AN ABSTRACT OF THE THESIS OF

Amit D Phalgune for the degree of Master of Science in Computer Science presented on September 12, 2005.

Title: Garbage In, Garbage Out? An Empirical Look at Oracle Mistakes by End-User Programmers.

Abstract approved: ____

Margaret M. Burnett

End-user programmers, because they are human, make mistakes. However, past research has not considered how visual end-user debugging devices could be designed to ameliorate the effects of mistakes. This work empirically examines oracle mistakes – mistakes users make about which values are right and which are wrong – to reveal differences in how different types of oracle mistakes impact the quality of visual feedback about bugs. We then consider the implications of these empirical results for designers of end-user software engineering environments.

Garbage In, Garbage Out?
An Empirical Look at Oracle Mistakes by End-User Programmers

by
Amit D. Phalgune

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented September 12, 2005
Commencement June 2006

Master of Science thesis of Amit D. Phalgune presented on September 12, 2005

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Amit D. Phalgune, Author

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude towards my advisor Dr. Margaret Burnett for her constant support, guidance and encouragement throughout my work. She has been a wonderful guide and a great source of inspiration towards the work presented in this thesis.

This work has been done in collaboration with Cory Kissinger, Dr. Margaret Burnett, Laura Beckwith, Dr. Curtis Cook and Joseph Ruthruff. I would like to thank them from the bottom of my heart. A special thanks to Cory Kissinger for working on the related work and assisting me with data extraction. I would also like to thank Dr. Mary Beth Rosson, Dr. Tessa Lau and Dr. Deepak Phalgune for their useful insights into my work.

I also take this opportunity to thank all the members of the Forms/3 team.

Last but not the least; I must thank my father, Dr. Deepak Phalgune, my mom Mrs. Usha Phalgune, my sister Anagha and all other family members who have been very supportive and have encouraged me throughout my career. I must thank my parents and sister for having the courage and determination of passing through the most difficult phase of their lives, while I pursued my master's program.

This work was supported in part by the National Science Foundation under Award ITR-0325273.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST TO TABLES

LIST OF APPENDIX FIGURES

LIST OF APPENDIX TABLES

Garbage In, Garbage Out?
An Empirical Look at Oracle Mistakes by End-User Programmers

1. Introduction

1.1 Introduction

When software is developed by software professionals, these professionals have formal training in writing software as well as are expected to know formal techniques for testing and thereby debugging the software programs. Since these software professionals have formal knowledge in testing and debugging it is much easier for them to test their programs.

However, with the widespread use of computers in the present age, a lot of software is being created by people with little or no programming experience, referred to as end-user programmers. With the use of computers becoming more popular day by day we find more and more end-user programmers writing their own programs. One estimate indicates that there are about 55 million end-user programmers as compared to the 2.75 million professional programmers [Boehm et al. 2000]. Another estimate, [Scaffidi et al. 2005] indicates that by 2012 there will be 90 million end-user programmers of which more than 55 million will use spreadsheets or databases. These estimates clearly indicate that the spreadsheet paradigm is a commonly used paradigm. For example, we can imagine the payroll staff for a large company (the payroll staff do not have formal training in writing software and hence are considered end users) using spreadsheets to calculate the various tax benefits and paychecks for different employees in the company. Teachers in schools and colleges might want to write spreadsheets that help them calculate grades of students for a particular class. Creating a payroll spreadsheet or a grades spreadsheet involves writing formulas. Writing formulas in Excel is how one programs in the spreadsheet paradigm.

Though the spreadsheet paradigm may seem to be easy, empirical research suggests that users make an astonishingly large number of faults [Panko 1995, Betts and Horowitz 2004, Panko 1998]. Some of these faults can have a serious economic impact [Hilzenrath 2003, Panko 1995, Robertson 2003]. The term *fault* mentioned above refers to an incorrect formula in a spreadsheet cell, which causes an incorrect output to be generated. Similarly another commonly used term in the software engineering literature is *failure*, referring to an incorrect output generated in the presence of certain input values.

The above findings encourage us to think seriously about supporting the software development process of end users and helping them write reliable software. To this end, the End Users Shaping Effective Software (EUSES) Consortium has come up with an approach called “end-user software engineering” [Burnett et al. 2004]. This holistic approach provides support for end users to make their software dependable and more reliable. The approach provides various devices that help in improving the quality of software, such as testing methodologies that help detect failures, *assertions,* which continually monitor values and indicate that values are “out of range” and *fault localization* devices that help programmers to find causes of failures.

The end-user software engineering environments tend to differ from the traditional software engineering environments in that they are more interactive in nature, are usually non-modal, and the feedback provided by these systems is incremental based on user input [Ruthruff and Burnett 2004]. Further, it is usual for traditional fault localization techniques to assume that the inputs provided to their algorithms are indeed correct and produce an incorrect output if the input data provided to these algorithms is incorrect. Previous studies [Prabhakararao et al. 2003, Ruthruff et al. 2005A] indicate that users often make mistakes when performing testing and debugging activities (interacting with the system). Thus it may be unreasonable to have fault localization techniques that are unreliable in the presence of incorrect information, especially in the end-user programming environments. The work presented in this thesis concentrates on this critical difference between professional and end-user software development environments, specifically in the context of fault localization devices.

1.2 The Problem Addressed by this Thesis

Mistakes occur in every domain of human action, and thus it should be no surprise that they arise in end-user programming. In the early decades of computing, a common saying was “garbage in, garbage out.” That is, mistakes in communicating with a computer were aberrations, and if users provided bad data (garbage in), then they should expect the software to produce incorrect answers (garbage out).

Of course, with the joint advents of interactive systems and HCI as a subarea of computer science, it was realized that people do make mistakes in communicating with computers, and features began to appear to help prevent mistakes (such as menus instead of typed-in commands) and to allow people to detect and recover from them (such as immediate feedback and undo facilities).

Still, below this surface level, the philosophy of “garbage in, garbage out” remains: if the user’s mistake somehow gets in unnoticed, then surely he or she still should expect “garbage out.” The unfortunate consequence is that it then seems reasonable for developers to assume that software needs to work correctly only when no mistaken data is provided to the system.

In the work presented in this thesis, we consider whether the above assumption is a reasonable assumption in software whose purpose is to help end-user programmers reason about and debug their programs. In the problem solving domain it is not always straightforward for users to make correct judgments about how well different parts of their program are working, and thus some mistakes are *inevitable*. We consider the prevalence and effects of these mistakes in order to determine whether end-user software engineering environments offering testing and debugging support to end-user programmers must be designed with this inevitability in mind.

The type of testing and debugging mistakes upon which this thesis focuses are *oracle mistakes* [Weyuker 1982], a term meaning a falsely positive or falsely negative judgment as to whether an output computed by the program is correct. The end-user programming environment prototype in which we consider this type of mistakes is a spreadsheet environment that includes a visual testing and fault localization device we have developed known as WYSIWYT (What You See Is What You Test) [Rothermel et al. 2001, Ruthruff et al. 2004]. In this thesis, we explore how different subsets of oracle mistakes impact the effectiveness of interactive, visual testing and debugging support for end-user programmers, and what might be done to ameliorate these impacts.

2. Background and Related Work

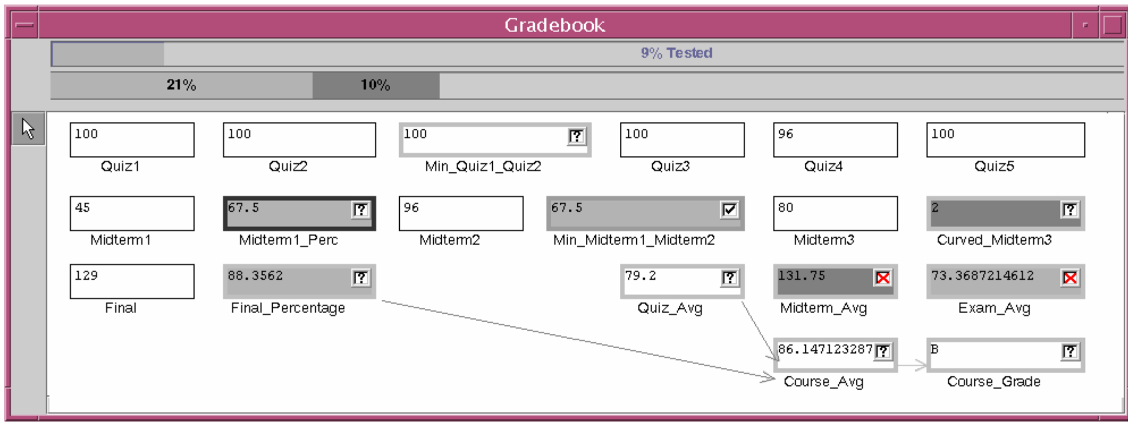
2.1 End-User Debugging

There has been recent research focusing on assisting end-user programmers in debugging. Virtually all of this work communicates with the user largely in the form of visual devices. Woodstein [Wagner and Lieberman 2004] is a software agent that visually assists users in debugging e-commerce applications. Ko and Myers present the Whyline [Ko and Myers 2004], an "interrogative debugging" device for the event-based programming environment Alice. There has also been a variety of work supporting program comprehension and debugging by end users in the spreadsheet paradigm. For example, Igarashi et al. [Igarashi et al. 1998] present devices to aid spreadsheet users in dataflow visualization and editing tasks. S2 [Sajanieme 2000] 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 [Davis 1996]. Ayalew and Mittermeir [Ayalew and Mittermeir 2003] present a method of fault tracing based on "interval testing" and slicing, which is similar to our own work on assertions to help users automatically guard against faults [Burnett et al. 2003]. There is also recent work to automatically detect certain kinds of errors, such as errors in spreadsheet units [Abraham and Erwig 2004] and types [Ahmad et al. 2003].

2.2 WYSIWYT with Visual Fault Localization

Members of our research group have been working on a vision of end-user software engineering [Burnett et al. 2004] that we have prototyped in the spreadsheet paradigm because it is so widespread. Our vision of end-user software engineering involves holistic support of the facets of software development in which end users engage, tied together through incremental visual devices. These visual devices are necessarily low in cost to maintain the immediate responsiveness expected by spreadsheet users, and are immediately updated as end users add to, modify, test, and debug their programs.

WYSIWYT with visual fault localization is part of this vision. An example from our research prototype, Forms/3 [Burnett et al. 2001], a spreadsheet language that utilizes "free-floating" cells in addition to traditional spreadsheet grids, is shown in Figure 1. The underlying assumption behind the WYSIWYT testing methodology is that, as a user incrementally develops a spreadsheet program, he or she is also testing incrementally. Because the intended audience is end users, all communication about



Quiz1: 100
Quiz2: 100
Min_Quiz1_Quiz2: 100
Quiz3: 100
Quiz4: 96
Quiz5: 100
Midterm1: 45
Midterm1_Perc: 67.5
Midterm2: 96
Min_Midterm1_Midterm2: 67.5
Midterm3: 80
Curved_Midterm3: 2
Final: 129
Final_Percentage: 88.3562
Quiz_Avg: 79.2
Midterm_Avg: 131.75
Exam_Avg: 73.3687214612
Course_Avg: 86.147123287
Course_Grade: B

Figure 1: Gradebook spreadsheet with an oracle mistake (√, a false positive) on cell Min_Midterm1_Midterm2. The correct mark (X) would have resulted in a more accurate prediction of the faulty cell's (Midterm1_Perc) fault likelihood, resulting in a darker coloring than the relatively light coloring it has here.

testing is performed through visual devices.

In the course of testing the spreadsheet, the user can communicate a judgment that a cell's value is correct with a checkmark (√), or that a cell's value is incorrect with an X-mark (X), as shown in Figure 1. Checkmarks contribute to the "testedness" of the cells according to an adequacy criterion detailed in [Rothermel et al. 2001], and a cell's testedness is reflected in border colors along a red-to-blue continuum (light gray to black in this thesis). X-mark invokes the fault localization device. As Figure 1 shows, the system also provides dataflow arrows on demand. They reflect dependencies among cells and their color reflects the testedness of specific relationships among cells and subexpressions.

The system combines the user's checkmarks and X-marks with the dependencies in the cells' formulas to estimate likelihoods of the fault (erroneous formula) being located in various cells. It colors these cells' interiors in light-to dark amber (gray) to reflect these likelihoods [Ruthruff et al. 2003].

