure they were warranted. Our verbal transcripts include 10 different statements by treatment subjects with this sentiment. For example, consider the following from the same subject as quoted above:

TS1 (thinking aloud): I'll go ahead and left click the Life-InsurPrem box because I think that one's right for now.

What impact did the wrong checkmarks have on fault localization? Four of the 11 wrong checkmarks were placed with a combination of X-marks, resulting in incorrect fault localization feedback. All four of these particular checkmarks, placed by three different subjects, adversely affected the subjects' debugging efforts.

For example, during the Grades task, TS1 placed an incorrect checkmark in the (faulty) WeightedMidterm cell. He later noticed that the Total_Score cell, although its formula was correct, had an incorrect value (due to the fault in WeightedMidterm). Unable to detect the source of this failure, he turned to the fault localization technique and placed an X-mark in the Total_Score cell:

TS1 (thinking aloud): The total score box is darker though. And it says the error likelihood is low, while these other boxes that are a little lighter say the error likelihood is very low. Ok, so, I'm not sure if that tells me anything.

The subject knew that Total_Score was correct. Figure 3 illustrates that had it not been for the wrong checkmark, the faulty cell WeightedMidterm cell would have been one of the two darkest cells in the spreadsheet. Instead, the wrongly placed checkmark caused WeightedMidterm to be colored the same as its correct siblings, thus providing the subject with no insightful fault localization feedback. (The subject eventually corrected the fault after a search of over six minutes.)

Subject TS2, faced with a similar scenario as in Figure 3, was overcome with confusion:

TS2 (thinking aloud): All right... so, I'm doing something wrong here. (long pause) I can't figure out what I'm doing wrong.

TS2's confusion resulted in nearly seven minutes of inactivity. He eventually located and corrected the fault, but remained flustered for the duration of the session.

As this evidence makes clear, it would not be realistic to ignore the fact that end users will provide some wrong information. In our study, even though fewer than 5% of

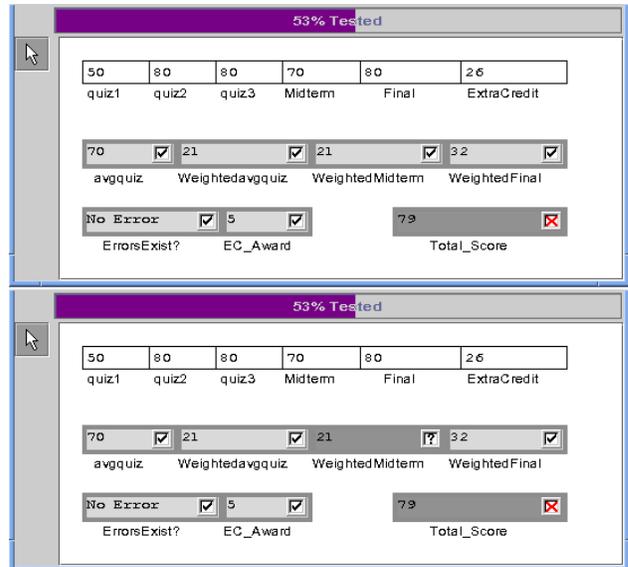


Figure 3. (Top) The Grades task, with an incorrect checkmark in WeightedMidterm, as seen by subject TS1. TotalScore is the darkest, and the other 6 all are the same shade. (Bottom) What TS1 would have seen without the wrong checkmark: WeightedMidterm would be as dark as TotalScore.

the checkmarks placed by the subjects were wrong, these marks affected 60% (3 out of 5) of the treatment subjects' success rates! Given the presence of mistakes, robustness features are necessary to allow success even in the presence of mistakes. Toward this end, recall from Section 4.1 that the fault localization technique used in this study colors every cell in the dataflow chain contributing to a failure—even the cells the user may have previously checked off. Clearly, however, this attempt at robustness was not enough to completely counteract the impacts of mistakes. Alternative techniques whose build-up of historical information can overcome some number of errors [18] are another possibility.

6. Conclusions

Previous fault localization research has focused primarily on techniques to aid *professional programmers* performing *batch* testing. In contrast, our study focuses on supporting *end-user programmers* with an *interactive* fault localization technique. Some revealing results were:

- The subjects did not use fault localization from the beginning. Rather, they treated fault localization as a resource to be called upon only when they had exhausted their own debugging abilities.
- When they did turn to fault localization, it often helped. Also, end users seemed to *perceive* the fault localization feedback as helpful to their interactive

debugging, as evidenced by the continued use of the technique by the majority of our treatment subjects.

- Some subjects realized without help that a dataflow strategy was needed, but some did not. While dataflow-based debugging strategies may seem intuitively superior in the eyes of traditional software engineers, our study indicates that such strategies may not come naturally to end-user programmers. One key way the fault localization technique helped was to lead them into a suitable strategy. Once subjects were engaged in a suitable strategy, fault localization helped further by prioritizing the order they should follow the strategy.
- End users make mistakes, and because even a few mistakes can have a big impact on fault localization's helpfulness, the importance of these mistakes should not be ignored. Thus, fault localization techniques should include features to enhance robustness in the face of a few mistakes.

