Interactive Fault Localization Techniques to Empower the Debugging Efforts of End-User Programmers

Joseph R. Ruthruff and Margaret M. Burnett

July 23, 2004

Abstract

End users develop more software than any other group of programmers, using software authoring devices such as e-mail filtering editors, by-demonstration macro builders, and spreadsheet environments. Despite this, there has been only a little research on finding ways to help these programmers with the dependability of the software they create. We have been working to address this problem in several ways, one of which includes supporting end-user debugging activities through interactive fault localization techniques. This thesis investigates these fault localization techniques in the realm of end-user programming. We investigate a technique previously described in the research literature, and two new techniques that are introduced in this thesis. This thesis also presents the results of two empirical studies to examine whether fault localization techniques are effective in end-user testing and debugging tasks. The first study compares how well the three techniques isolate the faults in two end-user programs. The second study examines the impact of two orthogonal factors on the effectiveness of fault localization techniques. Our results reveal several insights into the contributions such techniques can make to the end-user debugging process, and highlight key issues of interest to researchers and practitioners who may design and evaluate future fault localization techniques.

CHAPTER 1

INTRODUCTION

A quiet revolution is occurring in the world of software. Not too long ago, most software was developed primarily by professional programmers, a large portion of whom had reason to be interested in and understand software engineering theory and practice. Today, however, end-user programmers far outnumber professional programmers. It is estimated that, by next year, 55 million end users, as compared to only 2.75 million professional programmers [12], will be creating software applications such as multimedia simulations, dynamic web pages, e-mail filtering rules, and spreadsheets. The implications of this trend for the software engineering community are considerable—requiring that research emerge to provide support to the millions of end users that create software.

Is adequate support being provided to these end users? The evidence suggests that it is not. Research [10, 43, 44] has revealed that end-user spreadsheet programs, the most common type of software developed by end users, often contain an alarming number of faults. (Following standard terminology [6], in this thesis, a *failure* is an incorrect computational result, and a *fault* is the incorrect part of the program that caused the failure.) Perhaps even more disturbing, spreadsheet developers often express unwarranted confidence in the quality of these programs [22, 44]. More generally, Boehm and Basili [11] observe that 40–50% of the software created by

end users contains non-trivial faults. These faults can be serious, costing millions of dollars in some cases (e.g., [30, 43, 52]).

A problem for the software engineering community, then, is to provide end users with better support for their software development activities. For example, end-user programmers, like professional programmers, need devices for improving the quality of their software, such as testing and anomaly detection methodologies to help them detect failures, and *fault localization* devices to help them find the causes of failures. Fault localization devices for end-user programmers are the aspect of interest in this thesis.

Software engineering researchers have long recognized the importance of fault localization devices, and have invested considerable effort into bringing fault localization devices to professional programmers (e.g., [3, 14, 24, 29, 32, 39, 46, 51, 61]). However, significant differences exist between professional and end-user software development, and these differences have ramifications for fault localization devices by acting as constraints on the types of devices suitable for end users. We have divided these differences into four classes.

A first class of differences is that, unlike professional programmers, end users rarely have knowledge of software engineering theory and practice, and are unlikely to take the time to acquire it. This impacts fault localization devices because, traditionally, such devices often require at least partial knowledge of such theory to either properly employ the device or understand its feedback. For example, critical slicing [24] uses mutation-based testing, a strategy of which end-user programmers are unlikely to have any prior knowledge. This lack of knowledge could hinder end users' ability to understand why critical slicing is producing particular fault localization feedback, which in turn can result in a loss of trust in the feedback. (As [23]

explains, understanding is critical to trust, which in turn is critical to users actually believing a system's output and acting on it.) Techniques that require knowledge of software engineering theory and practice may be inherently unsuitable for end-user programmers.

A second class of differences pertains to the manner of interaction between the software developer and the programming environment. Most professional programming environments are modal, featuring separate code, compile, link, and execute modes, and separate devices for tasks such as fault localization. The lack of interaction in these environments has allowed many fault localization techniques to perform a batch processing of information before displaying feedback (e.g., [3, 32]). In contrast, end-user programming environments are usually modeless and highly interactive: users incrementally experiment with their software and see how the results seem to be working out after every change, using devices such as the automatic recalculation feature of spreadsheet environments. Techniques that perform batch processing are therefore at best unsuited, and at worst incompatible, with these types of interactive environments.

A third class of differences pertains to the amount of information available in professional versus end-user software development environments for fault localization. End users do not usually have suites of organized test cases, so large bases of information are rarely available to debugging devices such as fault localization techniques. Complicating the situation is the interactive nature of end-user debugging: end users may observe a failure and start the debugging process early—not just after some long batch of tests—at which time the system may have very little information with which to provide feedback. Techniques that require large amounts of data may be inappropriate.

A fourth class of differences pertains to a common assumption in software engineering devices created for professional programmers: that the accuracy of the information (such as testing information) provided to the devices is reliable. Evidence [45, 58] shows that end users often make mistakes when performing interactive testing and debugging. (Professional programmers are not perfect either, of course, but there is reason to hope that their own understanding of testing and their institution's testing processes render them less error-prone than end users.) Unfortunately, many fault localization techniques (e.g., [39]) cannot operate in the presence of such unreliable information. Techniques that require a high degree of reliability in data may be inherently unsuited for end-user programmers.

We have been working to bring fault localization support to end users, in ways that accommodate the foregoing considerations, as part of our *end-user software engineering* research [17, 19], prototyping our ideas in the spreadsheet paradigm because it is so widespread in practice. The concept of end-user software engineering is a holistic approach to the facets of software development in which end users engage. Its goal is to bring some of the gains from the software engineering community to end-user programming environment, *without* requiring training, or even interest, in traditional software engineering theory or practices.

Earlier work [50, 58] presented a single fault localization technique for end users that considers the four previously mentioned differences between professional and end-user programming. However, more research is needed to devise additional techniques, particularly with varying costs and benefits, for end-user programmers. Further, this single technique has been the subject of only one preliminary evaluation [45, 58] to examine interactive, human-centric issues arising in its use in debugging

tasks. More empirical investigation is needed to evaluate the effectiveness of this and other fault localization techniques.

This thesis addresses each of these concerns through three primary contributions. First, we present three unique fault localization techniques for end users—two of which are introduced for the first time in this thesis—that are cognizant to the four previously mentioned differences between professional and end-user programming. Second, we empirically investigate the effectiveness of these techniques through an experiment to inform our techniques' design. Our results suggest the possibility that there may be multiple, independent *factors* of techniques that impact effectiveness in the realm of end-user software development. An understanding of these factors and their role in technique effectiveness is necessary in order for future designers of end-user fault localization techniques to build upon principled design choices, instead of ad hoc guesses. Finally, we empirically investigate the importance of two such factors—information base and mapping—through a second experiment, and provide data on three specific information bases and three mappings. These results provide insights into the way that fault localization effectiveness needs to be measured, so as to inform others' evaluation work in end-user fault localization techniques.

The remainder of this thesis is organized as follows: Chapter 2 describes the end-user software engineering devices with which our fault localization techniques are integrated; Chapter 3 presents the fault localization technique from our previous work, introduces two new techniques for end users, and discusses previous fault localization research that is related to this thesis; Chapter 4 describes the procedures of our first experiment, as well as outlining and discussing the results of this experiment; Chapter 5 decomposes our techniques into two individual factors;

Chapter 6 describes the procedures of our second experiment, as well as outlining and discussing the results of this experiment; and Chapter 7 concludes the thesis.

CHAPTER 2

BACKGROUND

We believe that software development support can reduce the growing reliability problem in programs created by end users. Towards this end, our "end-user software engineering" strategy [17, 19] consists of a blend of components that come together seamlessly via interactive visual devices. One of these components is the "What You See Is What You Test" (WYSIWYT) testing methodology [20, 53, 54]. (Other components include an automated test case generation device [27], a test re-use strategy [28], and an approach for supporting assertions by end users [16].) Our fault localization techniques are prototyped in the spreadsheet paradigm, in conjunction with our WYSIWYT testing methodology, so we briefly describe that methodology here.

Figure 2.1 presents an example of WYSIWYT in Forms/3 [8, 15, 18], a spreadsheet language that utilizes "free-floating" cells in addition to traditional spreadsheet grids.[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 testing is performed through visual devices. In WYSIWYT, untested cells that have non-constant formulas are given a red border (light gray in this the-

[1] WYSIWYT has also been extended to the dataflow [33] and screen transition [13] paradigms.

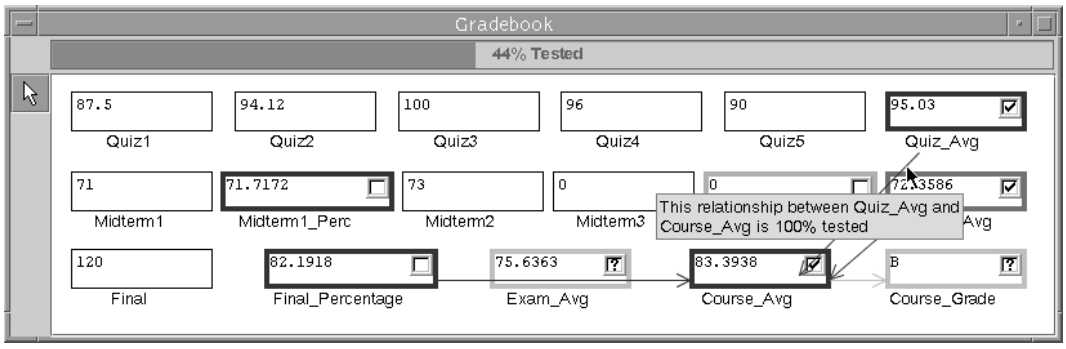

FIGURE 2.1: An example of WYSIWYT in the Forms/3 language.

sis), indicating that the cell is untested. (Cells whose formulas are simply constants do not participate in WYSIWYT devices, since the assumption is that they do not need to be tested.) For example, the Course_Grade cell has never been tested; hence, its border is red (light gray). The borders of such cells remain red until they become more "tested".

In order for cells to become more tested, users must create tests. Testing actions can occur at any time—intermingled with editing formulas, adding new formulas, and so on. The process is as follows. Whenever a user notices a correct value, he or she can place a checkmark ($\surd$) in the decision box at the corner of the cell he or she observes to be correct: this *testing decision* constitutes a successful "test". Such checkmarks increase the "testedness" of a cell, which is reflected by adding more blue to the cell's border (more black in this thesis). For example, in Figure 2.1, the Course_Avg cell has been given a checkmark, which is enough to fully test this cell, thereby changing its border from red to blue (light gray to black). Further, because a correct value in a cell c depends on the correctness of the cells contributing to c,

these contributing cells participate in c's test. Consequently, in this example, the border of cell Final_Percentage also turns blue (black).

Although users may not realize it, WYSIWYT "testedness" colors reflect the use of a definition-use test adequacy (du-adequacy) criterion [38, 41, 47] that measures the interrelationships in the source code that have been covered by the users' tests. A *definition* is a point in the source code where a variable (cell) is assigned a value, and a *use* is a point where a variable's value is used. A *definition-use pair*, or *du-pair*, is a tuple consisting of a definition of a variable and a use of that variable. A *du-adequate* test suite, which is based on the notion of an *output-influencing all-definition-use-pairs-adequate test suite* [25], is a test suite that exercises each du-pair in such a manner that it (dynamically) participates in the production of an output explicitly validated by the user. (Readers are referred to [53, 54] for details on this criterion.)