Given these communication devices, a *false positive* is a checkmark on a value that is incorrect, and a *false negative* is an X-mark on a value that is correct (Figure 2).

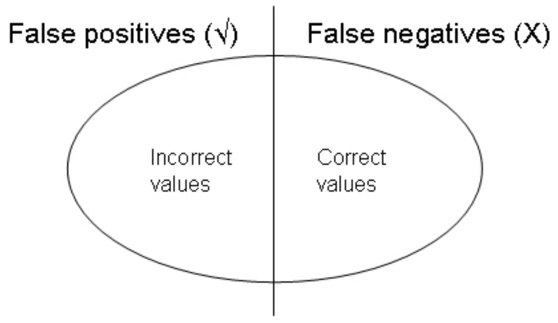


Figure 2: Oracle mistakes consist of false positives and false negatives.

2.3 Mistakes in Interactive Testing and Debugging Environments

Like most environments for end-user programmers, the spreadsheet paradigm is modeless and interactive: users incrementally experiment with their software and see how the results work out after each change; an example of this in spreadsheets is the automatic recalculation feature. This means that testing and debugging support in these types of environments must accommodate continuous interaction with the end user.

In Figure 1, suppose there is a fault in cell Midterm1_Perc, causing incorrect values in several downstream cells. Unfortunately, the user has made an oracle mistake by checking off (√) Min_Midterm1_Midterm2's value (a false positive). Because of this mistake, Midterm1_Perc is only mildly implicated as the culprit by the visual feedback, as the figure shows.

The research literature provides little guidance regarding how the developers of visual devices for interactive testing and debugging should handle oracle mistakes and their impacts on such devices' accuracy of feedback. In fact, the presence of *any* mistakes runs contrary to a common assumption in traditional testing and debugging research—that all information is accurate and reliable—meaning that much of the prior software visualization research is inherently unsuited for the interactive environments that end-user programmers utilize.

There is, however, some information about mistakes in end-user debugging upon which we can build. In the course of other investigations we have done on end-user debugging, we have reported the presence of oracle mistakes observed [Ruthruff et al.

2004, Ruthruff et al. 2005A, Ruthruff et al. 2005B], which gives a little preliminary information about their prevalence. Ko and Myers [Ko and Myers 2003] also shed some light on the impact of such mistakes. In their work, they used the phrase “determining a failure” to describe all errors in perceiving and understanding output, which has a significant overlap with our definition of oracle mistakes. In their observational study, problems of the “determining failure” type made up *28 of the 29 breakdowns* that occurred during debugging, strongly implying that this type of problem is one of the most important factors when debugging goes astray. Combined, these works suggest that designers of interactive visual fault localization devices may not be able to ignore the possibility of oracle mistakes. This paper investigates this possibility.

3. Experiment

To gain insight into the importance of oracle mistakes on end-user testing and debugging we consider the following research questions:

RQ1:	How often do oracle mistakes occur?
RQ2:	Do oracle mistakes impact effectiveness of a fault localization device?
RQ3:	Are oracle mistakes tied with end users' understanding of the debugging device?
RQ4:	Do oracle mistakes impact end users' ability to debug?
RQ5:	Do "smart" oracle mistakes impact effectiveness of the fault localization device differently?

The first research question, RQ1, provides us insights into whether we, as researchers, should care about oracle mistakes at all. Do oracle mistakes occur with a reasonable amount of frequency that we should be concerned about them?

Regarding RQ2 and RQ4, checkmarks and X-marks are the primary vehicle by which the user can interact with the system. End-user programming environments are typically interactive in nature and the visual feedback which the system provides the user with could have serious impact on the actions the user performs. Hence the tie between effectiveness of the fault localization device and oracle mistakes is an important consideration. Previous research conveys a trend that the testing and debugging strategies of end users are largely dependent upon the visual feedback they receive. For example our previous studies indicate that users move from an ad-hoc testing and debugging strategy to a more systematic strategy in the presence of visual feedback [Prabhakararao et al. 2003]. Thus, we consider RQ2 and RQ4 to determine the extent to which visual feedback impacts the testing and debugging progress an end user makes.

Another major effect of oracle mistakes could be on the end user's understanding of the device they are using (RQ3), which can indirectly affect their progress.

Regarding RQ5, there is a class of oracle mistakes we have termed "smart" mistakes that might be effective in terms of the fault localization feedback provided to end users. One would normally assume that mistakes are bad and would usually lead to bad feedback from the system (garbage in, garbage out). By evaluating the last research question, we investigate whether these "smart" oracle mistakes could provide visual feedback that might be helpful to the users.

The subsequent sections describe in detail the design of our experiment to pursue the research questions mentioned above.

3.1 Design

This investigation consists of both an observational study and a planned experiment.

A collection of naturally occurring oracle mistakes was available to us in the electronic transcripts from a previous experiment. We refer to this collection of transcripts, which contains the oracle mistakes the participants actually made, as *Version Original*. For the observational study, we simply analyze effects of an observed independent variable (number of oracle mistakes) on observed dependent variables (discussed later).

For the planned experiment, the design was a within-subjects design, in which the treatment variable being manipulated—i.e., the independent variable—starts with the same collection of observed oracle mistakes. We then manipulated these observed oracle mistakes, in ways we describe later in this section, to generate new simulated versions with which we can compare Version Original.

3.2 What the Original Participants Did

The data for Version Original were obtained from a previous experiment that we conducted [Beckwith et al. 2005]. Fifty one participants took part in that original experiment. Each participant was seated one per computer. The participants were mainly from the school of business at Oregon State University. They were given a background questionnaire to start with. After completing the background questionnaire, the 51 participants were given a "hands-on" tutorial to familiarize them with the Forms/3 environment. The tutorial, which lasted for 35 minutes taught participants the use of WYSIWYT checkbox for checking off correct values and associated feedback, but did not include debugging or testing strategy content. We also did not teach the use of fault localization; rather, participants were introduced to the mechanics of placing X-marks and given time to explore any aspects of the feedback that they found interesting.

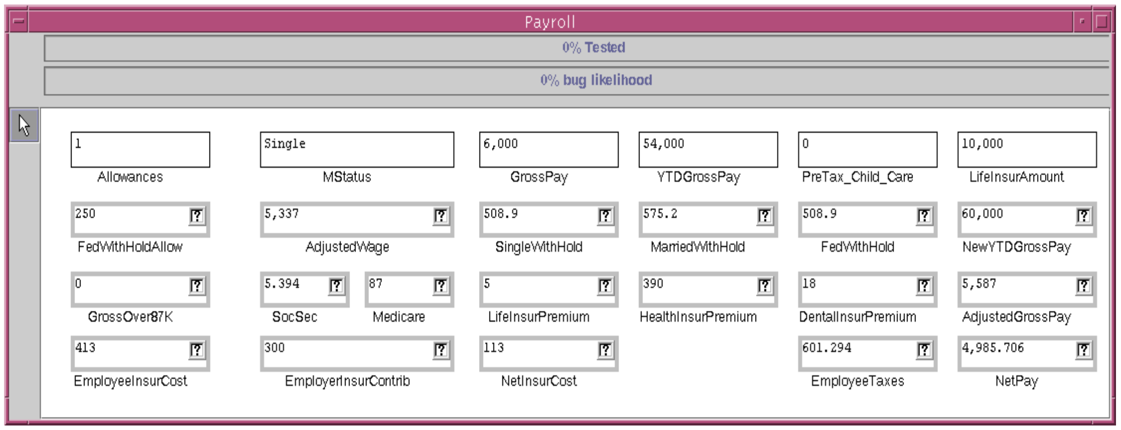
3.2.1 Tasks

The participants were provided with the two spreadsheets to debug. The use of two spreadsheets reduced the chances of the results being due to any one spreadsheet's particular characteristics. The experiment was counterbalanced with respect to task order so as to distribute learning effects evenly.

We collected electronic transcripts of every action taken by the participants and the system's resulting feedback. We also captured the final state of the spreadsheets after the experiment was over. At the end of each task the participants were asked to fill out a post-session questionnaire which measured the participants' ability to comprehend various aspects of our system.

The two spreadsheets were Gradebook (Figure 1) and Payroll (Figure 3). To make the spreadsheets representative of real end-user spreadsheets, Gradebook was derived from an Excel spreadsheet of an (end-user) instructor, which we ported into an equivalent Forms/3 spreadsheet. (To accommodate Forms/3 features, a minor change was made to two minimization operators.) Payroll was a spreadsheet designed by two Forms/3 researchers using a payroll description from a real company.

These spreadsheets were seeded with five faults created by real end users. Gradebook was seeded with three of these users' mechanical faults, one logical fault, and one omission fault, and Payroll with two mechanical faults, two logical faults, and



Allowances	1
MStatus	Single
GrossPay	6,000
YTDGrossPay	54,000
PreTax_Child_Care	0
LifeInsurAmount	10,000
FedWithHoldAllow	250
AdjustedWage	5,337
SingleWithHold	508.9
MarriedWithHold	575.2
FedWithHold	508.9
NewYTDGrossPay	60,000
GrossOver87K	0
SocSec	5.394
Medicare	87
LifeInsurPremium	5
HealthInsurPremium	390
DentalInsurPremium	18
AdjustedGrossPay	5,587
EmployeeInsurCost	413
EmployerInsurContrib	300
NetInsurCost	113
EmployeeTaxes	601.294
NetPay	4,985.706

Figure 3: The Payroll Spreadsheet

one omission fault. (Under Panko's classification [Panko 1998] mechanical faults include simple typographical errors or wrong cell references. Logical faults are mistakes in reasoning and are more difficult than mechanical faults. An omission fault is information that has never been entered into a cell formula, and is the most difficult to detect [Panko 1998].) Payroll was intended to be the more difficult task due to its larger size, greater length of dataflow chains, intertwined dataflow relationships, and more difficult faults. (To validate if Payroll was harder than Gradebook, we measured the participants' confidence about how many bugs they fixed, through post session questionnaires. Although we did not find a statistical significance in the confidence levels, the average confidence level in the Payroll task was lower than the Gradebook task, with a mean of 2.7 for Payroll and 2.8 for Gradebook.) The participants were provided, in varying order, with the Gradebook and Payroll spreadsheets along with their descriptions and were given 22 and 35 minutes respectively to test the spreadsheet thoroughly and to ensure that it does not contain errors and works according to the spreadsheet description. They were also instructed to fix any bugs they find in the spreadsheets. The time limits involved on the spreadsheet tasks helped us ensure that the participants worked on both spreadsheets.

3.2.2 Fault Localization Algorithm

We have been experimenting with three different fault localization algorithms [Ruthruff et al. 2005B]. In this experiment, the participants were supported by a fault localization algorithm known as Test Count [Ruthruff et al. 2003]. Let NumFailingTests (NFT) be the number of failed tests in which a cell c has contributed to the output, and NumSuccessfulTests (NST) be the number of successful tests in which c has contributed. With Test Count, if cell c has no failed tests, the fault likelihood of c is "None". Otherwise, the fault likelihood of cell c is computed as:

$$\text{Fault likelihood}(c) = \max(1, 2*NFT - NST)$$

3.3 Current Experiment's Procedures

Using Version Original as a base, we manipulated the numbers and types of oracle mistakes to generate three additional versions of the data: Version FalsePositivesOnly, Version FalseNegativesOnly, and Version Ideal.

3.3.1 Three Generated Versions

In *Version FalsePositivesOnly,* we removed each false negative mistake (in which the user placed an X-mark on a value that was in fact correct), replacing it with a non-mistaken judgment, namely a checkmark. To prevent this "what if" version from straying too far from what the original participants actually saw, however, we corrected each false negative mistake one at a time, in isolation from the others. Specifically, we ran a simulation on the original data to collect the fault localization feedback after each action, and as soon as a false negative mistake was detected, we replaced the erroneous X-mark with its correct counterpart (a checkmark), reported the new fault localization feedback that the system provided, and then restored the original mistake before proceeding on with the simulation. This procedure prevented a cascading effect from the accumulation of better feedback effects.

Similarly, in *Version FalseNegativesOnly*, we removed each false positive mistake (in which the user checked off a value that was in fact incorrect), replacing it with a non-mistaken judgment, namely an X-mark, using the same safeguard against accumulated effects as for Version FalsePositivesOnly.

By designing such a procedure we were able to isolate the difference in the visual feedback provided by the system when a false positive / negative oracle mistake was corrected, thus providing a way to determine the visual feedback that participants would have seen, had they not made such an oracle mistake.