Perhaps the most challenging result was the important role of early interactive feedback. Our study found that if a fault localization technique's *early* feedback is not seen to be useful, users may not give the technique a chance to produce better feedback later. The early feedback of a fault localization technique may be of little consequence to professional programmers performing batch testing of test suites; yet this issue may be paramount to the success of an interactive technique in an end-user programming environment.

Acknowledgments

We thank the members of the Visual Programming Research Group at Oregon State University for their feedback and help. This work was supported in part by NSF under ITR-0082265.

References

- [1] C. Allwood, "Error Detection Processes in Statistical Problem Solving," *Cognitive Science* 8, 4, 1984, 413-437.
- [2] A. Blackwell, "First Steps in Programming: a Rationale for Attention Investment Models," *IEEE Symp. Human-Centric Computing Langs. and Envs.*, Arlington, VA, Sept. 3-6, 2002, 2-10.
- [3] B.W. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Soft. Cost Estimation with COCOMO II*, Prentice Hall PTR, Upper Saddle River, NJ, 2000.
- [4] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace, "End-User Software Engineering with Assertions in the Spreadsheet Paradigm," *Int. Conf. Soft. Eng.*, Portland, OR, May 2003, 93-103.
- [5] 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," *J. Func. Prog.* 11, 2, Mar. 2001, 155-206.
- [6] D.A. Carr, "End-User Programmers Need Improved Development Support," *CHI 2003 Workshop on Perspectives in End User Development*, April 2003, 16-18.
- [7] T.Y. Chen and Y.Y. Cheung, "On Program Dicing," *Soft. Maintenance: Research and Practice* 9, 1, 1997, 33-46.
- [8] J.S. Davis, "Tools for Spreadsheet Auditing," *Int. J. Human-Computer Studies* 45, 1996, 429-442.
- [9] N. Heger, A. Cypher, and D. Smith, "Cocoa at the Visual Programming Challenge 1997," *J. Visual Langs. and Computing* 9, 2, Apr. 1998, 151-168.
- [10] T. Igarashi, J.D. Mackinlay, B.-W. Chang, and P.T. Zellweger, "Fluid Visualization of Spreadsheet Structures," *IEEE Symp. Visual Langs.*, 1998, 118-125.
- [11] J.A. Jones, M.J. Harrold, and J. Stasko, "Visualization of Test Information to Assist Fault Localization," *Int. Conf. Soft. Eng.*, May 2002, 467-477.
- [12] H. Lieberman and C. Fry, "ZStep 95: A Reversible, Animated Source Code Stepper," *Soft. Visualization: Programming as a Multimedia Experience* (J. Stasko, J. Domingue, M. Brown, and B. Price, eds.), MIT Press, Cambridge, MA, 1998, 277-292.
- [13] R.C. Miller and B.A. Myers, "Outlier Finding: Focusing User Attention on Possible Errors," *ACM Symp. User Interface Software and Technology*, Nov. 2001, 81-90.
- [14] B. Myers and A. Ko, "Studying Development and Debugging to Help Create a Better Programming Environment," *CHI 2003 Workshop on Perspectives in End User Development*, April 2003, 65-68.
- [15] R. Panko, "What We Know About Spreadsheet Errors," *J. End User Computing*, Spring 1998.
- [16] J. Reichwein, G. Rothermel, and M. Burnett, "Slicing Spreadsheets: An Integrated Methodology for Spreadsheet Testing and Debugging," *2nd Conf. Domain Specific Langs.*, Oct. 1999, 25-38.
- [17] G. Rothermel, L. Li., C. Dupuis, and M. Burnett, "What You See Is What You Test: A Methodology for Testing Form-based Visual Programs," *Int. Conf. Soft. Eng.*, Apr. 1998, 198-207.
- [18] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Prabhakararao, M. Fisher II, and M. Main, "End-User Software Visualizations for Fault Localization," *ACM Symp. Soft. Visualization*, Jun. 2003, 123-132.
- [19] J. Sajanieme, "Modeling Spreadsheet Audit: A Rigorous Approach to Automatic Visualization," *J. Visual Langs. and Computing* 11, 1, 2000, 49-82.
- [20] Telcordia Technologies, "xSlice: A Tool for Program Debugging," xsuds.argreenhouse.com/html-man/coverpage.html, July 1998.
- [21] F. Tip, "A Survey of Program Slicing Techniques," *J. Programming Langs.* 3, 3, 1995, 121-189.
- [22] 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," *ACM Conf. Human-Computer Interaction*, Fort Lauderdale, FL, Apr. 5-10, 2003, 305-312.