In addition to providing feedback at the cell level, WYSIWYT gives the user feedback about testedness at two other granularities. A percent testedness indicator provides testedness feedback at the program granularity by displaying a bar that fills and changes color from red to blue (following the same colorization continuum as cell borders) as the overall testedness of the program increases; in Figure 2.1, the program is 44% tested. Testedness feedback is also available at a finer granularity through dataflow arrows. In addition to displaying dataflow relationships at the cell level (in Figure 2.1, the user has triggered dataflow arrows for the Course_Avg cell), users can also display arrows at the subexpression level (shown in Figure 2.2). The system also provides testedness feedback through an intelligent explanation system [65], implemented via "on-demand" tooltips that display the testedness of any specified cell or dataflow relationship. In Figure 2.1, the user has chosen to

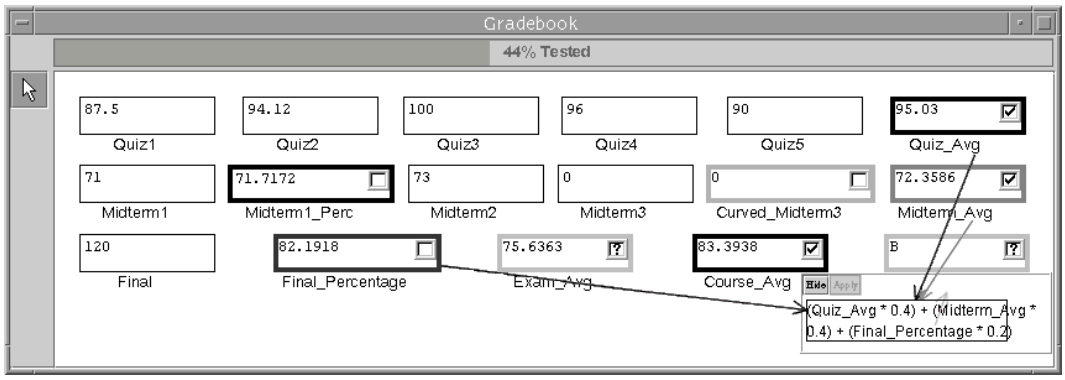

FIGURE 2.2: An example of dataflow arrows at the subexpression level.

examine a black arrow leading into Course_Avg, which shows that the relationship between Quiz_Avg and Course_Avg is 100% tested.

WYSIWYT's purpose is to help identify failures, and it has been empirically shown to be useful to both programmers and end users [37, 55]; however, it does not by itself explicitly support a debugging effort to localize the fault(s) causing an observed failure. Providing this debugging support is the aim of our fault localization techniques, which we describe next.

CHAPTER 3

FAULT LOCALIZATION

This chapter describes our approach to fault localization for end users. We begin by presenting an overview of our fault localization approach, including the goals of our design. We then present three fault localization techniques, along with accompanying algorithms. Finally, we discuss previous fault localization research that is related to this thesis.

3.1 Overview and Goals

Fault localization support attempts to help programmers locate the causes of failures in two ways: (1) by indicating the areas in a program that should be searched for a fault, thereby *reducing the search space*; and (2) by indicating the areas in a program most likely to contain a fault, thereby *prioritizing the sequence of the search* through this space.

In our particular prototype, which follows the spreadsheet paradigm[1], WYSIWYT serves as a springboard for fault localization: instead of noticing that a cell's value is correct and placing a checkmark, a user might notice that a cell's value is incorrect (a failure) and place an "X-mark".

---

[1] Our fault localization approach is also generalizable to other programming paradigms, as discussed in [58].

These X-marks trigger a *fault likelihood* estimation for each cell (with a non-constant formula) that might have contributed to the failure. Fault likelihood, updated for each appropriate cell after any testing decision or formula edit, is represented by visually highlighting the interior of suspect cells in shades of red (gray in this thesis). This serves our first goal of fault localization: reducing the user's search space.

As the fault likelihood of a cell grows, the suspect cell is highlighted in increasingly darker shades of red (gray). The darkest cells are estimated to be the most likely to contain the fault, and are the best candidates for the user to consider in trying to debug. This serves our second goal of fault localization: helping end users prioritize their search.

For example, suppose that after working a while, the user has gotten the Gradebook program to the stage shown at the top of Figure 3.1. At that point, the user notices that values of the Exam_Avg and Midterm1_Perc cells are incorrect: the values are obviously too high. Upon spotting these failures, the user places X-marks in the two cells, triggering fault likelihood estimations for all cells whose values dynamically contributed to the values in Exam_Avg and Midterm1_Perc.[2] The results of these fault likelihood estimations are communicated to the user by highlighting the interiors of cells suspected of containing faults (i.e., with non-zero estimated fault likelihood) in red (gray). The cells deemed (by the system) most likely to contain the fault (i.e., have higher estimated fault likelihood) are highlighted the darkest. An example is shown at the bottom of Figure 3.1. The Midterm1_Perc has an estimated

---

[2] In fact, in Figure 3.1, the fault is an incorrect mathematical operation in the Midterm1_Perc cell (not shown in the figure).

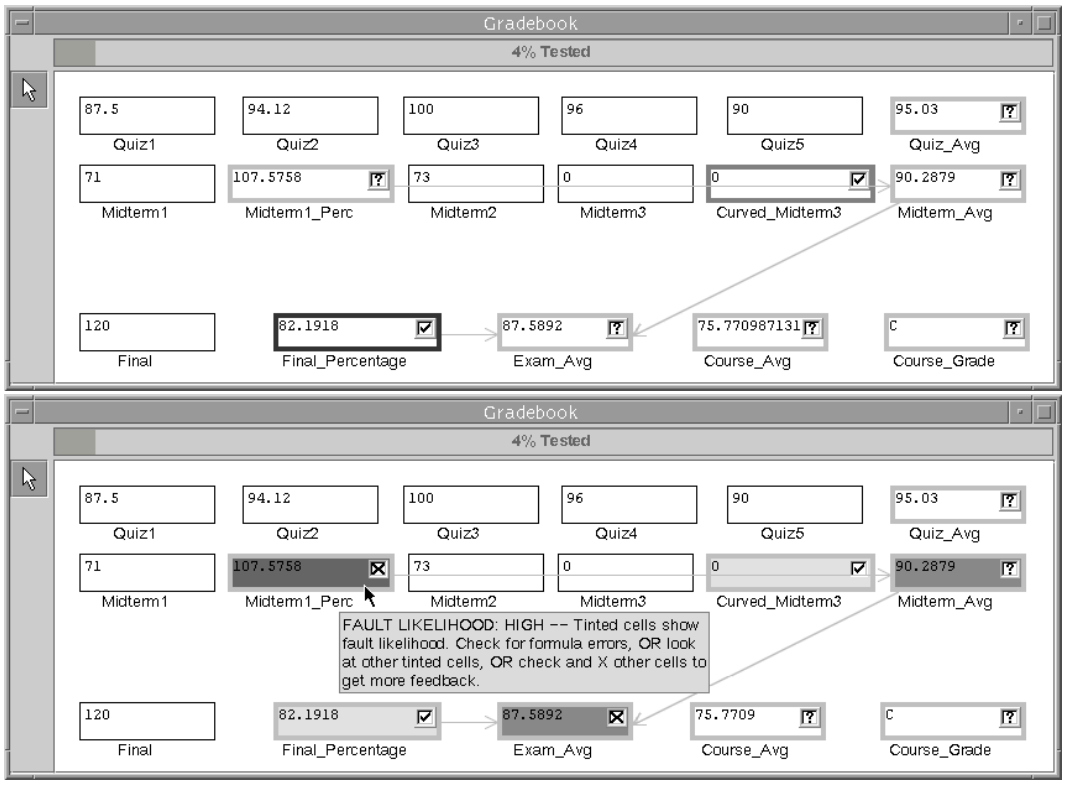

FIGURE 3.1: (Top) A Gradebook program at an early stage of testing. (Bottom) The user notices incorrect values in Exam_Avg and Midterm1_Perc—the values are obviously too high—and places an X-mark in each cell's decision box.

fault likelihood of "High" (indicated by the explanation tooltip), Midterm_Avg, and Exam_Avg cells have an estimated fault likelihood of "Medium" (not indicated by tooltips), and the Curved_Midterm3 and Final_Percentage cells have an estimated fault likelihood of "Very Low" (again, not indicated by tooltips). The testedness

borders of the cells with non-zero fault likelihood have faded to draw visual focus to the fault likelihood component of the cell.[3]

3.2 Three Fault Localization Techniques

How should these fault localization colors be computed? Computing exact fault likelihood values for a cell, of course, is not possible. Instead, we combine heuristics with deductions that can be drawn from analyzing the source code (formulas) and/or from the user's tests to estimate fault likelihood. Because these techniques are meant for highly interactive visual environments, we are also interested in the cost of each technique.

We now describe the algorithms for three fault localization techniques, including their information basis for making deductions; the algorithms and properties each technique uses or maintains, if any, in drawing further inferences from these deductions; and how much each technique's reasoning costs. In these descriptions, we will use producer-consumer terminology to keep dataflow relationships clear; that is, a *producer* of a cell c contributes to c's value, and a *consumer* of c uses c's value. Using slicing terminology [64], producers are all the cells in c's backward slice, and consumers are all the cells in c's forward slice. c is said to *participate in a test* (or to *have a test*) if the user has made a testing decision (e.g., with a checkmark) on c or any of c's consumers.

---

[3] Emerging work [57] has suggested that certain facets of debugging may be improved by not fading these borders.

3.2.1 The Blocking Technique

Similar to program dicing [21, 39], the "Blocking Technique"[4] notices the dataflow relationships existing in the testing decisions that *reach* a cell c, and whether tests are *blocked* from c by one or more other tests. It leverages these dataflow relationships with the testing information to estimate the likelihood that each cell in a program contains faults. Specifically, the technique observes the number of passed and failed tests that reach c (i.e., are not blocked from c by one or more other tests). This is accomplished by maintaining, for each cell c, (1) information on the tests to which that cell contributes, and (2) dataflow information to determine which of those tests are blocked and unblocked from the cell.

There are three basic interactions the user can perform that could potentially trigger action by the Blocking Technique. The interactions are: (1) users might change a constant cell's value (analogous to changing test cases by running with a different input value), (2) users might place an X-mark or checkmark in a cell's testing decision box (analogous to adding another test), or (3) users might add or modify a non-constant cell's formula (analogous to changing the program's logic).[5]

We describe the algorithms invoked in the case of each interaction, and also the cost of these algorithms. In this and all future discussions, we focus on the case of

---

[4] This technique was introduced by Reichwein et al. [50] as a debugging methodology for spreadsheet programs, and complete algorithms appear in [49]. We summarize Reichwein's algorithms in this thesis to present a full account of the material.

[5] Variations on these interactions also trigger activity. For example, removing a test (e.g., removing a checkmark) constitutes a change in the testing information base of the system, thereby requiring the same type of action as if a test had been added. However, we will ignore these variations because they are obvious and do not add materially to this discussion.

a “bare” spreadsheet language, without the support of a testing methodology such as WYSIWYT and its accompanying data structures. We discuss this case because fault localization support may be desired without accompanying testing support. As such, this discussion goes to the heart of the cost of the fault localization technique.

3.2.1.1 Changing Test Cases

Suppose that the user changes a constant cell’s value. A constant cell edit already requires the spreadsheet environment to update the values of any non-constant cells affected by the edit. The Blocking Technique piggy-backs off of this work to remove all of the affected testing decisions on cells (although their effects on fault likelihood information are preserved). This also happens when a cell is affected by a change in the spreadsheet program’s logic. This interaction adds only $O(1)$ overhead to the work already being triggered by the user’s edit.

3.2.1.2 Making a Testing Decision

When X-marks or checkmarks are placed by the user, the Blocking Technique must perform maintenance on its data structures to account for the testing decision that was placed or removed, as well as to determine which tests are blocked or unblocked by other tests. To do this, the technique’s algorithm (see Figure 3.2) makes three passes over the dynamic backward slice of the cell in which a testing decision was made. The first two of these three passes are separately detailed in Figures 3.3 and 3.4, and are responsible for updating data structures. The third pass, which is explained in Section 3.2.1.4, is responsible for mapping the information base of the

```
1: procedure MarkPlaced (markedCell, aTest)
2:     Let markedCell's current testing decision be aTest
3:     FirstPass(markedCell, topologicalCellList by Ref)
4:     Remove first element from topologicalCellList
5:     Let updateGUIList = SecondPass(topologicalCellList, markedCell, aTest)
6:     Estimate fault likelihood and update user interface for every cell in updateGUIList
       {This is the third pass}
7: end procedure
```

FIGURE 3.2: Algorithm #1 — The MarkPlaced subroutine for the Blocking Technique's algorithm. This is called when a testing decision is made on or removed from a cell (e.g., when the user adds a checkmark).

```
1: procedure FirstPass (aCell, topologicalCellList by Ref)
2:     if aCell has not yet been visited by FirstPass then
3:         for all aUpwardCell that dynamically affect aCell do
4:             FirstPass(aUpwardCell, topologicalCellList by Ref)
5:         end for
6:         Add aCell to beginning of topologicalCellList
7:     end if
8: end procedure
```

FIGURE 3.3: Algorithm #2 — The FirstPass subroutine of the Blocking Technique. FirstPass builds topologicalCellList.

technique into a fault likelihood range for each cell in updateGUIList and updating the user interface accordingly.