Version Ideal was intended to reflect the best feedback each participant could have had if he or she had made no mistakes at all. It corrected both types of oracle mistakes. For this version, there was no reason to guard against cascading effects; it simply reflected the ideal feedback that could be achieved by making no oracle mistakes when judging (marking) whatever cells each participant judged.

3.3.2 "Smart" Mistakes

Some users did not confine their use of the checkmarks and X-marks solely to the way we designers had anticipated. The intent of the marks is to communicate judgments of the correctness of the *values* of the cells they marked. However, during several earlier think-aloud studies we have noticed users placing marks to communicate judgments of the *formulas*. For example, even when a value was

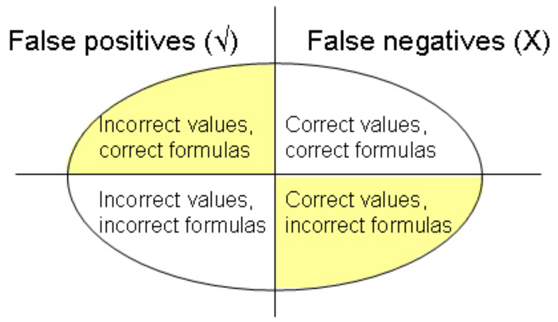

False positives (√)
False negatives (X)
Incorrect values, correct formulas
Correct values, correct formulas
Incorrect values, incorrect formulas
Correct values, incorrect formulas

Figure 4: The shaded sectors show the smart mistakes.

incorrect, some users checked off the cell because the formula was correct. This is indeed an oracle mistake, since it is falsely approving a value, but it has a possible rationale behind it. Thus, we term this type of mistake as a *smart mistake*. The remaining mistakes are termed *no-rationale mistakes*. See Figure 4.

In investigating the impacts of these two, mutually exclusive, subsets of mistakes, we created a variation of Version Ideal that isolated the smart mistakes (correcting only the no-rationale mistakes). This version is termed *Version Smart*.

3.4 Dependent Variables and Measures

For the portion of our investigation conducted via the observational study, the observed dependent variables were the participants' actual bugs fixed and their bugs introduced. We chose to focus on these variables because oracle mistakes seem likely to affect debugging success.

We also required a measure of the fault localization technique's effectiveness. Since an important goal of this experiment was to study the impact of oracle mistakes on the visual feedback of our fault localization device, as in our previous work [Ruthruff et al. 2005B], we defined the fault localization technique's effectiveness as the technique's ability to correctly and visually differentiate the correct cells in the spreadsheet from those that actually contain faults. Let *FaultyCells(AvgFL)* be the average fault likelihood of colored faulty cells. Let *CorrectCells(AvgFL)* be the average fault likelihood of colored correct cells. The formula to calculate visual effectiveness (VE) according to this measure is then:

$$\text{Visual Effectiveness} = \text{FaultyCells(AvgFL)} - \text{CorrectCells(AvgFL)}$$

4. Results

4.1 RQ1: Prevalence of Oracle Mistakes

As Table 1 illustrates, 17.1% and 22.5% of the judgments made in Gradebook and Payroll, respectively, were mistaken. This frequency is even worse than that observed in previous work [Ruthruff et al. 2005A, Ruthruff et al. 2005B], which ranged from 5% to a bit over 20%. In fact, in this study, only two out of the 51 participants managed to *not* make oracle mistakes.

4.1.1 Magnitude of Mistakes

To gain further understanding of oracle mistakes that users make, we investigated the magnitude of oracle mistakes. Magnitude of oracle mistakes was measured as the percentage difference in the actual value (the value the participant saw and made an oracle mistake on) against the correct value (the value that would have been generated by Version Ideal). The bar graphs (Figures 5 and 6) indicate percentage difference in the values on the X-axis and the corresponding count of mistakes on the Y-axis.

Discussion: The bar graphs provide us with interesting results. 48 out of 148 (32.65%) *no-rationale* oracle mistakes in the Gradebook spreadsheet and 25 out of 133 (18.8%) in the Payroll spreadsheet had a percentage difference of 9% or less.

Table 1: Frequency of oracle mistakes for each task (as observed in Version Original).

Task	**Number of marks (judgments)**	**Number of oracle mistakes**	**%**	**Number of false positives**	**Number of false negatives**	**Mean mistakes per user**	**Median**
Gradebook (n=51)	899	154	17.1	144	10	3.02	2
Payroll (n=51)	1,696	381	22.5	354	27	7.27	4

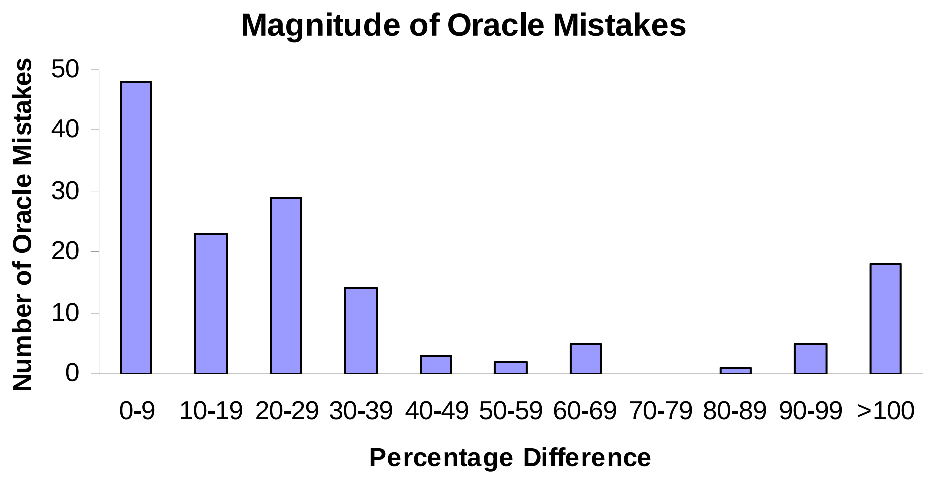


Figure 5: Number of no-rationale oracle mistakes categorized by percentage difference in the Gradebook spreadsheet.

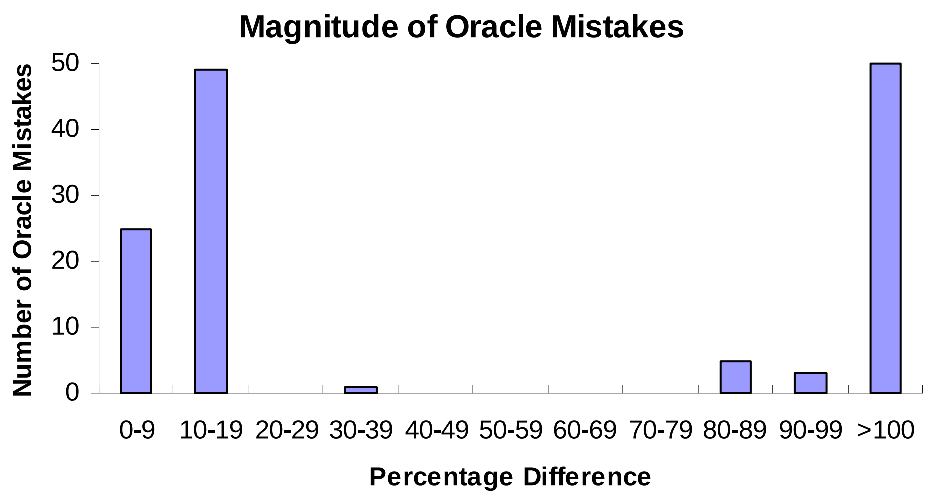


Figure 6: Number of no-rationale oracle mistakes categorized by percentage difference in the Payroll spreadsheet.

Mistakes with such a small percentage difference (less than 10%) in the actual value and the correct value can occur easily, probably because of slight miscalculations or improper approximations. Because these judgments were pretty close to the actual values in the spreadsheet, they seem likely to be resistant to researchers' (or users') attempts to eradicate them.

4.2 RQ2: Impact on Visual Effectiveness

To measure the impact of oracle mistakes on the effectiveness of the fault localization's feedback, we compared the visual effectiveness of the feedback the user actually saw to the visual effectiveness they might have seen if each single oracle mistake had instead been a correct judgment. Since we wanted to stay as close as possible to what the users really saw, Versions FalsePositivesOnly and FalseNegativesOnly were the right versions for this question, but Version FalsePositivesOnly had too few changes for any meaningful statistical analysis.

Thus, we compared Version Original with the version with most of the mistakes removed (i.e., Version FalseNegativesOnly), using the following (null) hypothesis as a statistical vehicle:

H2-1: There will be no difference between the visual effectiveness of the feedback produced in Version Original and that produced in Version FalseNegativesOnly.

For both Gradebook and Payroll (Table 2), there was a significant difference in the average visual effectiveness scores between the two versions (paired t-test: df = 50, Gradebook: t=10.57, p<.001; Payroll: t=2.19, p=.03). Thus, we reject H2-1.

Discussion: Obviously, the use of a device, any device, in an incorrect manner will negatively impact that device's effectiveness. Still, this result combined with the findings from RQ1 establishes a critical point: designing an interactive fault localization device under the assumption that oracle mistakes can be ignored is not reasonable. The investigation of RQ1 shows that oracle mistakes occurred with great frequency and the investigation of RQ2 shows they did significant damage.

Table 2: Mean / median of Version FalseNegativesOnly feedback and Version Original. (Note: The mean/median are average visual effectiveness scores over all participants)

	Visual Effectiveness Version FalseNegativesOnly	**Visual Effectiveness Version Original**
Gradebook	.73 / .71	.01 / .00
Payroll	.25 / .12	.01 / .00

4.3 RQ3: Relationship to Understanding

The results thus far clearly show that oracle mistakes are a major problem for end-user fault localization devices.

But a possible remedy springs to mind: maybe lack of understanding of the fault localization device is causing the problem. If that is the case, we might work on increasing users' understanding of the device to reduce the number of oracle mistakes. To consider whether this would be a profitable direction, we considered the following hypothesis:

H3-1: There will be no relationship between users' oracle mistakes and their understanding of the debugging device.

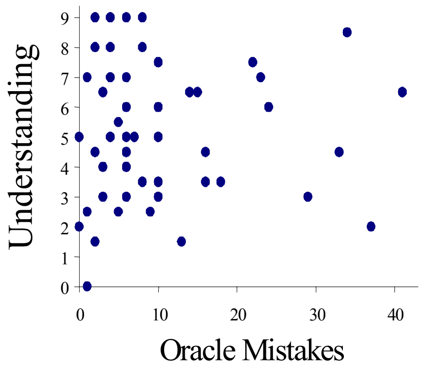


Figure 7: Most participants made between 1 and 10 oracle mistakes, regardless of their understanding scores.

Understanding was measured via post-session questionnaire scores, with a maximum score possible of 10. Regression analysis on the observed data in Version Original (Figure 7), showed no significant relationship between the users' understanding of the debugging device and the number of oracle mistakes made (linear regression: F(1,49) = .104, β=.204, R^2= .002, p = .75).

Discussion: The users' understanding of the device does not seem implicated. The results show that participants with a better understanding of the device did not make fewer mistakes.

4.4 RQ4: Impact on Debugging

Section 4.2 considered the *system's* ability to produce good feedback in the presence of oracle mistakes by making comparisons with a generated version of the data. We now turn to solely observed data to consider the *user's* ability to succeed at debugging in the presence of oracle mistakes.

H4-1: The number of oracle mistakes users made will have no relationship to the number of bugs they fixed or introduced.

Results of the regression analyses of the participants' number of oracle mistakes

Table 3: Regression analyses of number of oracle mistakes vs. bugs fixed and bugs introduced.

Progress Measure	**F(1,49)**	**B**	**R^2**	**p-value**
Gradebook:				
Fixed	4.34	-.179	.081	.043
Introduced	8.02	.195	.141	.007
Payroll:				
Fixed	< .001	< .001	< .001	.986
Introduced	.69	.021	.014	.411

compared to their ability to fix the bugs we seeded, and to avoid introducing new bugs, is shown in Table 3. The regression coefficient is the slope of the least squares fitting of number of mistakes against the progress measures.