The job of FirstPass is to build a "cells-to-process" list in topological order. It performs a recursive depth-first search on the dynamic backward slice of the cell marked with the aTest testing decision. In doing so, it visits (in topological order) the cells in the dynamic backward slice, adding them into the topologicalCellList,

```
 1: function SecondPass (topologicalCellList, markedCell, aTest)
 2:    Add markedCell to updateGUIList
 3:    if aTest was added to markedCell by user then
 4:       Record that aTest blocks all tests reaching markedCell, and add aTest to marked-
          Cell
 5:    else
 6:       {aTest was removed}
 7:       Unblock tests formerly blocked by aTest, and remove aTest from markedCell
 8:    end if
 9:    Propagate markedCell's tests to create dynamically affecting cells' workLists
10:    for all sortedCell in topologicalCellList do
11:       Add sortedCell to updateGUIList
12:       if sortedCell currently has a testing decision (i.e., an X-mark or checkmark) then
13:          Make any test in sortedCell's workList that is new to sortedCell a blocked test
             if that test is blocked by sortedCell's testing decision.
14:       end if
15:       Copy sortedCell's workList changes to sortedCell's tests
16:       Propagate sortedCell's workList changes to the workLists of affecting cells
17:    end for
18:    return updateGUIList
19: end function
```

FIGURE 3.4: Algorithm #3 — The SecondPass function of the Blocking Technique. This function updates the technique's information base of tests to which cells contribute. These tests are later used in line 6 of Figure 3.2 to estimate each cell's fault likelihood.

which is returned to the algorithm in Figure 3.2. The cell on which the aTest testing decision was made (i.e., the cell actually marked by the user) is the first element in the stack. This first element is not needed by SecondPass, so it is discarded in Figure 3.2.

SecondPass is responsible for maintaining the "blocking" characteristics of the Blocking Technique. Its job is to propagate the testing decision to the cellList returned by FirstPass, and to track whether that decision is blocked by (or blocks)

another testing decision. This is done by using a (temporary) workList for each cell. A workList tracks the "blocking" changes being made to its respective cell due to this new testing decision.[6] The blocked and unblocked tests for markedCell are propagated to the workList of each cell dynamically affecting markedCell. When each workList has been processed and all changes are complete, the changes are applied to each cell's information base. One of the implementation details in these data structures is that when marks are blocked, the number of paths in which they are blocked is also tracked. This characteristic is used to determine when a testing decision is no longer blocked by any other testing decisions.

Let p be the number of cell c's producers—i.e., cells in c's dynamic backward slice. Furthermore, let e be the number of edges in the graph consisting of c's dynamic backward slice, and let m be the total number of testing decisions (X-marks and checkmarks) in the program's history. The first pass (Figure 3.3) of the MarkPlaced algorithm in Figure 3.2 is simply a depth-first search, and therefore has $O(p + e)$ runtime complexity. The second pass (Figure 3.4) iterates over the p producing cells to propagate tests to the work lists of the producing cells using the e edges, and then to process those work lists. When propagating tests to work lists (line 8 of Figure 3.4), insertion and removal operations are performed on sets of blocked and unblocked tests. These operations consist of inserting (or removing) the dataflow paths from one testing decision in m to potentially all other decisions in m. Because as many as m tests could be propagated to a workList and require

---

[6] Temporary work lists are used, rather than propagating markCell's tests straight to affecting cell's information base of tests for efficiency reasons. This strategy allows the technique to process only those tests from markedCell, and not all the tests of each sortedCell in topologicalCellList.

m insertions or removals of dataflow paths, the worst case performance of these operations is $O(m^2)$. As a result, the second pass of this algorithm has a worst-case runtime complexity of $O((p+e)m^2)$. The third pass updates the user interface for all p cells, thereby having a complexity of $O(p)$. The runtime complexity of making a testing decision is dominated by the second pass. Therefore, the worst-case runtime complexity of making a testing decision is $O((p + e)m^2)$.

3.2.1.3 Changing the Spreadsheet Logic

When the user edits a non-constant formula or changes a constant formula into a non-constant formula, the spreadsheet program's logic is changed. In this case, the NewFormula algorithm, which is outlined in Figure 3.5, is invoked. This algorithm requires the cell in which the edit was made, and the cell's new formula. It is responsible for removing the effects of all testing decisions to which the cell contributes. As before, the actual fault likelihood estimation for each applicable cell is made when the user interface for those cells is updated (in line 8 of the algorithm).

In Figure 3.5, the UndoTestEffects function performs a recursive, depth-first search on the cells that statically affect aMarkedCell, where aMarkedCell is a cell on which a testing decision was made that affects editedCell, which was given the new formula. UndoTestEffects removes all testing decisions to which the edited cell contributes, as well as the effects of those decisions on affecting cells. (Obviously, the algorithm does not remove the effects of a testing decision that was placed on a value that was not affected by the edited cell.) For efficiency, this walk is careful not to visit the same cell's formula data structure twice during a single invocation of NewFormula.

```
 1: procedure NewFormula (editedCell)
 2:     if editedCell contributes to any blocked or unblocked tests then
 3:         Let allEditedCellTests be all the testing decisions affecting editedCell
 4:         for all aTest in allEditedCellTests do
 5:             Call UndoTestEffects(aMarkedCell, editedCell, allEditedCellTests), where
                aMarkedCell is where aTest was made, and add returned lists to updateGUIList
 6:         end for
 7:     end if
 8:     Estimate fault likelihood and update user interface for every cell in updateGUIList
 9: end procedure
10:
11: function UndoTestEffects (aMarkedCell, editedCell, allEditedCellTests)
12:     if aMarkedCell has already been visited in this pass then
13:         return null
14:     end if
15:     Add aMarkedCell to updateGUIList
16:     if aMarkedCell currently has a testing decision aTest in allEditedCellTests then
17:         Remove aTest from aMarkedCell
18:     end if
19:     if aMarkedCell was affected by editedCell then
20:         Remove all tests from aMarkedCell reaching editedCell, and remove the
            WYSIWYT testedness on aMarkedCell from these tests
21:     end if
22:     Call UndoTestEffects(aCellToUpdate, editedCell, allEditedCellTests) for all aCell-
        ToUpdate statically affecting aMarkedCell, and add returned lists to updateGUIList
23:     return updateGUIList
24: end function
```

FIGURE 3.5: Algorithm #4 — The NewFormula routine for the Blocking Technique, and the accompanying UndoTestEffects function.

Let p be the number of cells in c's *static* backward slice. Also, let e be the number of edges in the dataflow *multigraph*[7] consisting of c's static backward slice,

[7] The edges are those of a multigraph because a cell can be referenced multiple times in a formula.

and let m be the total number of testing decisions (X-marks and checkmarks) in the program's history. The algorithm performs a depth-first search on the cells that are statically affected by the edited cell. Because the algorithm is careful not to visit cells more than once, only p cells are visited. For each of these cells, at least one removal operation must be performed on sets of lists of tests (see line 17 of Figure 3.5). This operation has a worst-case runtime complexity of $O(m^2)$. Also, the UndoTestEffects function visits the incoming edges in the dataflow multigraph for c. The final runtime complexity of the NewFormula algorithm is therefore $O(e+pm^2)$.

3.2.1.4 Mapping Information to Estimated Fault Likelihood

The mapping behavior of the Blocking Technique's algorithms was summarized by Reichwein et al. [50] using five properties:

Property 1: *If c or any of its consumers have a failed test, then c will have non-zero fault likelihood.* This first property ensures that every cell that might have contributed to the computation of an incorrect value will be assigned some non-zero fault likelihood. This reduces the chance that the user will become frustrated searching for a fault that is not in any of the highlighted cells, which could ultimately lead to a loss of a user's trust in the system. The property also acts as a robustness feature by ensuring that (possibly incorrect) checkmarks do not bring the fault likelihood of a faulty cell to zero.

Property 2: *The fault likelihood of c is proportional to the number of c's failed tests.* This property is based on the assumption that the more incorrect calculations a cell contributes to, the more likely it is that the cell contains a fault.

Property 3: *The fault likelihood of c is inversely proportional to the number of c's successful tests.* The third property, in contrast to Property 2, assumes that the more correct calculations a cell contributes to, the less likely it is that the cell contains a fault.

Property 4: *An X-mark on c blocks the effects of any checkmarks on c's consumers (forward slice) from propagating to c's producers (backward slice).* This property is specifically to help users narrow down where the fault is located by preventing "dilution" of important clues. More specifically, producers that contribute only to incorrect values are darker, even if those incorrect values contribute to correct values further downstream. This prevents dilution of the cells' colors that lead only to X-marks. (In short, X-marks block the propagation of checkmarks.)

Property 5: *A checkmark on c blocks the effects of any X-marks on c's consumers (forward slice) from propagating to c's producers (backward slice), with the exception of the minimal fault likelihood property required by Property 1.* Similar to Property 4, this property uses checkmarks to prune off c's producers from the highlighted area if they contribute to only correct values, even if those values eventually contribute to incorrect values. (Put another way, checkmarks block most of the propagation of X-marks.)

To implement these properties, let NumBlockedFailedTests(c) (*NBFT*) be the number of cell c's consumers that are marked incorrect, but are blocked by a value marked correct along the dataflow path from c to the value marked failed. Furthermore, let NumUnblockedFailedTests(c) (*NUFT*) be the result of subtracting *NBFT* from the number of c's consumers. Finally, let there be NumBlockedPassedTests(c) (*NBPT*) and NumUnblockedPassedTests(c) (*NUPT*), with definitions similar to those above.

If c has no failed tests, the fault likelihood of c is estimated to be "None". If c has failed tests but none are reachable (i.e., unblocked), then c's fault likelihood is estimated to be "Very Low". Otherwise, the Blocking Technique first assigns a discrete "0–5" fault likelihood range to *NUFT* and *NUPT* using the scheme presented in Table 3.1.[8] The technique then maps this testing information into an estimated fault likelihood for c using the following equation:

$$fault\ likelihood(c) = max(1, NUFT - floor(NUPT/2))$$

The formula $max(1, NUFT - floor(NUPT/2))$ was chosen to fulfill Properties 2 and 3. It ensures that as the number of X-marks affected by c increases, the fault likelihood increases. In contrast, as the number of checkmarks affected by c increases, the fault likelihood decreases. The number of X-marks was given a higher weight than the number of checkmarks to prevent an equal number of X-marks and checkmarks from canceling each other out.

---

[8] Note that the "Very Low" fault likelihood range is not included in Table 3.1. Rather, that range is reserved exclusively for the "blocking" situation mentioned in Property 5.

Numerical Values	Fault Likelihood Range	Fault Likelihood Description
0	0	"None"
1–2	2	"Low"
3–4	3	"Medium"
5–9	4	"High"
10–∞	5	"Very High"

TABLE 3.1: The scheme used by the Blocking Technique to transform numerical values to discrete fault likelihood ranges. The range of 1 (not shown in this table) corresponds to the "Very Low" fault likelihood used in Property 5.

3.2.1.5 An Example of the Blocking Technique

Figure 3.6 demonstrates the Blocking Technique on the Gradebook spreadsheet program from Section 3.1. The two failures, noted with X-marks by the user, have contributed to fault likelihood estimations for the Midterm1_Perc, Exam_Avg, and Course_Avg cells. However, the strategically-placed checkmarks on Final_Percentage and Curved_Midterm3 block most of the effects of these X-marks, causing those two cells to be assigned a lower fault likelihood.

3.2.2 The Nearest Consumers Technique

In interactive end-user programming environments, fault localization feedback can be invoked during any interactive testing or debugging activity. In the case of spreadsheet environments, such *triggers* are the modification of a cell formula, and a testing decision (i.e., X-marks and checkmarks) regarding the correctness of a cell value. The cost of fault localization techniques such as the Blocking Tech-

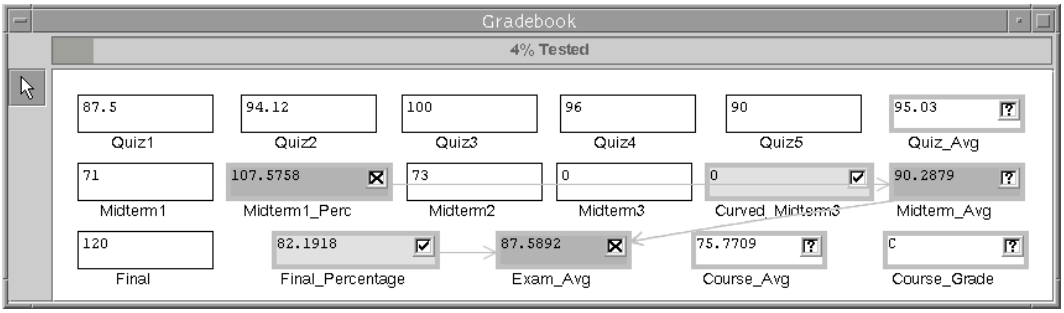

FIGURE 3.6: The Gradebook program with the Blocking Technique.

nique during these interactive activities, however, may be too great to maintain the responsiveness required as program size increases.

We designed the Nearest Consumers Technique with this concern in mind. It is a greedy technique that considers only the *direct consumers* of a cell c (those connected with c directly by a dataflow edge). Accessing only neighboring consumers is one way this techniques keeps costs down.

The fault likelihood of c is estimated solely from the X-marks and checkmarks *currently* placed on cells, and the average fault likelihood of c's direct consumers (if any). c's producers are then updated using the same calculations. However, all of these calculations use the testing decisions for only the current input values (i.e., the current test case). The technique does not utilize any historical information regarding previous testing decisions. This use of only current information is the second way this technique keeps costs down.

3.2.2.1 Changing Test Cases

Just as with the Blocking Technique, when a user changes a constant cell's value, all of the affected testing decisions on cells are removed. The Nearest Consumers Technique removes these testing decisions as the spreadsheet environment updates cell values, thereby adding only $O(1)$ overhead to this interaction.

3.2.2.2 Making a Testing Decision

When a testing decision is made, the Nearest Consumers Technique attempts to approximate the five properties of the Blocking Technique at a low cost using the algorithm outlined in Figure 3.7. Nearest Consumers begins by computing the numerical fault likelihood of cell c. In MarkPlaced_Helper, this numerical value is the average fault likelihood of c's direct consumers. This value is then mapped to a discrete fault likelihood range as outlined in Table 3.2. Note that, unlike the Blocking Technique, the fault likelihood of cells is estimated early in the algorithm rather than immediately before the user interface is updated. This is because the algorithms of the Blocking Technique are primarily designed around updating the technique's information base, and estimating fault likelihood should be done only after these information bases are updated. In contrast, Nearest Consumers relies on the fault likelihood of other cells, and so this estimation for each cell should occur immediately. (The estimation should occur immediately because producing cells' fault likelihood estimations depend on the fault likelihood of consuming cells that were previously visited by the technique.)

The AdjustFaultLikelihood subroutine adjusts the fault likelihood of aCell based on trends it observes in the current testing decisions. This subroutine implements

```
 1: procedure MarkPlaced (markedCell, aTest)
 2:     Let markedCell's current testing decision be aTest
 3:     Call MarkPlaced_Helper(markedCell, false, false)
 4:     Update the user interface for markedCell and all of markedCell's producers
 5: end procedure
 6:
 7: procedure MarkPlaced_Helper (aCell, retainHighFL?, retainLowFL?)
 8:     Let aCell's fault likelihood be the average fault likelihood of its direct consumers
 9:     Map the numeric fault likelihood of aCell to a fault likelihood range
10:     Call AdjustFaultLikelihood(aCell, retainHighFL? by ref, retainLowFL? by ref)
11:     Set retainHighFL? to true if aCell currently has an "X-mark" testing decision
12:     Set retainLowFL? to true if aCell currently has a "Checkmark" testing decision
13:     if aCell's current fault likelihood is less than its previous fault likelihood then
14:         Set aCell's fault likelihood to the previous fault likelihood if retainHighFL? is true
15:     end if
16:     if aCell's current fault likelihood is greater than its previous fault likelihood then
17:         Set aCell's fault likelihood to the previous fault likelihood if retainLowFL? is true
18:     end if
19:     Using a breadth-first search, call MarkPlaced_Helper(directProd, retainHighFL?, re-
        tainLowFL?) for all directProd in aCell's direct producers
20: end procedure
```

FIGURE 3.7: Algorithm #5 — The MarkPlaced subroutine for the Nearest Consumers Technique. MarkPlaced_Helper performs a breadth-first search of the producing cells of markedCell.

five rules, which are presented in the upcoming discussion in placed of presenting the algorithm.

The first three rules potentially adjust the fault likelihood of aCell, which was initially estimated in line 8 of Figure 3.7. It sequentially checks the conditions of each rule, and applies (fires) the first rule whose condition is met. These three rules are described next. In doing so, we use the following notation: DC is the set of a cell c's direct consumers, $avgFL(DC)$ is the average fault likelihood of DC

Numerical Values	Fault Likelihood Range	Fault Likelihood Description
0	0	"None"
1	1	"Very Low"
2	2	"Low"
3	3	"Medium"
4	4	"High"
5–∞	5	"Very High"

TABLE 3.2: The mapping used by the Nearest Consumers Technique to transform numerical values to discrete fault likelihood ranges.

(previously calculated in line 8 of Figure 3.7), xm is the number of X-marks in DC, and cm is the number of checkmarks in DC.

Rule 1. If the value of c is currently marked correct by the placement of a checkmark, c is assigned a fault likelihood of "Very Low"—regardless of $avgFL(DC)$—to approximate Property 5 of the Blocking Technique.

Rule 2. If c has a failure (X-mark) but $avgFL(DC) <$ "Medium", then c is assigned a fault likelihood of "Medium". This assignment approximates Property 4 of the Blocking Technique by preventing cells' low fault likelihood from diluting the fault likelihood of a cell in which a user has observed a failure. A common occurrence of this situation is when a user places the very first X-mark in the program's testing history in a cell c. Since it is the first X-mark placed, before the algorithm is applied, the fault likelihood of all cells, including c's direct consumers, is obviously "None".

Rule 3. The fault likelihood from $avgFL(DC)$ is incremented one level whenever there are more X-marks than checkmarks, provided sufficient evidence is present. Three specific cases are handled: (1) if $xm > 1$ and $cm = 0$, (2) if $xm > cm$ and $cm > 0$, or (3) if $xm > cm$ and c has an X-mark. The first two cases differ only in that the fault likelihood of c is not incremented if $xm = 1$ and $cm = 0$. (This is not seen as strong enough evidence to increment fault likelihood.) The third case differs from the first two by considering whether c has an X-mark. In essence, this rule increases fault likelihood in areas of the program where more failures than successes (i.e., more X-marks than checkmarks) have been observed, thereby approximating Properties 2 and 3 of the Blocking Technique.

After adjusting the fault likelihood of aCell through Rules 1–3, MarkPlaced_Helper applies two final rules to the fault likelihood. Unlike Rules 1–3, both Rules 4–5 can be applied if their conditions hold true.

Rule 4. If the value of c is currently marked as incorrect with an X-mark, the technique constrains the fault likelihood of c and c's producers to their previous estimated fault likelihood, or to higher estimations. This rule serves to help enforce Property 4 of the Blocking Technique by preventing cells with lower fault likelihood (due to checkmarks) from diluting the fault likelihood of a cell in which a failure has been observed, as well as all upstream cells.

Rule 5. Similarly, if a checkmark is currently placed in c, the technique constrains the fault likelihood of c and c's producers to their previous estimated fault likeli-

hood, or to lower estimations. This rule serves to help enforce Property 5 of the Blocking Technique by pruning off cells contributing to correct values.

The Nearest Consumer Technique enforces Property 1 of the Blocking Technique. Ensuring that any cell that could have contributed to a failure is assigned at least some fault likelihood is done by taking a ceiling of the average fault likelihood of each cell c's direct consumers on line 8 of Figure 3.7. This mathematical mechanism ensures that if any direct consumer of c has at least some fault likelihood (due to at least one X-mark), then the average of those fault likelihood values will ensure that c has at least a "Very Low" fault likelihood, as will its affecting cells in its dynamic backward slice.

The technique's advantages are that it does not require the maintenance of any data structures; it stores only the fault likelihood of cells. Given this information, after marking a cell c, estimating c's fault likelihood requires only a look at c's direct consumers. This is followed by a single $O(p+d)$ breadth-first traversal up c's dynamic backward slice to estimate the fault likelihood of these producers, where d is the number of direct consumers connected to the p producers.

3.2.2.3 Changing the Spreadsheet Logic

Unlike the Blocking Technique, Nearest Consumers does not have to worry about removing the effects of all testing decisions to which an edited non-constant cell c contributes because it does not maintain such testing decisions. The Nearest Consumers Technique must consider those cells that are affected by c because those cells will likely have new values, thereby rendering their current testing decisions

(if any) obsolete. Consequently, as the spreadsheet environment traverses down the static forward slice of c and updates cell values, Nearest Consumers removes any X-marks or checkmarks on this set of affected cells A, including c. Because the forward slice is traversed, all cells in A will have no current testing decision for the test case in question. Therefore, with no testing decision history to draw on, Nearest Consumers also resets the fault likelihood of both c and all cells in A to "None". This is all done with $O(1)$ overhead to the spreadsheet environment.

However, since c's fault likelihood has changed to "None", the cells in c's static backward slice must have their fault likelihood updated. This is performed by calling the NewFormula algorithm in Figure 3.8. Because this algorithm requires a single breadth-first traversal up c's static backward slice, the runtime complexity is $O(p + d)$, where d is the number of direct consumers connected to the p producers in the backward slice.

```
1: procedure NewFormula (editedCell)
2:     Call MarkPlaced_Helper(editedCell, false, false)
3:     Estimate fault likelihood and update the user interface for editedCell and all of edit-
       edCell's producers
4: end procedure
```

FIGURE 3.8: Algorithm #6 — The NewFormula subroutine of the Nearest Consumers Technique. This algorithm performs a breadth-first search on c's backward slice using the MarkPlaced_Helper subroutine in Figure 3.7.

3.2.2.4 Mapping Information to Estimated Fault Likelihood

Table 3.2 shows that the Nearest Consumers Technique uses the same fault likelihood ranges as the Blocking Technique, albeit with different numbers mapping to different ranges. (This is due to the fact that Nearest Consumers uses averages to make fault likelihood estimations, rather than multiplications and subtractions). Moreover, Nearest Consumers approximates the five properties of the Blocking Technique through the procedures described earlier. This behavior is done by averaging the fault likelihood of a cell's direct consumers, and then potentially adjusting that calculation based on trends the technique observes in the current testing decisions on cells (i.e., applying one or more of the five rules described in Section 3.2.2.2).

3.2.2.5 An Example of the Nearest Consumers Technique

The Nearest Consumers Technique was used to create Figure 3.1 on the same spreadsheet program as in previous examples. In approximating the blocking behavior of the Blocking Technique, the fault likelihood of the Final_Percentage and Curved_Midterm3 cells has been estimated as "Very Low". However, the Exam_Avg and Midterm_Avg cells contribute to a single X-mark in Exam_Avg and have an estimated fault likelihood of "Medium". In addition, Midterm1_Perc contributes to the Exam_Avg X-mark as well as the X-mark in its own cell, and has an estimated fault likelihood of "High". Both the bottom of Figure 3.1 and Figure 3.6 show that the Final_Percentage and Curved_Midterm3 cells have been estimated to have the lowest estimated fault likelihood of all cells, while the other three cells have higher estimated fault likelihood. However, notice that these differences be-

tween the former two cells and the latter three cells have been exaggerated by the Nearest Consumers Technique at the bottom of Figure 3.1. Clearly, the Nearest Consumers Technique does not always compute the same results as the Blocking Technique.