We found a significant relationship between oracle mistakes and both bugs fixed and introduced for the Gradebook task (linear regression: data shown in Table 3). While debugging the Gradebook task, participants made 154 oracle mistakes, introduced 82 bugs and fixed 173 bugs. The Payroll task, however, did not show any significant relationship between the number of oracle mistakes a user made and the user's debugging. While performing the Payroll task, participants made 371 oracle mistakes, introduced 68 bugs, while fixing 150 bugs. In the Gradebook spreadsheet the number of oracle mistakes was directly proportional to the number of bugs fixed and inversely proportional to the number of bugs introduced (See Figure 8). Thus, for the Gradebook task, we reject H4-1. (Even so, note that the low R^2 values do not indicate a good fit despite the level of significance.)

Discussion: We were surprised that on the Payroll task, the harder of the two, oracle mistakes had no relationship to a user's debugging success. However, a closer

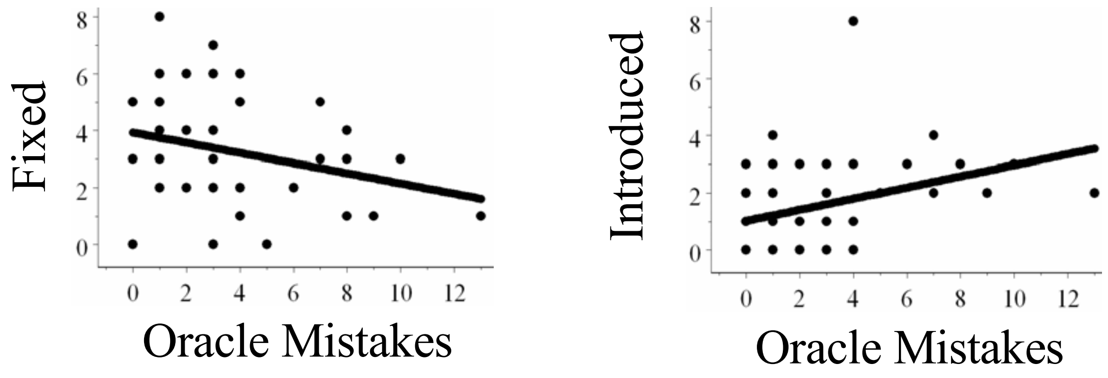


Figure 8: Relationship between oracle mistakes and bugs fixed, bugs introduced in the Gradebook spreadsheet.

analysis of the characteristics of the oracle mistakes themselves may explain this, which we consider next.

4.5 RQ5: Impact of Smart Mistakes

Consider the implications of making a “smart” oracle mistake. For example, suppose the user placed an X-mark on a cell with a correct value but an incorrect formula. Most fault likelihood algorithms are based on evidence of “guilt” (judgments that values are bad), and this is the case with our fault localization algorithm as well. As a result, this cell will be colored “more faulty” (darker in our prototype) because the user has just implicated it. Thus, although the user has mistakenly communicated that the value is wrong; a desirable side effect is that this cell, which is indeed faulty, has just gotten darker. For this particular cell then, its visualized fault likelihood is *better* than if the user had not made the oracle mistake!

However, looking to this cell’s backward slice (the cells contributing to this cell’s value), there is likely to be a detrimental effect on visual effectiveness, because all of these cells will be wrongly labeled as contributing to an incorrect value (which they did not), and as therefore being potentially faulty.

So, these “smart” mistakes are helpful in some respects and harmful in others. To consider just how helpful or harmful they are, we compared the effects of smart mistakes to the ideal:

Table 4: Number of smart mistakes made compared to total number of mistakes.

	Number of smart mistakes	**Number of oracle mistakes**	**%**
Gradebook	10	154	6.5
Payroll	213	381	55.9

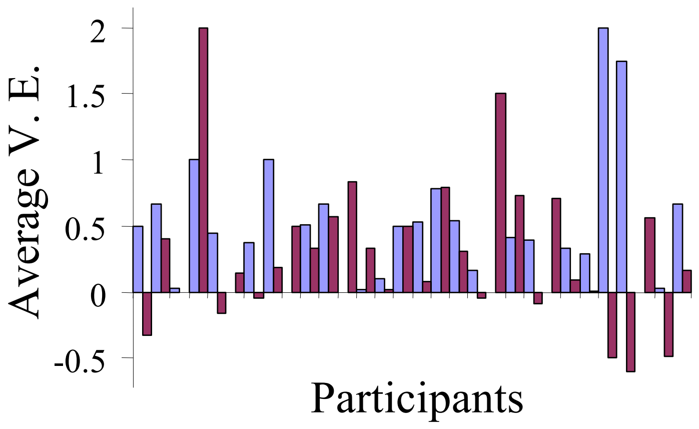


Figure 9: Average visual effectiveness for each participant under Version Smart (light bars) and Version Ideal (dark bars).

H5-1: There will be no difference in visual effectiveness between Version Smart and Version Ideal.

As Table 4 shows, there were only a few smart mistakes in the Gradebook task, but over half of the oracle mistakes in Payroll were smart mistakes. (This is interesting, and could be tied to the difference in relative difficulty between the two.)

Because Gradebook had so few smart mistakes, the effects of smart mistakes on its visual effectiveness scores could not be analyzed statistically, but we analyzed Payroll as follows. The feedback in Version Smart (the version from which the no-rationale mistakes had been corrected) was compared to that of Version Ideal. The difference in the average visual effectiveness scores was not significant in magnitude, due in part to the fact that there were also a number of correct marks present. However, as Figure 9 shows, the average visual effectiveness score for each participant under Version Smart was higher a startling number of times (11 out of 19 times). Analysis of these counts showed that Version Smart indeed had significantly better feedback than Version Ideal (Fishers Exact Test: p = .03).

Discussion: Our results above showed that, for this particular spreadsheet, smart mistakes were better than perfection! These results may, of course, be due to the particular relationships present in that spreadsheet. (For example, for a spreadsheet with longer dataflow chains the detrimental effect on the backward slice could outweigh the positive effect on the cell that was directly marked.) But despite the fact that in some cases the global negatives could win out, the fact remains that this type of mistake was extremely common in Payroll, and has a strongly positive impact on the cell being marked.

RQ5's result may explain the differences that were seen between Gradebook and Payroll in RQ4. Many of the mistakes in Payroll were smart mistakes, and RQ5 implies that these mistakes would not have deleterious effects on debugging.

Finally, recall that the results of RQ2 showed that, overall, oracle mistakes had a negative impact on visual effectiveness. Since the smart mistakes included in RQ2's results were actually helping the visual effectiveness of the feedback, the implication is that the no-rationale mistakes had a very strong negative effect—strong enough to significantly negate the positive effects of correct marks and smart mistakes together.

5. Implications for Designers of End-User Environments

Given these results, how should designers of end-user environments proceed? At least two possible strategies present themselves: (1) find ways to lessen the impact of no-rationale mistakes, and (2) find ways to strengthen the positive impacts of smart mistakes.

As it happens, the Test Count fault localization algorithm already tempers the negative impact of no-rationale mistakes. The way it does so is through a "robustness" property [Ruthruff et al. 2005B]. Suppose there is a cell c marked with an X-mark. The robustness property requires that all cells in the backward slice of cell c receive at least some visual fault coloring, no matter how many positive tests have also been run. This feature guarantees that any faulty cell contributing to a failure the user observed will be one of the cells highlighted. Since false positives were by far the most common type of oracle mistake, without this robustness feature, the detrimental impact of mistakes would have been even worse than it was in our study.

Another fortunate attribute of Test Count is that it gives double the weight to negative judgments as it does to positive judgments (refer back to Section 3.2 for the calculation). This is fortunate because most of the mistakes were false positives, not false negatives, as has also been true in our previous experiments. Hence, correct judgments are getting more weight than incorrect ones. This may well account for the mostly positive (above zero) visual effectiveness scores observed throughout our results.

The surprising results of RQ5 suggest an opportunity for improved robustness. We realized that any negative effects of smart mistakes must be solely the global effects, since locally a smart mistake is actually beneficial. The Test Count algorithm used, gives equal weight to local and global impacts of the marks made. Thus, we wondered if reducing the impact a mark can have on cells in its backward slice (or equivalently, increasing the local impact) would improve visual effectiveness. We decided to evaluate this idea empirically.

Recall from Section 3.2 that NumFailingTests (NFT) is the number of failed tests in which a cell c participated, and NumSuccessfulTests (NST) is the number of successful tests in which c participated. For an increased effect on local impacts (an algorithm variant we will term "Test Count Local"), we partitioned NFT into NFT_L (number of failed tests locally) and NFT_G (number of failed tests globally). Similarly, we partitioned NST into NST_L and NST_G. To double the impact of local decisions, the fault localization formula for Test Count Local is thus defined as:

$$\text{Fault likelihood}(c) = \max(1, (4*NFT\_L + 2*NFT\_G) - (2*NST\_L + NFT\_G))$$

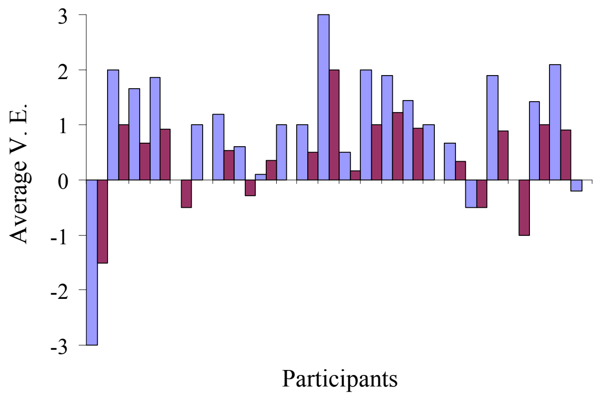
We then compared the visual effectiveness of the feedback the participants actually received (Version Original) to that which they would have received with the Test Count Local algorithm. A paired t-test (used in the same manner as in RQ2 on visual

Table 5: Mean / median of visual effectiveness with Test Count Local and Test Count Original

	Visual Effectiveness Test Count Local	**Visual Effectiveness Test Count Original**
Gradebook	.94 / 1.00	.36 / .35
Payroll	.38 / .39	.09 / .13

effectiveness scores) revealed that Test Count Local produced feedback whose visual effectiveness was significantly better than the feedback produced by Test Count Original (paired t-test: Gradebook: df =23, t=4.73, p<.001; Payroll: df=31, t=2.99, p=.005). See Figure 10 and Table 5. This is especially revealing since Gradebook had few smart mistakes whereas Payroll had many. Thus, the local emphasis used in Test Count Local was significantly better overall at ameliorating the effects of oracle mistakes than Test Count Original.

These results do not appear to be specific to our particular Test Count algorithm. In a previous investigation of fault localization algorithms [Ruthruff et al. 2005B], one algorithm, Nearest Consumer, outperformed the other two (one of which was Test



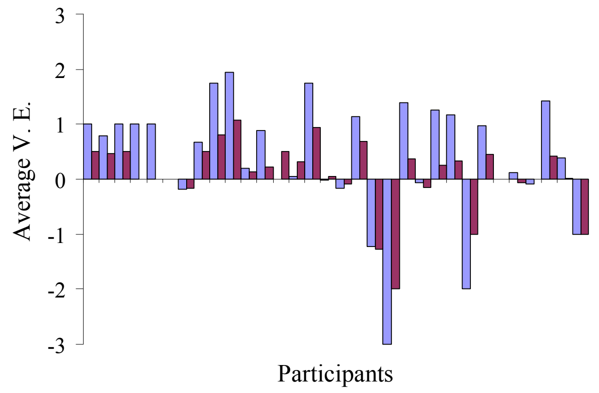


Figure 10: Average visual effectiveness for each user with Test Count Local (light bars) versus Test Count Original (dark bars). Top: Gradebook Bottom: Payroll

Count), both with and without the presence of oracle mistakes. We were unable to explain this, since Nearest Consumer actually uses less information and is less precise than Test Count. However, it emphasizes local impacts in a manner similar to Test Count Local, which now seems likely to have played an important role in that algorithm's increased effectiveness and robustness.

6. Conclusion

In this work, we have investigated the impact of different types of oracle mistakes on the quality of visual feedback that can be achieved by end-user fault localization devices based on testing. Our investigation uncovered these surprising results:

- Contrary to traditional assumptions, it is not reasonable to ignore oracle mistakes in designing interactive fault localization devices: oracle mistakes occur with too great frequency and have too destructive impacts for such an assumption to be reasonable.
- Finding ways to improve users' understanding of testing and fault localization devices is not likely to solve the problem of oracle mistakes.
- There is a subset of oracle mistakes (termed "smart" mistakes) that is actually helpful—achieving better effects locally than even ideal (oracle mistake-free) performance.