3.2.3 The Test Count Technique

The technique we term "Test Count" maintains, for each cell c, an information base of the number of *successful tests* (indicated by the user via a checkmark) and *failed tests* in which c has participated. This information base can be considered a subset of that of the Blocking Technique, as Test Count maintains a record of previous tests, but not the information that would be required to calculate complex, intertwined dataflow relationships.

This technique came about by leveraging algorithms and data structures that were written for another purpose—to support semi-automated test re-use [28] (regression testing) in the spreadsheet paradigm. The detailed algorithms and data structures used for test re-use purposes are detailed in [28]. Here, we present the algorithms relevant to fault localization support for the three interactions that could trigger action from the Test Count Technique.

3.2.3.1 Changing Test Cases

Changing the input values of the spreadsheet program causes the Test Count Technique to retrieve the prior testing decision for each cell affected by the new inputs, if such testing decisions have been previously made. In the worst case, all n cells in the spreadsheet program will have contributed to all m testing decisions, and the

technique would have to search through the entire history of each cell to retrieve the testing decision. The runtime complexity in this worst case scenario is therefore $O(n * m)$.

3.2.3.2 Making a Testing Decision

The placement or removal of a testing decision triggers the algorithm outlined in Figure 3.9. The algorithm adds or removes the appropriate testing decision from the cell under consideration, and then propagates that decision to the test histories of the cells in the dynamic backward slice of the marked cell. Finally, as the user interface is updated for these cells, a fault likelihood estimation is made for each cell c. The scheme used to make this estimation is described in Section 3.2.3.4.

Using the test re-use algorithms presented in [28], after a testing decision is placed on or removed from a cell c, the information base of the Test Count Technique—the test history of each cell—is updated in $O(u * p)$ time, where u is the maximum number of uses of (references to) any cell in the program, and p is the number of c's producers.

```
1: procedure MarkPlaced (markedCell, aTest)
2:     Let markedCell's current testing decision be aTest
3:     Add or remove aTest to/from markedCell's test history as appropriate
4:     Propagate aTest to the test history of all dynamic producers of markedCell
5:     Estimate fault likelihood and update the user interface for markedCell and all of
       markedCell's dynamic producers
6: end procedure
```

FIGURE 3.9: Algorithm #7 — The MarkPlaced subroutine for the Test Count Technique.

3.2.3.3 Changing the Spreadsheet Logic

Changing cell c's non-constant formula to a different formula requires all saved information about c's tests and those of its consumers to be considered to be obsolete. For the same reason, the fault likelihood of c and its consumers must all be reinitialized to zero.

Using the test re-use methodology [28], all of the related testing information can be updated in $O(t * m * max(u, \textit{cost of set operations}))$, where t is the number of tests that reach the modified cell, m is the maximum number of consumers affected by c's tests, and u (as it was earlier) is the maximum number of uses for any cell in the program.

3.2.3.4 Mapping Information to Estimated Fault Likelihood

The Test Count Technique maintains the first three properties of the Blocking Technique. Because it does not track the dataflow relationships between cells and testing decisions, it cannot maintain the "blocking" characteristics of the Blocking Technique, and therefore does not maintain the fourth and fifth properties. Instead, fault

```
1: procedure NewFormula (editedCell)
2:     Remove all decisions in editedCell's test history from editedCell's consumers
3:     Clear the test history of editedCell
4:     Set fault likelihood of editedCell and all consumers to "None"
5:     Update the user interface for editedCell and all of editedCell's consumers
6: end procedure
```

FIGURE 3.10: Algorithm #8 — The NewFormula subroutine for the Test Count Technique.

Numerical Values	Fault Likelihood Range	Fault Likelihood Description
0	0	"None"
1 – 3	1	"Low"
3 – 4	2	"Medium"
5 – 6	3	"High"
7 – ∞	4	"Very High"

TABLE 3.3: The mapping used by the Test Count Technique to transform numerical values into discrete fault likelihood ranges.

likelihood estimations are made purely by observing the number of X-marks and checkmarks in each cell's test history.

Let NumFailingTests (*NFT*) be the number of X-marks placed on c, and let NumSuccessfulTests (*NST*) be the number of checkmarks placed on c. If a cell c has no failed tests, the fault likelihood of c is "None". Otherwise, the fault likelihood of a cell is estimated as follows:

$$fault\ likelihood(c) = max(1, 2 * NFT - NST)$$

This calculation is mapped to one of four possible fault likelihood ranges using the scheme outlined in Table 3.3.

3.2.3.5 An Example of the Test Count Technique

An example of the previously introduced Gradebook spreadsheet program with the Test Count Technique is provided in Figure 3.11. The Midterm1_Perc cell contributes to two X-marks, and has an estimated fault likelihood of "Medium". The other four highlighted cells contribute to only a single X-mark, and therefore have a fault likelihood of "Low".

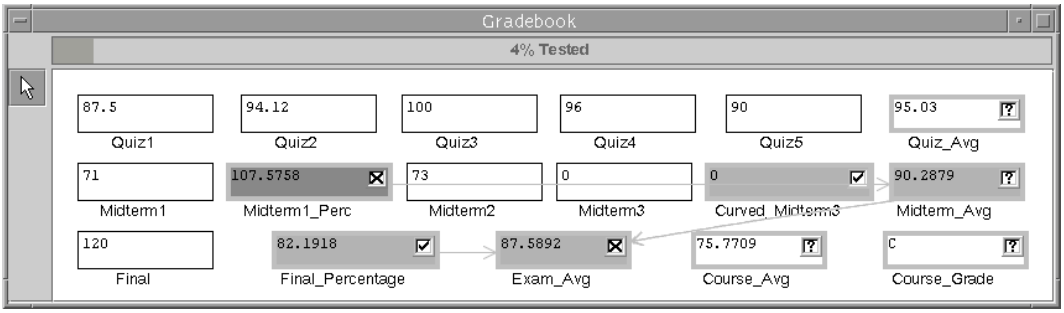

FIGURE 3.11: The Gradebook program with the Test Count Technique.

Note that the Final_Percentage and Midterm_Avg cells have the same fault likelihood as Exam_Avg and Course_Avg despite their checkmarks. This is because the Test Count Technique does not maintain or approximate the blocking properties of the Blocking Technique (i.e., Properties 4 and 5). Although the technique does maintain the third property of making fault likelihood inversely proportional to the number of checkmarks in which the cell participates, the "Low" fault likelihood range is the lowest property maintained by this technique. Consequently, the technique cannot reduce the estimated fault likelihood of Final_Percentage and Midterm_Avg, despite the checkmarks on these cells.

3.3 Related Work

Chapter 1 mentioned some previous research into fault localization techniques for professional programmers. We first expand on that discussion, and then outline some of the emerging work into fault localization for end-user programmers.

3.3.1 Fault Localization for Professional Programmers

Most fault localization research has been based on program slicing [64] and dicing [39] techniques; see [60] for a survey of this work. In general, a program slice relative to a variable v at a program point p is the set of all statements in the program that affect the value of v at p. Our fault localization techniques draw from information gleaned via dynamic program slicing [2, 36], and make use of that information using heuristics inspired by dicing.

There has been a great deal of work on fault localization strategies for professional programmers (e.g., [3, 14, 24, 29, 32, 39, 46, 51, 61]). For example, Agrawal et al. [3] present a technique, implemented as a tool called χslice, for locating faults in traditional programming languages using execution traces from tests. This technique is based on displaying dices of the program relative to one failing test and a set of passing tests. Jones et al. [32] describe a similar approach implemented as a tool called TARANTULA. Unlike χslice, TARANTULA utilizes information from *all* passing and failing tests, coloring statements based on the likelihood that each statement is faulty according to its ratio of failing tests to passing tests. Francel and Rugaber [29] use execution traces to build a directed graph that models the propagation of values, and then use output values to narrow the region that should be examined. Using a faulty "run" and a larger number of correct runs, Renieris and Reiss [51] propose a fault localization technique that compares a faulty run with the correct run that most resembles that faulty run, and reports "suspicious" areas of the program based on this comparison. Two ways that our methods differ from all of these approaches are that our methods (1) are targeted at end users, and (2) are

interactive and incremental at the granularity of revising fault likelihood estimations in real time after each single program edit.

Pan and Spafford [42] developed a family of twenty heuristics appropriate for automated fault localization. These heuristics are based on the program statements exercised by passing and failing tests. Our strategy directly relates to three of these heuristics: the set of all program points exercised by failed tests, program points that are exercised by a large number of failed tests, and cells that are exercised by failing tests and that are not exercised by passing tests.

3.3.2 Fault Localization for End-User Programmers

Although work aimed specifically at aiding end users with debugging is beginning to emerge, fault localization support for end users remains scarce. Focusing specifically on fault localization, Ayalew and Mittermeir [9] present a method of "fault tracing" for spreadsheet programs based on "interval testing" and slicing. This strategy reduces the search domain after it detects a failure, and selects a single cell as the "most influential faulty". Woodstein [62, 63] is a web interface agent that assists e-commerce debugging by allowing users to directly interact with web pages. Users can invoke an inspector that converts web page data into buttons, which the user can manipulate to traverse transactions. Ko and Myers [35] present a type of fault localization via the Whyline, an "interrogative debugging" technique. Users pose questions in the form of "Why did…" or "Why didn't…" that the Whyline answers by displaying visualizations of the program. This work builds on their model of programming errors [34], which classifies errors and their causes. Our approach differs from the first strategy by allowing users to interactively improve

feedback by providing the system with additional information, and from all these strategies through the incorporation of the robustness feature built into Property 1.

There is other work that can help end users find faults. S2 [59] 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. Igarashi et al. [31] present comprehension devices that can aid spreadsheet users in dataflow visualization and editing tasks, and finding faults. There is also recent work to automatically detect certain kinds of errors, such as errors in spreadsheet units [1, 7, 26] and types [4]. Our approach differs from these approaches by harnessing the relationship between testing and debugging to provide explicit fault localization feedback.

There has also been work to help end users detect failures. Statistical outlier finding [40] and anomaly detection [48] use statistical analysis and interactive techniques to direct end-user programmers' attention to potentially problematic areas during automation tasks. Also, the assertions approach in Forms/3 automatically detects failures in spreadsheet cells, and has been shown empirically to help end-user programmers correct errors [16, 65].

CHAPTER 4

AN EXPERIMENT TO INVESTIGATE TECHNIQUE EFFECTIVENESS

Which of the three fault localization techniques described in Chapter 3 provide effective fault localization feedback? In what ways should we consider adjusting our techniques so that they might provide more effective feedback? Prior to this thesis, no empirical work had been conducted that could answer these questions. Consequently, we conducted a formative experiment—described in this chapter—to gain insights into the effectiveness of our three techniques' feedback at two points in debugging: first, very early in debugging, when the fault localization techniques provide initial feedback to the user; and second, at the end of debugging, when the techniques have accumulated a quantitatively greater amount of testing information with which to provide feedback. Answering questions such as those that were previously mentioned is the purpose of a formative experiment, so named because it helps to inform the design of a system or process.[1]

4.1 Design

One possible experiment aimed at these questions would have been to create hypothetical test suites to simulate a user's actions in some collection of spreadsheet

---

[1] The results of this experiment were first reported in [56].

programs. We could have then run each hypothetical test suite under each technique to compare the techniques' effectiveness. However, to tie our study more closely to its ultimate users, we instead elected to use as test suites the testing actions that end users actually performed in an earlier experiment. These test suites were the subjects of our experiment.

In this previous study, no fault localization technique was present in the programming environment. Also, the participants were not allowed to change any formulas—they could only change the values of the input cells and communicate testing decisions using X-marks and checkmarks. These restrictions were useful for control and measurement purposes of the current experiment because they ensured that all spreadsheet programs contained the same faults throughout the duration of each session. Also, allowing program changes would carry the possibility of invalidating previous testing decisions, which would inhibit one of our experiment's goals: to measure the effectiveness of our fault localization techniques as they accumulate greater amounts of testing information.

4.1.1 Materials

In the previous study that generated our current study's test suite subjects, end-user participants recruited from a computer literacy course were instructed to test and identify errors in two Forms/3 spreadsheet programs: Change and Grade, which are shown in Figures 4.1 and 4.2, respectively. Change calculates the optimized number of dollars, quarters, dimes, nickels, and pennies that are returned when a jar of pennies is cashed in at a bank. Grade computes a letter grade (A, B, C, D,

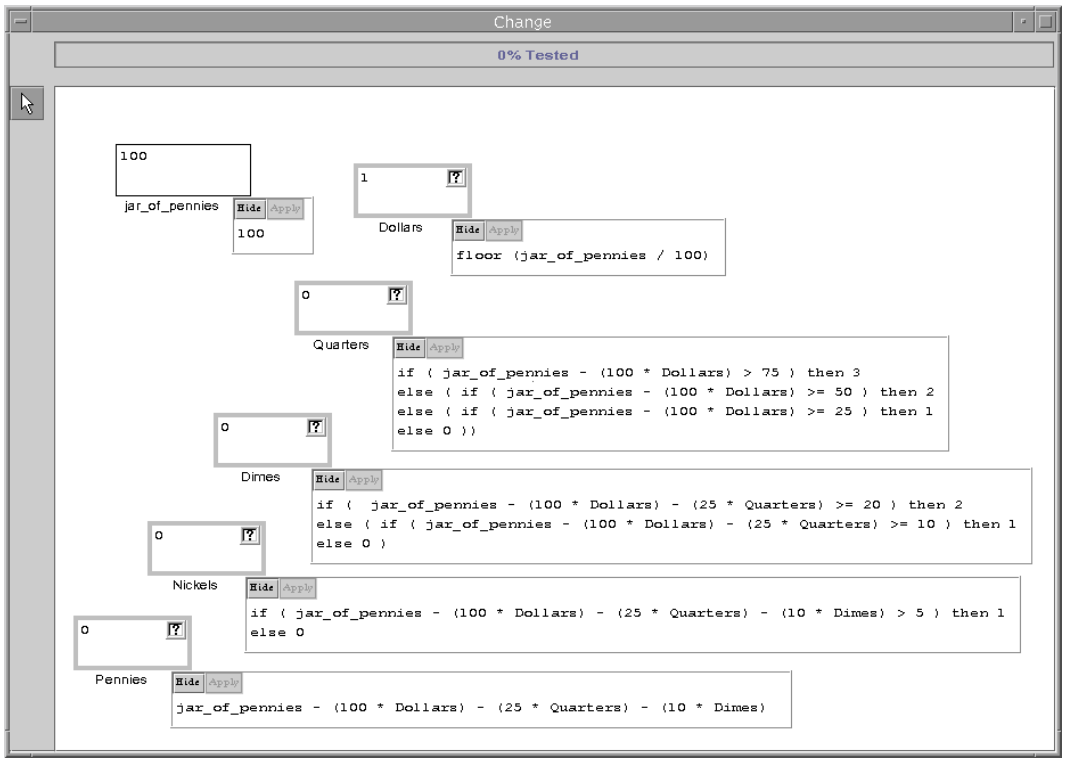

FIGURE 4.1: The Change spreadsheet program used in our first experiment. The Quarters, Nickels, and Pennies cells contain faults.

F) based on four quiz scores and an extra credit score. Each of the two programs contained three seeded faults, which are outlined in Tables 4.1 and 4.2.

A difference between the two spreadsheet programs with implications for fault localization is that Change involves a narrow dataflow chain leading to a single output, whereas Grade has several relatively independent dataflow paths.

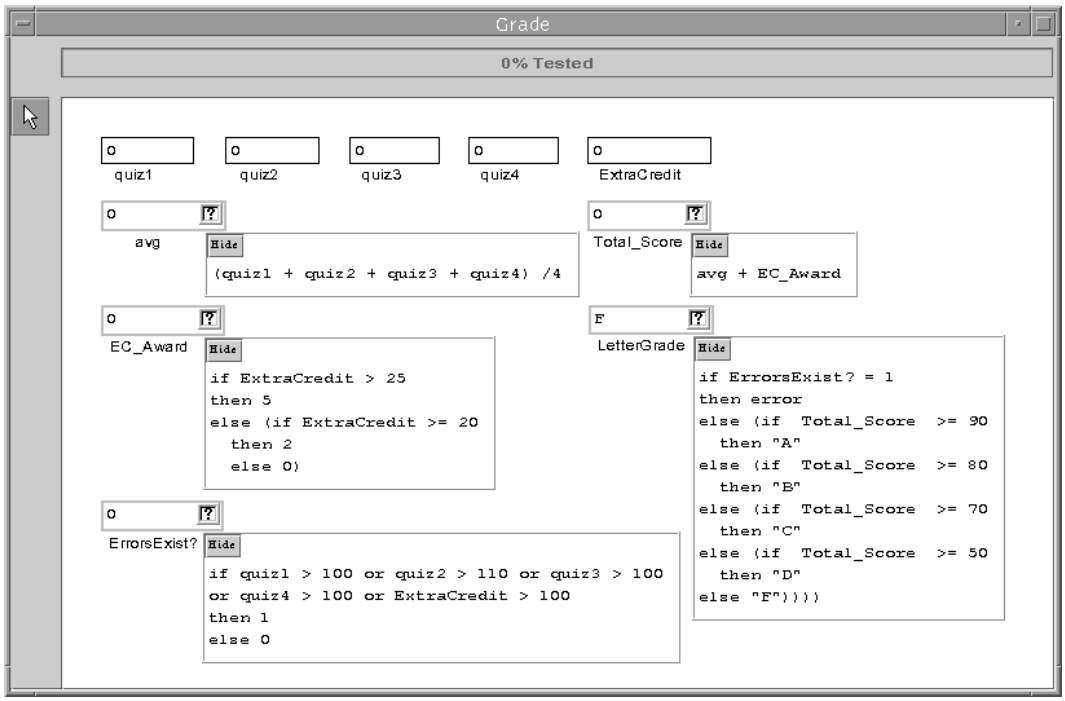

FIGURE 4.2: The Grade spreadsheet program used in our first experiment. The EC_Award, LetterGrade, and ErrorsExist? cells contain faults.

4.1.2 Procedures

The previous study began with the end-user participants being given a brief tutorial to familiarize themselves with the Forms/3 environment and WYSIWYT testing methodology, which was described in Chapter 2. The tutorial taught the use of placing checkmarks and X-marks, but did not include any testing or debugging strategy content. We led the participants through tasks in the Forms/3 environment, with guidance at each step. The participants completed these guided tasks on their own machines. Participants were free to ask questions or seek clarifications during the tutorial.

Cell Name	Faulty Formula	Correct Formula
Quarters	if (jar_of_pennies - (100 * Dollars) $>$ 75) then 3 else (if (jar_of_pennies - (100 * Dollars) $\geq$ 50) then 2 else (if (jar_of_pennies - (100 * Dollars) $\geq$ 25) then 1 else 0))	if (jar_of_pennies - (100 * Dollars) $\geq$ 75) then 3 else (if (jar_of_pennies - (100 * Dollars) $\geq$ 50) then 2 else (if (jar_of_pennies - (100 * Dollars) $\geq$ 25) then 1 else 0))
Nickels	if (jar_of_pennies - (100 * Dollars) - (25 * Quarters) - (10 * Dimes) $>$ 5) then 1 else 0	if (jar_of_pennies - (100 * Dollars) - (25 * Quarters) - (10 * Dimes) $\geq$ 5) then 1 else 0
Pennies	jar_of_pennies - (100 * Dollars) - (25 * Quarters) - (10 * Dimes)	jar_of_pennies - (100 * Dollars) - (25 * Quarters) - (10 * Dimes) - (5 * Nickels)

TABLE 4.1: The faults seeded in the Change task.

After the tutorial, participants were given the Change and Grade spreadsheet programs in varying order, with instructions to try input values and mark cells containing correct values with checkmarks, and cells containing incorrect values with X-marks. During the course of this study, we recorded the testing actions of each participant into an electronic transcript.

For our current study, these recorded transcripts were our subjects. We ran these recorded transcripts through a tool that replays the testing actions contained therein to extract the test values the users entered into input cells, as well as the cells in which they placed X-marks or checkmarks.

Cell Name	Faulty Formula	Correct Formula
LetterGrade	if ErrorsExist? = 1 then error else (if Total_Score $\geq$ 90 then "A" else (if Total_Score $\geq$ 80 then "B" else (if Total_Score $\geq$ 70 then "C" else (if Total_Score $\geq$ 50 then "D" else "F"))))	if ErrorsExist? = 1 then error else (if Total_Score $\geq$ 90 then "A" else (if Total_Score $\geq$ 80 then "B" else (if Total_Score $\geq$ 70 then "C" else (if Total_Score $\geq$ 60 then "D" else "F"))))
EC_Award	if ExtraCredit $>$ 25 then 5 else (if ExtraCredit $\geq$ 20 then 2 else 0)	if ExtraCredit $>$ 25 then 5 else (if ExtraCredit $\geq$ 20 then 3 else 0)
ErrorsExist?	if quiz1 $>$ 100 or quiz2 $>$ 110 or quiz3 $>$ 100 or quiz4 $>$ 100 or ExtraCredit $>$ 100 then 1 else 0	if quiz1 $>$ 100 or quiz2 $>$ 100 or quiz3 $>$ 100 or quiz4 $>$ 100 or ExtraCredit $>$ 100 then 1 else 0

TABLE 4.2: The faults seeded in the Grade task.

4.1.3 Measures for Evaluation

The experiment reported in this chapter seeks to measure the *effectiveness* of a fault localization technique. For this experiment, we define the effectiveness of a fault localization technique as its ability to correctly and visually differentiate the correct cells in a program from those cells that contain faults. The better job a technique does in visually distinguishing a program's faulty cells from its correct cells, the more effective the technique. Thus, we measure effectiveness by measuring the *visual separation* between the faulty cells and the correct cells of each program. For

this experiment, our effectiveness metric subtracts the average fault likelihood estimated for the correct cells from the average fault likelihood estimated for the faulty cells. (Subtraction is used instead of calculating a ratio because the fault likelihood ranges form an ordinal, not a ratio, scale.) Positive effectiveness is preferable to negative effectiveness, and a greater effectiveness indicates a better distinction between faulty and non-faulty cells.

We measure effectiveness at two points in time: (1) at the end of testing after all information has been gathered; and (2) very early, just after the first failure is observed (regardless of whether successful tests have occurred). The reason for measuring visual separation at the end of testing is because this is the point at which the greatest amount of testing information is available for providing fault localization feedback. The reason for measuring after the first failure (X-mark) is because it is at this point that a technique would first give a user visual feedback. This initial visual feedback may be flawed because the technique has very little information upon which to base its estimations. However, initial feedback is important because it may influence the subsequent actions of the user.

4.1.4 Threats to Validity

Every experiment has threats to the validity of its results[2], and these threats must be considered in order to assess the meaning and impact of results. This section discusses potential threats to the validity of our experiment and, where possible,

---

[2] Readers are referred to [66] for a general discussion of validity evaluation and a threats classification.

how we attempted to mitigate the impact of these threats on our results. The threats we discuss include: (1) threats to internal validity (could other factors be responsible for our results), (2) threats to construct validity (are the results yielded by this experiment based on appropriate information), (3) threats to external validity (to what extent could our results be generalized), and (4) threats to conclusional validity (what are the limitations of the conclusions drawn from this experiment, and how could a stronger experiment be designed).

4.1.4.1 Threats to Internal Validity

The specific faults in the experiment's programs may have contributed to our results. Unfortunately, the specific faults could be responsible for the results of any experiment we might conduct, so our results must always be interpreted in the context of this limitation.

4.1.4.2 Threats to Construct Validity

The effectiveness metric used in this experiment may not accurately reflect the fault localization feedback provided by our techniques. While our results should be considered in light of this threat, it is also important to note that this experiment was designed to be a formative experiment to shed some light on the effectiveness of our previously unevaluated techniques. Future studies could address this concern by employing a variety of metrics to measure fault localization effectiveness.

4.1.4.3 Threats to External Validity

Program representativeness is an issue facing this experiment. The spreadsheet programs used in this experiment are relatively small, and therefore might not resemble those used in practice. Another threat is the selection of faults in the two programs. To reduce this threat, experienced Forms/3 developers seeded "real-world" faults into these programs that resemble those that the developers have seen in spreadsheet programs, and in their own use of the Forms/3 system. Finally, this experiment was conducted in the Forms/3 research language [8, 15, 18], and end users could make different testing decisions in a different language. All of these threats can be addressed only through repeated studies, using different programs, faults, and languages.