These findings lead to implications that can be employed by interactive fault localization algorithms to help ameliorate the effects of oracle mistakes.

- Smart mistakes turned out to be helpful locally. Thus, weighing the local effects more than the global effects for the marks users place should help fault localization algorithms continue to give good feedback even in the presence of smart mistakes.
- The fault localization algorithms should have a robustness feature.
- Previous work as well as this work shows that users make a lot more false positive oracle mistakes than false negative oracle mistakes. Hence negative judgments should be trusted more than the positive judgments.

Empirical results in this work as well as previous work [Ruthruff et al. 2005B] provide evidence that using these features can significantly improve the effectiveness of interactive fault localization devices for end-user programmers.

Bibliography

[Abraham and Erwig 2004] R. Abraham and M. Erwig, "Header and unit inference for spreadsheets through spatial analyses", *Proc. IEEE Symp. Visual Languages and Human-Centric Computing*, 2004, 165-172.

[Ahmad et al. 2003] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi, "A type system for statically detecting spreadsheet errors", *Proc. IEEE Intl. Conf. Automated Software Engineering*, 2003, 174-183.

[Ayalew and Mittermeir 2003] Y. Ayalew and R. Mittermeir, "Spreadsheet debugging", *Proc. European Spreadsheet Risks Interest Group*, 2003.

[Beckwith et al. 2005] L. Beckwith, M. Burnett, S. Wiedenbeck, C. Cook, S. Sorte, M. Hastings, "Effectiveness of end-user debugging software features: Are there gender issues?", *Proc. ACM Conf. Human Factors in Computing Systems,* 2005, 869-878.

[Betts and Horowitz 2004] M. Betts and A. S. Horowitz, "Oops! Audits find errors in 49 out of 54 spreadsheets", *Computerworld*, May 24, 2004, page 47.

[Boehm et al. 2000] B. W. Boehm, C. Abts, A. W. Brown, and S. Chulani. *Software Cost Estimation with COCOMO II*. Prentice Hall PTR, Upper Saddle River, 2000.

[Burnett et al. 2001] 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. Functional Programming*, 11, 2, 2001, 155- 206.

[Burnett et al. 2003] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace, "End-user software engineering with assertions in the spreadsheet paradigm", *Proc. Intl. Conf. Software Engineering*, 2003, 93-103.

[Burnett et al. 2004] M. Burnett, C. Cook, and G. Rothermel, "End-user software engineering", *Comm. ACM*, 2004, 53-58.

[Davis 1996] J. S. Davis, "Tools for spreadsheet auditing", *Intl. J. Human-Computer Studies,* 45, 1996, 429-442.

[Hilzenrath 2003] D. S. Hilzenrath, "Finding errors a plus, Fannie says; Mortgage giant tries to soften effect of $1 billion in mistakes", *The Washington Post*, October 31,
2003.

[Igarashi et al. 1998] T. Igarashi, J. D. Mackinlay, B. W. Chang, and P. T. Zellweger, "Fluid visualization of spreadsheet structures", *Proc. IEEE Symp. Visual Languages*, 1998, 118-125.

[Ko and Myers 2003] A. J. Ko and B. A. Myers, “Development and evaluation of a model of programming errors”, *Proc. IEEE Symp. Human-Centric Computing Languages and Environments*, 2003, 7-14.

[Ko and Myers 2004] A. J. Ko and B. A. Myers, “Designing the Whyline: A debugging interface for asking questions about program failures”, *Proc. ACM Conf. Human Factors Computing Systems*, 2004, 151-158.

[Panko 1995] R. Panko, “Finding spreadsheet errors: Most spreadsheet errors have design flaws that may lead to long-term miscalculation”, *Information Week*, May 1995, 100.

[Panko 1998] R. Panko, “What we know about spreadsheet errors”, *J. End User Computing*, 1998, 15-21.

[Prabhakararao et al. 2003] S. Prabhakararao, C. Cook, J. Ruthruff, E. Creswick, M. Main, M. Durham, and M. Burnett, “Strategies and behaviors of end-user programmers with interactive fault localization”, *Proc. IEEE Symp. Human-Centric Computing Languages and Environments*, 2003, 15–22.

[Robertson et al. 2003] G. Robertson, “Officials red-faced by $24m gaffe: Error in contract bid hits bottom line of TransAlta Corp.”, *Ottawa Citizen*, June 5, 2003.

[Rothermel et al. 2001] G. Rothermel, M. Burnett, L. Li, C. Dupuis, and A. Sheretov, “A methodology for testing spreadsheets”, *ACM Trans. Software Engineering and Methodology,* 10, 1, 2001, 110-147.

[Ruthruff et al. 2003] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Prabhakararao, M. Fisher II, and M. Main, “End-user software visualizations for fault localization”, *Proc. ACM Symp. Software Visualization,* 2003, 123-132.

[Ruthruff and Burnett 2004] J. Ruthruff and M. Burnett, “Interactive Fault Localization Techniques to Empower the Debugging Efforts of End-User Programmers”, *Technical Report*, Oregon State University, 04-60-10, July 2004.

[Ruthruff et al. 2004] J. Ruthruff, A. Phalgune, L. Beckwith, M. Burnett, and C. Cook, “Rewarding ‘good’ behavior: End-user debugging and rewards”, *Proc. IEEE Symp. Visual Languages and Human-Centric Computing*, 2004, 107-114.

[Ruthruff et al. 2005A] J. Ruthruff, S. Prabhakararao, J. Reichwein, C. Cook, E. Creswick, and M. Burnett, “Interactive, visual fault localization support for end-user programmers”, *J. Visual Languages and Computing,* 16, 1-2, 2005, 3-40.

[Ruthruff et al. 2005B] J. Ruthruff, M. Burnett, and G. Rothermel, “An empirical study of fault localization for end-user programmers”, *Proc. Intl. Conf. Software*

Engineering. 2005, 352-361.

[Sajanieme 2000] J. Sajanieme, “Modeling spreadsheet audit: A rigorous approach to automatic visualization”, *J. Visual Languages and Computing,* 11, 1, 2000, 49-82.

[Scaffidi et al. 2005] C. Scaffidi, M. Shaw and B. Myers, “Estimating the numbers of end users and end user programmers”, *Proc. IEEE Symp. Visual Languages and Human-Centric Computing,* Dallas, Texas, USA, 2005 (to appear).

[Wagner and Lieberman 2004] E. J. Wagner and H. Lieberman, “Supporting user hypotheses in problem diagnosis on the web and elsewhere”, *Proc. Intl. Conf. Intelligent User Interfaces*, 2004, 30-37.

[Weyuker 1982] E. Weyuker, “On testing non-testable programs”, *The Computer Journal,* 25, 4, 1982, 465-470.

Appendices

Appendix A: Tutorial Materials

Introduction

Hi, my name is Amit Phalgune, and I will be leading you through today's study.

The other people involved in this study are Dr. Margaret Burnett, Dr. Curtis Cook, Laura Beckwith, Joey Ruthruff, and the assistants helping me out today.

Just so you know, I'll be reading through this script so that I am consistent in the information I provide you and the other people taking part in this study, for scientific purposes.

The aim of our research is to help people create correct spreadsheets Past studies indicate that spreadsheets contain several errors like incorrectly entered input values and formulas. Our research is aimed at helping users find and correct these errors.

For today's experiment, I'll lead you through a brief tutorial of Forms/3, and then you will have a few experimental tasks to work on.

But first, I am required by Oregon State University to read aloud the text of the "Informed Consent Form" that you currently have in front of you:
(Read form).

Please do NOT discuss this study with anyone. We are doing later sessions and would prefer the students coming in not to have any advance knowledge.

Questions?

Contact:

- Dr. Margaret Burnett	burnett@cs.orst.edu
- Dr. Curtis Cook	cook@cs.orst.edu

Any other questions may be directed to IRB Coordinator, Sponsored Programs Office, OSU Research Office, (541) 737-8008

Background Questionnaire *(hand it out, have them fill it out)*

Tutorial

Before we begin, I'd like to ask if anyone in here is colorblind. We will be working with something that requires the ability to distinguish between certain colors, and so we would need to give you a version that does not use color.

In this experiment, you will be working with the spreadsheet language Forms/3. To get you familiarized with the features of Forms/3, we're going to start with a short tutorial in which we'll work through a couple of sample spreadsheet problems. After the tutorial, you will be given two different spreadsheets; asked to test the spreadsheets, and correct any errors you find in them.

As we go through this tutorial, I want you to ACTUALLY PERFORM the steps I'm describing. For example, at times I will want you to click the left mouse button, at times I will want you to click the middle mouse button (the scroll button in the middle of your mouse) and at other times I will want you to click the right mouse button. I will be very clear regarding what actions I want you to perform. Please pay attention to your computer screen while you do the steps.
If you have any questions, please don't hesitate to ask me to explain.
For each spreadsheet that we will be working with, you will have a sheet of paper describing what the spreadsheet is supposed to do.

(Hand out PurchaseBudget Description)

Let's read the description of the "PurchaseBudget" spreadsheet now.

(Wait for them to read)

Now open the PurchaseBudget spreadsheet by selecting the bar labeled PurchaseBudget at the bottom of the screen with your left mouse button.

This is a Forms/3 spreadsheet. There are a few ways that Forms/3 spreadsheets look different than the spreadsheets you may be familiar with:
Forms/3 spreadsheets don't have cells in a grid layout. We can put cells anywhere *(select and move a cell around a bit)*. However, just like with any other spreadsheet, you can see a value associated with each cell.
We can give the cells useful names like PenTotalCost *(point to the cell on the spreadsheet)*.
You can also see that some cells have colored borders.

Let's find out what the red color around the border means. Rest your mouse on top of the border of the PenTotalCost cell *(show wave the mouse around the cell and then rest mouse on border)*. Note that a message will pop up and tell us what this color means. Can anyone tell me what the message says? *(PAUSE, look for a hand.)* Yes, it means that the cell has not been tested.

You might be wondering, what does testing have to do with spreadsheets? Well, it is possible for errors to exist in spreadsheets, but what usually happens is that they tend to go unnoticed. It is in our best interest to find and weed out the bugs or errors in our spreadsheets so that we can be confident that they are correct.

So, the red border around the cells is just telling us that the cell has not been tested. It is up to us to make a decision about the correctness of the cells based on how we know the spreadsheet should work. In our case, we have the spreadsheet description that tells us how it should work.

Observe that the Pens and Paper cells do not have any special border color *(wave mouse around cells).* Such cells without colored borders are called input cells. Cells with colored borders are called formula cells.

Let's test our first cell. To do this, we'll examine the TotalCost cell. Is the cell's value of zero correct? *(PAUSE for a second).* Well, let's look at our spreadsheet description. Look at the Total Cost section of the spreadsheet. It says, "The total cost is the combined cost of pens and paper." Well, both PenTotalCost and PaperTotalCost are zero, so TotalCost appears to have the correct value.

Now drag your mouse over the small box with a question mark in the upper-right-hand corner of the cell. Can anyone tell me what the popup message says? *(PAUSE, wait for answer.)* Yes, it says that if the value of this cell is correct, we can left-click and if the value of the cell is wrong, we can right-click. It also tells us that these decisions help test and find errors.

So let's left-click the question mark in this decision box for TotalCost. Notice what happened. Three things changed. A checkmark replaced the question mark in the decision box *(wave mouse).* The border colors of some cells changed—three cells have blue borders instead of red, and the percent testedness indicator changed to 28% *(point to it).* Forms/3 lets us know what percent of the spreadsheet is tested through the percent testedness indicator. It is telling us that we have tested 28% of this spreadsheet.

Now if you accidentally place a checkmark in the decision box, if the value in the cell was really wrong, or if you haven't seen the changes that occurred, you can "uncheck" the decision about TotalCost by left-clicking on that checkmark in TotalCost's decision box. ***(Try it, and Pause)*** Everything went back to how it was. The cells' borders turned back to red, the % testedness indicator dropped back to 0% and a question mark reappeared in the decision box.

Since we've already decided the value in the TotalCost cell is correct, we want to retell Forms/3 that this value is correct for the inputs. So left-click in the decision box for TotalCost to put our checkmark back in that box.

You may have noticed that the border colors of the PenTotalCost and PaperTotalCost cells are both blue. Now let’s find out what the blue border indicates by holding the mouse over the PenTotalCost cell's border in the same way as before. The message tells us that the cell is fully tested. *(PAUSE)* Also notice the blank decision box in the PenTotalCost and PaperTotalCost cells. What does that mean? Position your mouse on top of the box to find out why it is blank. A message pops up that says we have already made a decision about this cell. But wait, I don't remember us making any decisions about PenTotalCost or PaperTotalCost. How did that happen?