4.1.4.4 Threats to Conclusional Validity

The use of additional spreadsheet programs, with varying degrees of complexity, would enhance the power of our conclusions. We could also strengthen future studies by comparing a greater number of fault localization techniques.

4.2 Results

4.2.1 Effectiveness Comparisons

Which fault localization technique would fare the best at the point of the "First X-mark", when very little information is available with which to provide fault localization feedback? Would the same techniques with a favorable effectiveness at the test suites' first X-mark do well by the "Last Test", at the end of the subject test

suites? As a statistical vehicle to shed light on these effectiveness questions, we state the following (null) hypotheses:

H1: *At the point the first failure is recorded, there is no difference in effectiveness among the three fault localization techniques.*

H2: *By the time all tests have been completed, there is no difference in effectiveness among the three fault localization techniques.*

We used the Friedman test to statistically analyze the results pertaining to each research question. This test is an alternative to the repeated measures ANOVA, when the assumption of normality or equality is not met. Because Friedman tests only indicate whether there is a significant difference in the effectiveness of the three techniques, we also used the Sum of Ranks test to determine which technique(s) were significantly more effective than other technique(s).

Table 4.3 presents the mean effectiveness calculations of the three fault localization techniques. In the table, the mean effectiveness calculations are shown, with standard deviations parenthesized. An effectiveness of 1.000 would signify that one discrete fault likelihood range separates the faulty cells from the correct cells. The maximum possible effectiveness is 5.000, the minimum possible effectiveness is -5.000, and 0.000 signifies that the average fault likelihood of faulty and correct cells is the same. The technique with the greatest average effectiveness is shown in bold. The "n" denotes the number of subjects measured at each point. Also, $\sim$ denotes marginal statistical significance in a technique's difference from the other techniques (0.10 level of significance), * denotes a difference significant at the 0.05 level, and ** at the 0.01 level.

	First X-mark			Last Test		
	TC	BL	NC	TC	BL	NC
Change	-0.212	**0.008**$\sim$	-0.201	**0.466***	0.227	0.152
$(n = 44)$	(0.798)	**(0.612)**	(1.213)	**(0.812)**	(0.593)	(0.868)
Grade	0.016	0.093	**0.190****	**0.151***	0.031	-0.081
$(n = 43)$	(0.879)	(0.738)	**(1.213)**	**(0.756)**	(0.597)	(0.874)

TABLE 4.3: The effectiveness of the three fault localization techniques.

For the first X-mark placed in the Change spreadsheet program, statistical analysis using the Friedman test showed a marginally significant difference (at the 0.10 level) among the three techniques ($df = 2, p = 0.0561$). At this 0.10 level, the Sum of Ranks test indicated that the Blocking Technique was more effective (with marginal significance) than the Test Count and Nearest Consumers Techniques. In the Grade program, there was a significant difference (at the 0.01 level) among the three techniques ($df = 2, p = 0.0028$), and the Sum of Ranks test indicated that the Nearest Consumers Technique was significantly more effective than Blocking or Test Count. We therefore reject H1.

By the last test of the subject test suites, the Test Count Technique showed the benefits of the information that it builds up over time. More tests did not help the Blocking and Nearest Consumers Techniques as much. Friedman tests showed a significant difference among the three techniques (Change: $df = 2, p = 0.0268$; Grade: $df = 2, p = 0.0109$). Perhaps more revealing, by the last test, the Sum of Ranks test revealed that the Nearest Consumers Technique had significantly worse separations than the Test Count Technique in both programs. (The Test Count and

Nearest Consumers Techniques were not significantly different from the Blocking Technique.) Given these differences, H2 must also be rejected.

We caution that the standard deviations in the data are large. These large standard deviations may not be surprising given the variations in human behavior during interactive testing; still, we must be cautious about generalizing from these data.

The large standard deviations suggest another view of the effectiveness of our visualizations: for how many subjects did the visualization succeed and fail? That is, how often was the darkest cell—colored dark to attract the attention of the user—truly faulty? At the left of Table 4.4 is the percent of test suites (subjects) in which the darkest cell at the end of testing was indeed faulty. The right side shows the percent of subjects in which a correct cell was erroneously colored darker than the faulty ones. (Ties, in which the darkest correct cell and the faulty cell are the same color, are omitted from this table.) Low false identification rates may be particularly important to end users, who may be relying on the visualizations for guidance in finding the faults.

	True Identifications			False Identifications		
	TC	BL	NC	TC	BL	NC
Change	18.18%	13.64%	31.82%	11.36%	13.64%	9.09%
Grade	44.19%	51.16%	39.53%	6.98%	11.63%	16.28%

TABLE 4.4: Is the darkest cell faulty at the end of testing?

4.2.2 Robustness

As explained in Chapter 1, empirical evidence suggests that end users engaging in interactive testing are likely to make some mistakes [45, 58]. For example, a user might pronounce a test successful when in fact it reveals a failure. A fault localization technique may not be useful, regardless of its effectiveness, if it cannot withstand such mistakes.

To consider this issue, of the 44 subjects for the Change spreadsheet program, we isolated 29 that contained either a wrong X-mark or a wrong checkmark. Similarly, 21 of the 43 subjects for the Grade program contained an erroneous mark. We refer to subjects with no erroneous marks as "perfect", and refer to the others as "imperfect". Given this information, we form the following (null) hypotheses to investigate robustness.

H3: *At the point the first failure is recorded, there is no significant difference in effectiveness among the three fault localization techniques for "perfect" subjects.*

H4: *By the time all tests have been completed, there is no significant difference in effectiveness among the three fault localization techniques for "perfect" subjects.*

H5: *By the time all tests have been completed, there is no significant difference in effectiveness among the three fault localization techniques for "imperfect" subjects.*

Table 4.5 contains the data for the perfect subjects. When restricted to these subjects, the differences indicated by Friedman tests were significant only for the first X-mark of Grade (First X-mark Change: $df = 2, p = 0.6592$; Last Test Change:

	First X-mark			Last Test		
	TC	BL	NC	TC	BL	NC
Change	-0.089	**0.078**	-0.078	**0.667**	0.389	0.456
$(n = 15)$	(0.877)	**(0.678)**	(1.367)	**(0.893)**	(0.760)	(0.856)
Grade	0.296	0.333	**0.722***	**0.037**	0.000	-0.148
$(n = 9)$	(0.735)	(0.493)	**(0.833)**	**(0.740)**	(0.577)	(0.626)

TABLE 4.5: The average effectiveness, with standard deviations, for the "perfect" subjects.

$df = 2, p = 0.3618$; First X-mark Grade: $df = 2, p = 0.0169$; Last Test Grade: $df = 2, p = 0.2359$). Although this may be due simply to the smaller sample sizes involved, another possibility is that the errors included in the full sample were part of what distinguished the techniques from one another. We reject H3, but not H4.

Comparing data on the imperfect subjects in isolation is a way to investigate this possibility. We cannot do a "First X-mark" comparison for these subjects, since the sample necessarily includes both subjects with a correct first X-mark followed by other errors, and subjects with an incorrect first X-mark, followed possibly by other errors.

However, the data for the last test are shown in Table 4.6. In the Change spreadsheet program, the Friedman test indicates marginally significant differences ($df = 2, p = 0.0697$). The Sum of Ranks test adds to this information by indicating that the Test Count Technique is more effective than the other two techniques at this significance level. For the Grade program, the Friedman test indicates significant differences ($df = 2, p = 0.0433$), and the Sum of Ranks test indicates that the Test

	First X-mark			Last Test		
	TC	BL	NC	TC	BL	NC
Change ($n = 29$)	Not applicable			**0.362**$\sim$ **(0.763)**	0.144 (0.479)	-0.006 (0.847)
Grade ($n = 34$)	Not applicable			**0.181*** **(0.768)**	0.039 (0.610)	-0.064 (0.936)

TABLE 4.6: The average effectiveness, with standard deviations, for the "imperfect" subjects.

Count Technique was significantly better than the Nearest Consumer Technique (but not the Blocking Technique). Thus, H5 is rejected.

4.3 Discussion

4.3.1 Dataflow and Initial Fault Localization Feedback

In Section 4.2.1, note that the "First X-mark" results for the Change spreadsheet program had most of its effectiveness calculations in the wrong direction—the correct cells tended to be colored darker than the faulty cells—but this did not happen in the Grade program. Recall that the Change program's cells lined up in a single dataflow chain feeding into the final answer, whereas the Grades program had multiple dataflow chains. This may suggest that the techniques are all at a disadvantage in providing reasonable early feedback, given a single dataflow chain. The question of whether certain dataflow patterns are especially resistant to early feedback about faulty cells is an interesting one, but it requires further experimentation with additional spreadsheet programs.

4.3.2 Cost-Effectiveness Implications

Whenever the Friedman tests revealed a significant difference by the last test of a test suite, Test Count was the most effective. Test Count was also fairly reliable, reporting a reasonably low number of false identifications, although it did not distinguish itself in true identifications. Further, it seemed to be the most robust, maintaining the best effectiveness even in the presence of incorrect testing decisions. These positive attributes are despite the fact that it is not as expensive as the Blocking Technique. However, it was the least effective of the three techniques at providing early feedback.

The Blocking Technique is the most expensive, and we had expected its effectiveness to be commensurately high, producing a good cost-effectiveness ratio. The data from Section 4.2, however, do not lead to this conclusion by the end of a test suite.

The Blocking Technique did show two important advantages, however. First, it was always better than Test Count for the early feedback. Second, it was much more consistent than the other two techniques, showing smaller spans and smaller variances than the other two in most cases. These facts suggest that if effectiveness were the only goal, a possibility might be to start with the Blocking Technique for early feedback, and switch to the Test Count Technique in the long run.

We devised the Nearest Consumers Technique with the hope that it would approach the performance of the other two with less cost. Instead, we learned that the Nearest Consumers Technique was less consistent that the other two techniques, both in the comparisons with the others' data, and in the large variance within its results. Further, in some cases, it performed quite badly by the last test. However,

it sometimes performed quite well, especially in the “First X-mark” comparisons. This may say that, despite the fact that it is less consistent than the Blocking Technique, it may still be viable to use during the early feedback period, and its lower expense may make it the most cost-effective choice for this purpose.

4.3.3 Improving the Blocking Technique

Recall that one of the questions that we hoped to answer through this formative experiment was: “In what ways should we consider adjusting our techniques so that they might provide more effective feedback?” Coming into this experiment, we had expected the Blocking Technique to outperform all other techniques due to its consideration of dataflow relationships in its fault likelihood estimations *in addition to* its maintenance of passed and failed tests. But this was not the case: by the last test of our subject test suites, the Test Count Technique proved to be superior to the Blocking Technique. What characteristic does the Test Count Technique have that the Blocking Technique does not that could account for the differences seen in Section 4.2?

We noticed that the mathematical mapping of testing information employed by the two techniques has a stark difference. Both techniques give failed tests twice the weighting as passed tests. However, the ratio between the weighting of failed and passed tests in Test Count is double that in Blocking. We hypothesized that

this difference may have accounted for the disparity in the effectiveness of the two techniques.[3]

To investigate this, we made the mathematical mapping of the Blocking Technique more similar to that of the Test Count Technique. This was done by changing Blocking's ratio between the weighting of failed and passed tests so that it was identical to that of Test Count. We then duplicated the procedures of the experiment for this "modified" Blocking Technique.

A comparison among the "original" Blocking Technique with the modified technique and Test Count is provided in Table 4.7. The modification to the mathematical mapping of the Blocking Technique did not improve effectiveness at the first X-mark of the subject test suites. However, by the last test, this modification dramatically improved technique effectiveness. In fact, the modified Blocking Technique had nearly the same average effectiveness as the Test Count Technique, which in Section 4.2 proved to be the superior technique at this point of measurement.

This result was quite surprising, indicating that adjusting a factor such as mapping may dramatically improve a technique's effectiveness. We caution that our ability to draw such conclusions is limited without a measurement of statistical confidence in this result. However, because we simply changed one aspect of the Blocking Technique—the mathematical mapping—and achieved a quantitatively greater effectiveness, the result caused us to ponder whether there is more than one inde-

---

[3] A second difference between the techniques was that the Blocking Technique had five fault likelihood ranges ("Very Low" to "Very High"), while Test Count had only four fault likelihood ranges ("Low" to "Very High"). However, a greater amount of fault likelihood ranges was seen as more of an asset to the Blocking Technique, so we decided not to isolate this particular factor.