Let's find out. Position your mouse to the TotalCost cell and click the middle mouse button. Notice that colored arrows appear. Click the middle mouse button again on any one of these arrows—it disappears. *(PAUSE)* Now, click the middle mouse button again on TotalCost cell—all the other arrows disappear. Now bring the arrows back again by re-clicking the middle mouse button on TotalCost.

Move your mouse over to the top blue arrow and hold it there until a message appears. It explains that the arrow is showing a relationship that exists between TotalCost and PenTotalCost. The answer for PenTotalCost goes into or contributes to the answer for TotalCost. *(PAUSE)*

Oh, ok, so does explain why the arrow is pointed in the direction of TotalCost? Yes it is, and it also explains why the cell borders of PenTotalCost and PaperTotalCost turned blue. Again, if you mark one cell as being correct and there were other cells contributing to it, then those cells will also be marked correct. *(PAUSE)* We don’t need those arrows on TotalCost anymore, so let’s hide them by middle-clicking on the TotalCost cell.

Now, let’s test the BudgetOk cell by making a decision whether or not the value is correct for the inputs. What does the spreadsheet description say about my budget? Let me go back and read…oh yeah, “You cannot exceed a budget of $2000”.

This time, let’s use the example correct spreadsheet from our spreadsheet description to help us out. Let’s set the input cells of this sheet identical to the values of our example correct spreadsheet in the spreadsheet description. The Pens cell is already zero. But we need to change the value of the Paper cell to 400 so that it matches the example spreadsheet in the description. How do I do this? Move your mouse to the Paper cell and rest the mouse cursor over the little button with an arrow on the bottom-right-hand side of the cell. It says “Click here to show formula.” Let’s do that by clicking on this arrow button. A formula box popped up. Change the 0 to a 400, and click the Apply button. I think I’m done with this formula, so let’s hide it by clicking on the “Hide” button. Moving on, in this example correct spreadsheet, PensOnHand is 25, and PaperOnHand is 21. (*Wave paper around*) Oh good, my spreadsheet already has these values, so I don’t have to change anything.

Now, according to this example correct spreadsheet, BudgetOk should have the value “Budget Ok”. But it doesn’t; my spreadsheet says “Over Budget”. So the value of my BudgetOK? cell is wrong. What should I do?

Remember, anytime you have a question about an item of the Forms/3 environment, you can place your mouse over that item, and wait for the popup message. To remind us what the question mark means, move your mouse to the BudgetOk decision box. The popup message tells us that if the cell’s value is wrong to right-click. Well, this value is wrong, so go ahead and right-click on the question mark in this decision box.

Hey, look at that! Things have changed! Why don’t you take a few seconds to explore the things that have changed by moving your mouse over the items and viewing the popup messages.

Now let’s make a decision about TotalCost’s value. For the current set of inputs, TotalCost should be 1600. But our TotalCost cell says 2800. That means the value associated with the TotalCost cell is “Wrong”. Let’s right-click in the decision box to place an X-mark. Take a few seconds to explore anything that might have changed by moving your mouse over the items and viewing the popup messages.

Finally, I notice that, according to the example spreadsheet in my description, PaperTotalCost should be 1600. But our value is 2800, and that is wrong. So let’s place an X-mark on this cell as well.

There is at least one bug in a formula somewhere that is causing these three cells to have incorrect values. I’m going to start looking for this bug by examining the PaperTotalCost cell. Let’s open PaperTotalCost’s formula. PaperTotalCost is taking the value of the Paper cell and multiplying it by 7. Let me go back and read my spreadsheet description. I’m going to read from the “Costs of Pen and Paper” section. *(read the section)* So the cost of paper is four dollars, but this cell is using a cost of seven. This is wrong. So let’s change the 7 in this formula to a 4, and click the Apply button to finalize my changes.

Hey wait, my total spreadsheet testedness at the top of my window went down to 0%! What happened? Well, since we corrected the formula, Forms/3 had to discard some of our previous testing. After all, those tests were for the old formula. I have a new formula in this cell, so those tests are no longer valid. But, never fear, I can still retest these cells.

For example, the value of this PaperTotalCost cell is 1600, which matches the example spreadsheet in my description. Since this cell is correct, let’s left-click to place a checkmark in the decision box for PaperTotalCost. Oh good, the percent testedness of my spreadsheet went up to 7%; I got some of my testedness back.

Let’s work on getting another cell fully tested. Look at the value of the PaperQCheck cell. Is this value correct? Let’s read the second paragraph at the top of the spreadsheet description. *(read it)* With a value of 400 in the Paper cell, and a value of 21 in the PaperOnHand cell, we have 421 sheets of paper, which is enough to fill our shelves. Since the PaperQCheck cell says "paper quantity ok", its value is correct. So let's click in the decision box of this cell to place a checkmark.

But wait! The border of this cell is only purple. Let's rest our mouse over this cell border to see why. The popup message says that this cell is only 50 percent tested.

Let's middle-click on this cell to bring up the cell's arrows. Hey, the arrows are both purple too. Let's rest our mouse over the top arrow that is coming from the Paper cell. Ah ha, the relationship between Paper and PaperQCheck is only 50% tested! So there is some other situation we haven't tested yet. Let's change the value of the Paper cell to see if we can find this other situation. Click on the little button with an arrow on the bottom-right-hand side of the cell. Let's try changing the value to 380, and click the Apply button.

Now look at the decision box of the PaperQCheck cell. It is blank. I don't remember what that means, so let's rest my mouse over the decision box of this PaperQCheck cell. Oh yeah, it means I've already made a decision for a situation like this one. Okay, let's try another value for the Paper cell. I'm going to try a really small value. Move your mouse back to the formula box for the Paper cell, change its value to 10, and left-click the Apply button. Now push the Hide button on this formula box.

Now look at the PaperQCheck cell. There we go! The decision box for the cell now has a question mark, meaning that if I make a testing decision on this cell, I will make some progress. Let's look at the cell's value. Well, with 10 in the Paper cell and 21 in the PaperOnHand cell, I have 31 paper on stock. Is this enough paper? The spreadsheet description says I need 400 reams of paper, but I only have 31. So this is not enough paper. And the PaperQCheck cell says "not enough paper". Well, this is correct, so let's left-click on the PaperQCheck cell's decision box. Alright! The border changed to blue, and even more, the spreadsheet is now 35% tested.We don't need those arrows on PaperQCheck anymore, so let's hide them by middle-clicking on the PaperQCheck cell.

Why did it take two checkmarks to fully test the PaperQCheck cell? Let's open the cell's formula to find out (*open the formula*). See that this formula has an if-then-else statement. It says that **if** the sum of Paper and PaperOnHand is less than 400, **then** the cell should display "not enough paper". **Else or otherwise**, it should display "paper quantity ok". In other words, for PaperQCheck, if Paper plus PaperOnHand is less than 400, then "not enough paper" should appear in the cell, and if Paper plus PaperOnHand is greater than or equal to 400, "paper quantity ok" should appear in the cell.Push the Hide button on the formula box of the PaperQCheck cell.

Now let's look at the PenQCheck cell. This cell is displaying "pen quantity ok". Is this correct? Our spreadsheet description says you must keep more than 68 boxes of pens on hand. But we only have 25 boxes of pens on hand, because the Pens cell is 0 and the PensOnHand cell is 25. So even though we don't have enough pens, the PenQCheck cell is displaying "pen quantity ok". This value is not correct, so let's right-click on the question mark in PenQCheck's decision box.

I'll give you a couple minutes to try to fix the bug that caused PenQCheck to have this wrong value. After a couple minutes, we'll fix the bug together to make sure that everyone found it.
(*wait exactly two minutes*)

Okay, let's start by looking at PenQCheck's formula. Unless you have changed this cell's formula, it says that if the sum of the Pens and PensOnHand cells is greater than 68, then the cell should contain "not enough pens", and otherwise it should contain "pen quantity ok". But let's go back and look at our spreadsheet description and read that second paragraph again. It says that we only need to keep 68 or more boxes of pens in stock. So, based on the description PenQCheck should really print "pen quantity ok" if Pens plus PensOnHand is greater than 68, and otherwise it should print "not enough pens". So let's change this formula accordingly and push the "Apply" button when we are done. (*wait a second*). Note that PenQCheck now displays the correct value. So let's go ahead and put a checkmark in this cell by left-clicking on the question mark.

Look at the bottom of the description. It says, "Test the spreadsheet to see if it works correctly, and correct any errors you find." Remember, if you are curious about any aspect of the system, you can hover your mouse over the item and read the popup. Also, you might find those checkmarks and X-marks to be useful. Starting now, you'll have a few minutes to test and explore the rest of this spreadsheet, and to fix any bugs you find. Remember, your task is at the bottom of your spreadsheet description.

Number: __________

Background Questionnaire

1. Gender (circle your selection): Male / Female
2. Age < 20 20 – 30 30 – 40 40 – 50 50 – 60 >60
3. Major or Educational Background: ______________________
4. Year or Degree Completed: Fresh. Soph. Jun. Sen. Post Bac. Grad.
5. Cumulative GPA: ______________________
6. Do you have previous programming experience?
    a. High school:
        - How many courses? _____
        - What programming languages?

          ____
    b. College:
        - How many courses? _____
        - What programming languages?

          ____
    c. Professional and/or recreational
        - How many years? _____
        - What programming languages?

          ____

7. Have you ever created a spreadsheet for (please check all that apply):

❑ A high school course	How many? ____________
❑ A college course	How many? ____________
❑ Professional use	How many years? ____________
❑ Personal use	How many years? ____________

8. Have you participated in any previous Forms/3 experiments? Yes / No

9. Is English your primary language? Yes / No

If not, how long have you been speaking English? ______ years.

Pre-session Questionnaire

The following questions ask you to indicate whether you could use a new spreadsheet system under a variety of conditions. For each of the conditions please indicate whether you think you would be able to complete the job using the system.

Given a spreadsheet which performs common tasks (such as calculating course grades or payroll) I could find and fix errors:

... if there was no one around to tell me what to do as I go.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had never used a spreadsheet like it before.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had only the software manuals for references.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had seen someone else using it before trying it myself.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I could call someone for help if I got stuck.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if someone else had helped me get started.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had a lot of time to complete the task.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had just the built-in help facility for assistance.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if someone showed me how to do it first.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had used similar spreadsheets before this one to do this same task.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree

Gradebook.frm

Here is a gradebook spreadsheet problem. Let's read the second paragraph at the top of the description:

"Your task is to test the updated spreadsheet to see if it works correctly and to correct any errors you find."

The frontside of this description describes how the spreadsheet should work.

Also, if you turn to the backside of this sheet (*turn over your description*), you'll see that two correct sample report cards are provided to you. You can use these to help you in your task.

Remember, your task is to test the spreadsheet, and correct any bugs you find. To help you do this, use the checkmarks by left-clicking cell decision boxes, and use the X-marks by right-clicking decision boxes.

Start your task now, and I'll tell you when time is up.

(*Task is 22 minutes*)

Number: _________

Post-session Questionnaire (Gradebook)

I. Circle the answer corresponding to how much you agree or disagree with the following statements.

1. I am confident that I found all the bugs in the Gradebook spreadsheet? (circle one)

Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree

2. I am confident that I fixed all the bugs in the Gradebook spreadsheet? (circle one)

Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree

II. How much additional time would you need to complete this task?

_____ None. It only took me _____ minutes.

_____ None. I took about the entire time.

_____ I would need about _____ more minutes.

_____ I am not sure.

Now that you have worked with the software mark your answers for the following:

I could find and fix errors in a spreadsheet <u>using this software</u> …

... if there was no one around to tell me what to do as I go.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had never used a spreadsheet like it before.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had only the software manuals for references.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had seen someone else using it before trying it myself.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I could call someone for help if I got stuck.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if someone else had helped me get started.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had a lot of time to complete the task.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had just the built-in help facility for assistance.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if someone showed me how to do it first.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
... if I had used similar spreadsheets before this one to do this same task.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree

Payroll.frm

Here is a payroll spreadsheet problem. Let's read the second paragraph at the top of the description:

"Your task is to test the updated spreadsheet to see if it works correctly and to correct any errors you find."

The frontside of this description describes how the spreadsheet should work.

Also, if you turn to the backside of this sheet (*turn over your description*), you'll see that two correct sample payroll stubs are provided to you. You can use these to help you in your task.

Remember, your task is to test the spreadsheet, and correct any bugs you find. To help you do this, use the checkmarks by left-clicking cell decision boxes, and use the X-marks by right-clicking decision boxes.

Start your task now, and I'll tell you when time is up.

(*Task is 35 minutes*)

Number: _________

Post-session Questionnaire (Payroll)

I. Circle the answer corresponding to how much you agree or disagree with the following statements.

3. I am confident that I found all the bugs in the Payroll spreadsheet? (circle one)

Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree

4. I am confident that I fixed all the bugs in the Payroll spreadsheet? (circle one)

Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree

3. How much additional time would you need to complete this task?

_____ None. It only took me _____ minutes.

_____ None. I took about the entire time.

_____ I would need about _____ more minutes.

_____ I am not sure.

(please turn page over)

4. Mark how you found the following features for **finding and fixing errors**:

Cell border colors helped me make progress	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
Interior Cell Coloring (yellow and red) helped me make progress	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
X-marks helped me make progress	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
Checkmarks (√)helped me make progress	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
Pop up messages helped me make progress	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
Arrows helped me make progress	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
Percent tested indicator helped me make progress	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
Bug likelihood bar helped me make progress	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree

4b. Rank your preference for the following features (**1 – most preferred feature; 2 – 2nd most preferred feature; 3 – 3rd most preferred feature; and so on**):

_______ Cell border colors

_______ Interior cell colorings

_______ X-marks

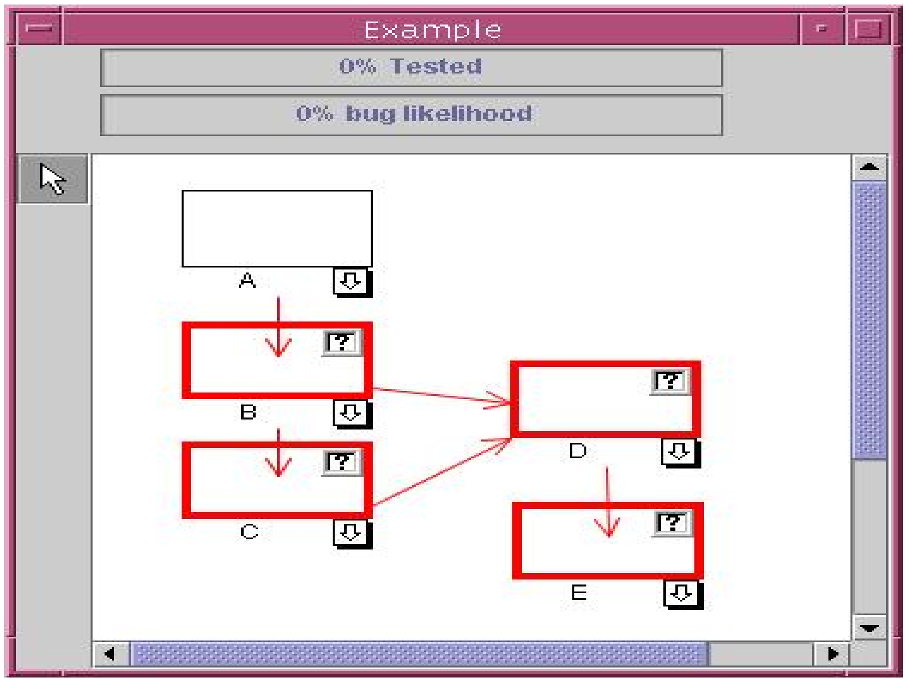
_______ Checkmarks

_______ Pop-up messages

_______ Arrows

_______ Percent testedness indicator

_______ Bug likelihood bar



Q5 to Q10: Refer to the Figure Above and choose your answers from the choices below.

One or more Questions can have the same answer.

5. If we place an X- mark in cell D the color of the cell D:
   a. Remains the same
   b. Gets darker
   c. Gets lighter
   d. Don't know

6. If we place an X- mark in cell D the color of the cell C
   a. Remains the same
   b. Gets darker
   c. Gets lighter
   d. Don't know

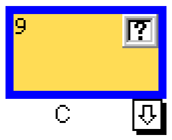
7. If we place an X- mark in cell D the color of the cell E
   a. Remains the same
   b. Gets darker
   c. Gets lighter
   d. Don't know

Assume for the next three Questions (8-10) that an X- mark has been placed on the cell D.

8. If we place an X- mark in cell C the color of the cell C
   a. Remains the same
   b. Gets darker
   c. Gets lighter
   d. Don't know

9. If we place an X- mark in cell C the color of the cell B
   a. Remains the same
   b. Gets darker
   c. Gets lighter
   d. Don't know

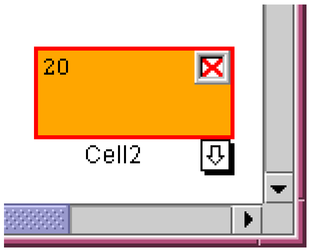
10. If we place a Checkmark in cell C the color of the cell D
   a. Remains the same
   b. Gets darker
   c. Gets lighter
   d. Don't know



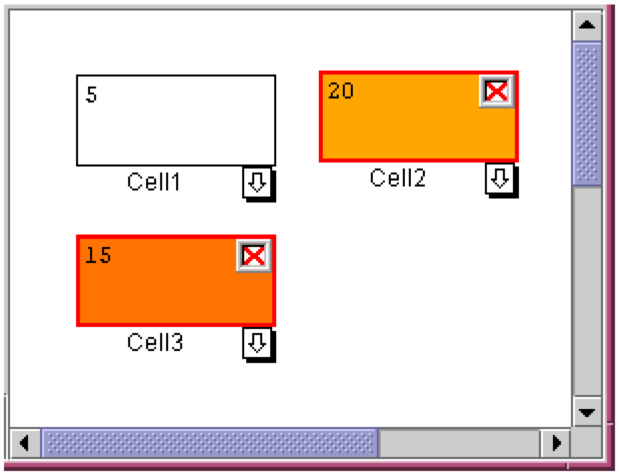
11. What does a blue border of a cell with a yellow-orange interior mean (refer to above figure)? (Circle 1 option for each part)

a) The value is: (circle 1)	CORRECT	WRONG	COULD BE EITHER
b) The cell is: (circle 1)	TESTED	UNTESTED	COULD BE EITHER
c) The cell has: (circle 1)	BUG LIKELIHOOD	NO BUG LIKELIHOOD	COULD BE EITHER
d) My answers to a, b, and c are just guesses.	YES, JUST GUESSES	NO, NOT GUESSES	
e) The combination of blue border and yellow-orange interior colors on this cell: (circle 1)	MAKES SENSE	MAKES NO SENSE	NOT SURE

12. What does the X- mark in the decision box mean?



13. In the above figure what does the orange color in the interior of the cell mean?



14. In the above figure what does it mean when the colors in the interior of one cell is darker the others?

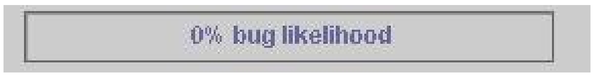
Please provide any other general comments you may have regarding the cell interior colorings:

____

____

____

____



15. In the above figure what does the bug likelihood bar mean?

Please provide any other general comments you may have regarding the bug likelihood bar:

____
____
____
____

(please turn page over)

Did you place X marks? If yes answer Question 16, otherwise answer Question 17

16. When I placed an X mark…

… the computer made bad decisions with them.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
… I worried they would distract me from my original goal.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
… I was afraid that I would not use them properly.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
… it seemed like they were causing problems with the spreadsheet.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
… I worried that they would not help achieve my goal(s).	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
… I was afraid I would take too long to learn them.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree

17. I did not place X marks because…

… the computer would make bad decisions with them.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
… I worried they would distract me from my original goal.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
… I was afraid that I would not use them properly.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
… it seemed like they could cause problems with the spreadsheet.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
… I worried that they would not help achieve my goal(s).	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree
… I was afraid I would take too long to learn them.	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree

18. If there are still errors in the spreadsheet this is because… (Circle **1** reason you agree with most)

a. The computer should have helped me spot the errors
b. I should have spent more time trying to find the errors
c. There was not enough time
d. None of the above

Appendix B: Spreadsheets and Spreadsheet Descriptions

PURCHASE BUDGET

You are in charge of ordering office supplies for the office you work at. You must order enough pens and paper to have on hand, but you cannot spend more than your allotted budget for office supplies.

You must keep more than 68 boxes of pens and 400 reams of paper on hand and you cannot exceed a budget of $2000.

Pen and Paper
The quantity of pens and paper that you are ordering and the quantity you have on hand.

Costs of Pen and Paper
The cost of pens is $2 per box, and the cost of paper is twice that, $4.

Pen and Paper Check
These cells are used to check to ensure you are ordering enough pens and paper to restock the shelves.

Total Cost
The total cost is the combined cost of pens and paper. The BudgetOK cell determines if you went over your allotted budget.

Task: Test the spreadsheet to see if it works correctly and correct any errors you find.

Example data for correct spreadsheet

Pens	0
Paper	400
PensOnHand	25
PaperOnHand	21
PenTotalCost	0
PaperTotalCost	1600
PenQCheck	not enough pens
PaperQCheck	paper quantity ok
TotalCost	1600
BudgetOK?	Budget ok

GRADEBOOK SPREADSHEET PROBLEM

Another teacher has updated a spreadsheet program that computes the course grade of a student. Two correct sample report cards and information about the class' grading policy are provided.
Your task is to test the updated spreadsheet to see if it works correctly and to correct any errors you find.

Quizzes
There are five quizzes. The lower of the first two quiz scores is dropped. The average quiz score is then the average of the highest four quiz scores.

Midterm Exams
There are three midterms. The first midterm has 50 possible points; however, it must be adjusted to a "0-100" percentage scale. The third midterm score is curved; students receive a two-point bonus if their score is not zero.

The lower of the first two midterm scores is dropped. The average midterm score is then the average of the third midterm and the higher of the first two midterm scores.

Final Exam
There is one final exam. It has 146 possible points. It must be adjusted to a "0-100" percentage scale.

Exam Average
The exam average is the average of three scores: the two highest midterm scores and the final exam score.

Course Average
Quizzes are worth 40% of a student's grade. Midterms are worth 40% of a student's grade. The final exam is worth 20% of a student's grade.

Course Grade
A student's course grade is determined by their course average, in accordance with the following scale:

90 and up : A 80 - 89 : B	70 - 79 : C 60 - 69 : D Below 60 : F

Example Correct Gradebook Report Cards

John Doe	**Report Card**
Quiz1	81.25
Quiz2	100
Quiz3	100
Quiz4	96
Quiz5	100
Quiz_Average	99
Midterm1 (Original)	45
Midterm2	96
Midterm3 (Original)	80
Midterm_Average	89
Final	129
Final_Percentage	88.36
Course_Avg	92.87
Course_Grade	A

Mary Smith	**Report Card**
Quiz1	0
Quiz2	88.24
Quiz3	85
Quiz4	87
Quiz5	100
Quiz_Average	90.06
Midterm1 (Original)	24
Midterm2	61
Midterm3 (Original)	66
Midterm_Average	64.5
Final	106
Final_Percentage	72.6
Course_Avg	76.34
Course_Grade	C

List of bugs in the Gradebook spreadsheet

The Gradebook spreadsheet was seeded with five faults created by real end users.

Table 6: List of bugs in Gradebook spreadsheet: Output cells with their formulas when the spreadsheet is first loaded. (Note: All the other input cells have a value 0)

Cellname	**Original Formula**	**Correct Formula**
Curved_Midterm3	if Midterm3 > 0 then 2 else 0	if Midterm3 > 0 then Midterm3+2 else 0
Quiz_Avg	((Quiz1 + Quiz2 + Quiz3 + Quiz4 + Quiz5) - Min_Quiz1_Quiz2) / 5	((Quiz1 + Quiz2 + Quiz3 + Quiz4 + Quiz5) - Min_Quiz1_Quiz2) / 4
Midterm_Avg	Midterm1_Perc + Midterm2 + Curved_Midterm3 - Min_Midterm1_Midterm2 / 2	(Midterm1_Perc + Midterm2 + Curved_Midterm3 - Min_Midterm1_Midterm2) / 2
Exam_Avg	(Midterm_Avg + Final_Percentage) / 3	(Midterm_Avg*2 + Final_Percentage) / 3
Course_Avg	(Quiz_Avg * 0.4) + (Midterm_Avg * 0.4) + (Final_Percentage * 0.2) / 10	(Quiz_Avg * 0.4) + (Midterm_Avg * 0.4) + (Final_Percentage * 0.2)

Table 7: Formula’s of output cells in the Gradebook spreadsheet (Note: All the other input cells have a value 0)

Cell Name	Original Formula

Min_Quiz1_Quiz2	if (Quiz1 < Quiz2) then Quiz1
	else Quiz2

Midterm1_Perc	2 * Midterm1

Min_Midterm1_Midterm2	if (Midterm1_Perc < Midterm2) then Midterm1_Perc
	else Midterm2

Curved_Midterm3	if Midterm3 > 0
	then 2
	else 0

Final_Percentage	Final / 146 * 100
Quiz_Avg	((Quiz1 + Quiz2 + Quiz3 + Quiz4 + Quiz5) - Min_Quiz1_Quiz2) / 5
Midterm_Avg	Midterm1_Perc + Midterm2 + Curved_Midterm3 - Min_Midterm1_Midterm2 / 2
Exam_Avg	(Midterm_Avg + Final_Percentage) / 3
Course_Avg	(Quiz_Avg * 0.4) + (Midterm_Avg * 0.4) + (Final_Percentage * 0.2) / 10

Course_Grade	if Course_Avg >= 90 then "A"
	else
	(if Course_Avg >= 80 then "B"
	else
	(if Course_Avg >= 70 then "C"
	else
	(if Course_Avg >= 60 then "D"
	else "F")))

PAYROLL SPREADSHEET PROBLEM

- A spreadsheet program that computes the net pay of an employee has been updated by one of your co-workers.
- Below is a description about how to compute the answers.
- On the backside of this sheet are two correct examples, which you can compare with the values on screen.

Your task is to test the updated spreadsheet to see if it works correctly and to correct any errors you find.

FEDERAL INCOME TAX WITHHOLDING

To determine the federal income tax withholding:

1. From the monthly adjusted gross pay subtract the allowance amount (number of allowances claimed multiplied by $250). Call this amount the adjusted wage.
2. Calculate the withholding tax on adjusted wage using the formulas below:
    a. If Single and adjusted wage is not greater than $119, the withholding tax is $0; otherwise the withholding amount is 10% of (adjusted wage – $119).
    b. If Married and adjusted wage is not greater than $248, the withholding tax is $0; otherwise the withholding amount is 10% of (adjusted wage – $248).

SOCIAL SECURITY AND MEDICARE

Social Security and Medicare is withheld at a combined rate of 7.65% of Gross Pay. The Social Security portion (6.20%) will be withheld on the first $87,000 of Gross Pay, but there is no cap on the 1.45% withheld for Medicare.

INSURANCE COSTS

The monthly health insurance premium is $480 for Married and $390 for Single. Monthly dental insurance premium is $39 for Married and $18 for Single. Life insurance premium rate is $5 per $10,000 of insurance. The monthly employer insurance contribution is $520 for Married and $300 for Single.

ADJUSTED GROSS PAY

Pretax deductions (such as child care and employee insurance expense above the employer's insurance contribution) are subtracted from Gross Pay to obtain Adjusted Gross Pay.

Example Correct Payroll Stubs

John Doe		Month	Year-To-Date
Marital Status – Single			
Allowances	1		
Gross Pay		6,000.00	54,000.00
Pre-Tax Child Care		0.00	
Life Insurance Policy Amount		10,000	
Health Insurance Premium	390.00		
Dental Insurance Premium	18.00		
Life Insurance Premium	5.00		
Employee Insurance Cost	413.00		
Employer Insurance Contribution	300.00		
Net Insurance Cost		113.00	
Adjusted Gross Pay		5,887.00	
Federal Income Tax Withheld	551.80		
Social Security Tax	372.00		
Medicare Tax	87.00		
Total Employee Taxes		1,010.80	
Net Pay		4,876.20	

Mary Smith		Month	Year-To-Date
Marital Status – Married			
Allowances	5		
Gross Pay		8,000.00	72,000.00
Pre-Tax Child Care		400.00	
Life Insurance Policy Amount		50,000	
Health Insurance Premium	480.00		
Dental Insurance Premium	39.00		
Life Insurance Premium	25.00		
Employee Insurance Cost	544.00		
Employer Insurance Contribution	520.00		
Net Insurance Cost		24.00	
Adjusted Gross Pay		7,576.00	
Federal Income Tax Withheld	607.80		
Social Security Tax	496.00		
Medicare Tax	116.00		
Total Employee Taxes		1,219.80	
Net Pay		6,356.20	

List of bugs in the Payroll spreadsheet

The Payroll spreadsheet was seeded with five faults created by real end users.

Table 8: List of bugs in Payroll spreadsheet: Output cells with their formulas when the spreadsheet is first loaded. (Note: All the other input cells have a value 0)

Cellname	**Original Formula**	**Correct Formula**
SingleWithHold	if AdjustedWage < 119 then 0 else (AdjustedWage -248) *.10	if AdjustedWage < 119 then 0 else (AdjustedWage -119) *.10
MarriedWithHold	if GrossPay < 248 then 0 else (GrossPay – 248)*.10	if AdjustedWage < 248 then 0 else (AdjustedWage - 248)*.10
SocSec	if GrossOver87K = 0 then (GrossPay * 0.062 * 0.0145) else (87000 * GrossPay * 0.062 * 0.0145)	if GrossOver87K = 0 then (GrossPay * 0.062) else (87000 * 0.062)
SocSec	if GrossOver87K = 0 then (GrossPay * 0.062 * 0.0145) else (87000 * GrossPay * 0.062 * 0.0145)	if GrossOver87K = 0 then (GrossPay * 0.062) else (87000 * 0.062)
AdjustedGrossPay	GrossPay - PreTax_Child_Care – EmployeeInsurCost	GrossPay - PreTax_Child_Care - NetInsurCost

Table 9: Formula's of output cells in Payroll spreadsheet when it is first loaded. (Note: All the other input cells have a value 0)

Cell Name	**Original Formula**
FedWithHoldAllow	Allowances * 250
AdjustedWage	AdjustedGrossPay - FedWithHoldAllow

SingleWithHold	if AdjustedWage < 119
	then 0
	else (AdjustedWage -248) *.10

MarriedWithHold	if GrossPay < 248
	then 0
	else (GrossPay - 248)*.10

FedWithHold	if (MStatus = "Single") then SingleWithHold else MarriedWithHold
NewYTDGrossPay	YTDGrossPay + GrossPay

GrossOver87K	if NewYTDGrossPay > 87000
	then NewYTDGrossPay - 87000
	else 0

SocSec	if GrossOver87K = 0
	then (GrossPay * 0.062 * 0.0145)
	else (87000 * GrossPay * 0.062 * 0.0145)

Medicare	GrossPay *.0145
LifeInsurPremium	LifeInsurAmount *.0005

HealthInsurPremium	if MStatus="Married"
	then 480
	else 390

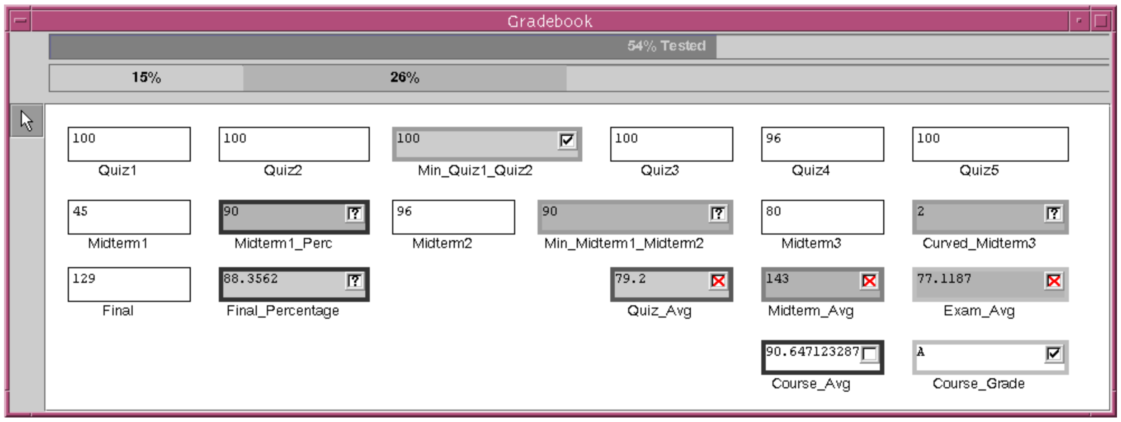
DentalInsurPremium	if MStatus = "Married"
	then 39
	else 18

AdjustedGrossPay	GrossPay - PreTax_Child_Care - EmployeeInsurCost
EmployeeInsurCost	HealthInsurPremium + LifeInsurPremium + DentalInsurPremium
EmployerInsurContrib	if MStatus = "Married" then 520 else 300

NetInsurCost	if EmployeeInsurCost > EmployerInsurContrib
	then EmployeeInsurCost - EmployerInsurContrib
	else 0

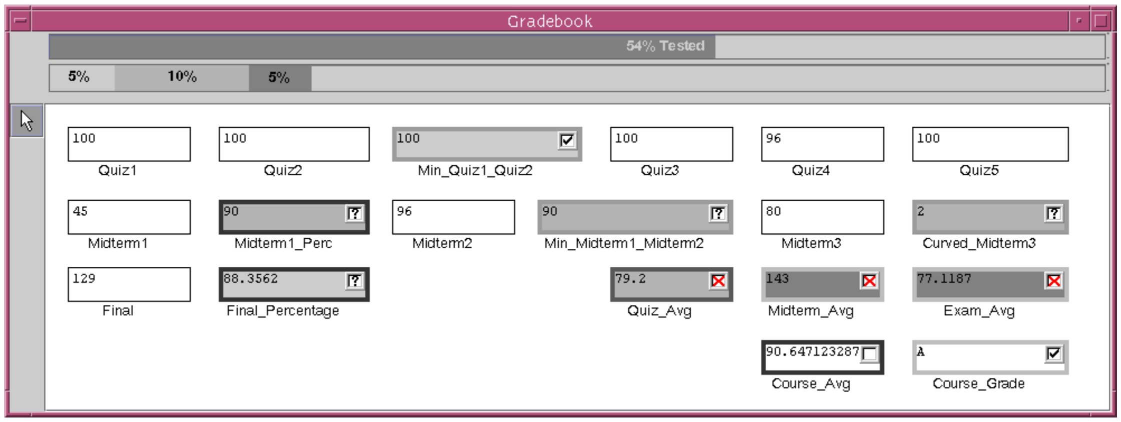
EmployeeTaxes	SocSec + Medicare + FedWithHold
NetPay	AdjustedGrossPay - EmployeeTaxes
Mstatus (Input cell, the formula is a constant)	Single

Appendix C: Visual Feedback provided by Test Count and Test Count Local Fault Localization Algorithms



Quiz1: 100	Quiz2: 100	Min_Quiz1_Quiz2: 100	Quiz3: 100	Quiz4: 96	Quiz5: 100
Midterm1: 45	Midterm1_Perc: 90	Midterm2: 96	Min_Midterm1_Midterm2: 90	Midterm3: 80	Curved_Midterm3: 2
Final: 129	Final_Percentage: 88.3562		Quiz_Avg: 79.2	Midterm_Avg: 143	Exam_Avg: 77.1187
				Course_Avg: 90.647123287	Course_Grade: A

Figure 10: Visual feedback with Test Count fault localization algorithm.



Quiz1: 100	Quiz2: 100	Min_Quiz1_Quiz2: 100	Quiz3: 100	Quiz4: 96	Quiz5: 100
Midterm1: 45	Midterm1_Perc: 90	Midterm2: 96	Min_Midterm1_Midterm2: 90	Midterm3: 80	Curved_Midterm3: 2
Final: 129	Final_Percentage: 88.3562		Quiz_Avg: 79.2	Midterm_Avg: 143	Exam_Avg: 77.1187
				Course_Avg: 90.647123287	Course_Grade: A

Figure 11: Visual feedback with the Test Count Local fault localization algorithm. Observe that, in this example, the faulty cells, Midterm_Avg and Exam_Avg cells, are colored darker when compared to the feedback in Figure 10 based on the Test Count algorithm.