First X-mark			
	Orig-BL	Mod-BL	TC
Change	**0.008**	-0.201	-0.212
$(n = 44)$	**(0.612)**	(0.775)	(0.798)
Grade	**0.093**	0.078	0.016
$(n = 43)$	**(0.738)**	(0.849)	(0.879)
Last Test			
	Orig-BL	Mod-BL	TC
Change	0.227	0.462	**0.466**
$(n = 44)$	(0.593)	(0.800)	**(0.812)**
Grade	0.031	**0.159**	0.151
$(n = 43)$	(0.597)	**(0.999)**	(0.756)

TABLE 4.7: A comparison of the "original" Blocking Technique (Orig-BL) described in Section 3.2.1, the "modified" Blocking Technique (Mod-BL), and the Test Count Technique (TC).

pendent factor involved a fault localization technique's effectiveness. In the next two chapters, we describe the steps we took to investigate this possibility.

CHAPTER 5

TWO FACTORS INFLUENCING FAULT LOCALIZATION TECHNIQUES

We observe that any fault localization approach that includes some form of reporting or feedback to a human is composed of two orthogonal factors[1]:

- *Information Base:* This factor is the *information* used by a technique to estimate fault likelihood. To abstract away implementation details such as data structures or algorithmic details, we use this term to refer to only the type of information used and the circumstances under which it is maintained.
- *Mapping:* This factor is how a technique maps the information base into fault localization feedback. Once again, we do not consider implementation details regarding specifically how such a mapping occurs.

For example, TARANTULA [32], a fault localization technique for traditional programming languages, uses a set of passed and failed tests as its information base. As its mapping, two mathematical formulas calculate (1) a color representing each statement's participation in testing, and (2) the technique's confidence in the correctness of each color.

---

[1] One could consider the specific mechanisms used to communicate fault likelihood to a user as a third factor. We do not discuss this factor in this chapter because the visualization mechanisms of communicating fault likelihood estimations are highly dependent on the targeted environment. Also, these mechanisms are unlikely to impact the *actual estimations* made by the technique, which are usually based on the information maintained by the technique and how that information is mapped into fault localization feedback.

5.1 Information Bases

To support the behavior of a fault localization technique, information must be stored and maintained by the technique (or the surrounding environment). Each of our three techniques maintains a unique base of information that is used to achieve the behavior described in Chapter 3. We draw from that chapter to describe each technique's information base, which we refer to as I-TC, I-BL, and I-NC for the remainder of this thesis.

- *Test Count* (I-TC). The information base of the Test Count Technique tracks, for each cell c, the set of passed and failed tests that dynamically execute c. This information base is similar to that used in TARANTULA [32].
- *Blocking* (I-BL). There are two aspects to this information base. Like I-TC, I-BL maintains a list of all passed and failed tests for each cell. However, to achieve its "blocking" behavior, I-BL also uses the dataflow relationships between each cell to allow tests to "block" other tests from reaching certain cells under the circumstances described in Chapter 3. This information base is similar to that used in dicing [21, 39].
- *Nearest Consumers* (I-NC). This base tracks only (1) the current fault likelihood of every cell in the program, and (2) the current testing decision for each cell *affected by the current test case* so as to adjust for trends in testing decisions. This "discount" information base is more modest than that used by most other fault localization techniques in the literature.

Note that, because the context for this discussion is *interactive* fault localization, each of these information bases is immediately updated whenever any action is taken by a user that affects the contents of the base, potentially interfering with the

environment's interactivity. One reason to compare these information bases, then, is to learn whether it is possible for a modest version such as I-NC to compete in effectiveness with the other two more extensive information bases.

5.2 Mappings

The manner in which our techniques draw from the information bases to estimate fault likelihood is through the mapping factor. Mappings transform information bases into fault localization feedback that fulfills the goals outlined at the beginning of Chapter 3.

Many mappings are possible, but it would not be feasible to compare them all. Further, doing so is not warranted until we learn whether mapping is important to a technique's effectiveness. Thus, we use the three particular mappings from our three fault localization techniques as a vehicle for investigating the importance of mapping as an independent factor. We refer to our three mappings techniques as M-TC, M-BL, and M-NC for the remainder of this thesis.

- *Test Count* (M-TC). The Test Count technique's mapping ensures that the fault likelihood of a cell c is directly proportional to the number of c's failed tests, and inversely proportional to the number of c's passed tests. It has the characteristic of mapping information bases to four fault likelihood values, and begins by assigning c the lowest fault likelihood if it contributes to a single failure (X-mark), thereby allowing fault likelihood to build with further failures.
- *Blocking* (M-BL). This mapping is similar to M-TC, except that it supports five, rather than four, fault likelihood values, and begins by assigning c the

second lowest fault likelihood value so as to be able to build but also to reduce a cell's fault likelihood value when a test blocks fault likelihood propagation to it. Also, as presented in Chapter 3, the weighting assigned to passed and failed tests in M-BL is different than that used in M-TC.

- *Nearest Consumers* (M-NC). This mapping computes an adjusted average of the fault likelihood of the cells to which c directly contributes. This calculated mean is adjusted as described in Chapter 3 based on trends in current testing decisions. It also supports five fault likelihood values, and begins by assigning c the third value so as to allow viability of increasing or decreasing fault likelihood values as the cell's neighbors' fault likelihoods increase and decrease.

These mappings have another important characteristic. A previous study investigating the strategies and behaviors of end-user programmers using fault localization techniques [45, 58] found that end-users often make mistakes when interactively testing or debugging their programs. In consideration of this, each of our mappings incorporates a "robustness" feature (labeled "Property 1" in Chapter 3) that ensures that any cell that might have contributed to the computation of an incorrect value (failure) will be assigned some positive fault likelihood. This property ensures that if there had been even one correctly placed X-mark involving a cell, ensuing incorrect checkmarks could not completely remove the cell from a user's search space.

Although each mapping comes from a different technique, all mappings have two characteristics: (1) some number of fault likelihood values possible, and (2) an "initial" value. Later in this thesis, when we refer to applying a mapping to an information base, we refer to applying these two characteristics. Note that we

do not attempt to tease apart the influences these characteristics might have, but simply consider them an atomic unit to learn whether mapping in general can have an impact on technique effectiveness.

To our knowledge, no previous work has considered the impact that a mapping alone might have on the effectiveness of a particular fault localization technique. Instead, any results pertaining to effectiveness are attributed to the technique as a whole (primarily to its reasoning mechanisms). A contribution of this thesis is to show that mapping must be separated as a factor from the information base in understanding effectiveness results. Otherwise, the effectiveness (or ineffectiveness) of a technique may be wrongly attributed to the underlying information base, when instead it is simply an impact of the mapping.

5.3 Evaluation of Interactive Fault Localization Techniques

In considering how to evaluate interactive fault localization techniques for end users, we reflect again on an important difference between end-user and professional software development that was outlined in Chapter 1. Many traditional fault localization techniques report feedback only at the end of a batch processing of information. This point of maximal system reasoning potential—when the system has its best (and only) chance of producing correct feedback—is therefore the appropriate point to measure these techniques.

Given the interactive nature of end-user environments, however, debugging, and therefore fault localization use, occurs not just at the end of testing, but *throughout* the testing process. Measuring technique effectiveness only at the end of testing would thus ignore most of the reporting being done by the interactive technique.

Ideally, then, we should measure at every point at which a user receives feedback. However, it is not statistically viable to measure at all feedback points, because not every point will be reached by enough subjects to allow comparison. Therefore, in the upcoming experiment, we measured at the following points where fault localization feedback is reported:

- *First X-mark.* When a failure is first reported by users (in our environment, signaled by an X-mark), they *immediately* receive fault localization feedback. We term this the beginning of a *debugging session*. (X-marks initiate such sessions only when no other session is already in progress.) Because this point marks the first (and perhaps only) opportunity for techniques to provide feedback, we measure technique effectiveness here.
- *Second X-mark.* The second X-mark's computations are based on a greater quantity of information than the first X-mark, so measuring at this point helps to gauge effectiveness trends over time. For the same reason, we measured at the third X-marks, fourth X-marks, and so on, but the participants in the upcoming experiment kept their debugging very incremental, which caused almost all debugging sessions to consist of two or fewer X-marks. Thus, we do not analyze marks beyond the second X-mark (except as they impact the fault localization feedback at the next point of measurement).
- *Last Test.* When users find the cause of a failure (a fault), they often *immediately* try to fix it. This point includes at least one X-mark and any number of checkmarks, and denotes the end of a debugging session. As such, it is the feedback point at which fault localization has the most information available to it, so technique effectiveness is also measured here.

Two points should become evident through this discussion. First, in interactive end-user environments, we clearly cannot measure technique effectiveness only at the end of testing, because doing so would not capture usage of the technique before the end of all testing actions. Second, while it would be possible to measure at additional feedback points in a debugging session, for reasons of statistical comparison, it was necessary to choose points that occur *uniformly* across all sessions.

We note that the need to evaluate at such points is not specific to our particular experiment. Rather, because the traditional measurement point of evaluating fault localization—at the end of testing or debugging—is insufficient in the domain of interactive debugging, *any* interactive fault localization technique must be evaluated on the basis of multiple feedback points. Without doing so, the experiment would be omitting most of the data reported by the technique.

CHAPTER 6

AN EXPERIMENT TO INVESTIGATE TWO ORTHOGONAL FACTORS IN FAULT LOCALIZATION

How important is each orthogonal factor of a fault localization technique to the effectiveness of that technique? Given the evidence of mistakes in interactive end-user testing, does each factor separately impact technique effectiveness in the presence of unreliable information? To shed some insight into these questions, we conducted an experiment investigating the following research questions:

RQ1: How do differences in information bases affect the effectiveness of a fault localization technique?

RQ2: How do differences in mappings affect the effectiveness of a fault localization technique?

RQ3: How does inaccurate information affect information bases and technique effectiveness?

RQ4: How does inaccurate information affect mappings and technique effectiveness?

One reason to investigate RQ1 in the context of our spreadsheet experiments is that the three information bases we have considered in that context are markedly different in cost. If there is no difference in their effectiveness, we can safely choose the least expensive; and if there is a difference in their effectiveness, that information can lend insights into which one to pursue.

Previous fault localization research often evaluates techniques as a whole, without considering the specific factors that contribute to observed results. We devised RQ2 because we suspected that mapping alone could be an important factor in determining technique effectiveness.

Our final two research questions were inspired by the unreliability in interactive end-user testing, which we have seen in previous empirical work [45, 58]. Differences in the information bases and mappings may be exaggerated, or diminished, when isolating situations where techniques must operate in the presence of unreliable information.

6.1 Design

In formulating our experiment, we considered three possible methodologies for gathering sources of data. The first possible methodology was to follow the classic human-subjects approach: gather participants for each possible mapping and information base combination and compare technique effectiveness across groups. This methodology has the advantage of eliciting test suites from real end users, but it has two drawbacks. First, given the nine possible combinations of information bases and mappings (techniques), it would require a very large number of subjects for proper statistical comparison. Second, and most importantly, each technique would be given differing testing actions, thereby making it impossible to ensure that differences in test suites were not confounding any results.

A second possible methodology was to follow a classic test suite generation approach: generate hypothetical test suites according to some criterion, and select (randomly or according to other criteria) from these test suites to simulate end users'

testing actions. We could then run each selected test suite under each technique, and compare effectiveness. This methodology features the tight controls we sought, but the test suites could not be tied to our ultimate users, and may not be representative of real end-user testing actions.

We chose instead to adopt a third methodology that draws upon advantages from both of the above, while avoiding their drawbacks. We obtained actual testing actions from real end users, and then uniformly applied these actions across all mapping and information base combinations. The test suites, as defined by the testing actions that the end users performed, were the subjects of our experiment. These test suites were sampled according to the methods outlined in Section 5.3.

6.1.1 Participants

To obtain the necessary test suites as subjects, we recruited 20 students (18 undergraduate students and 2 graduate students) from Oregon State University. We attempted to recruit students with spreadsheet experience because we did not want the learning of spreadsheet functionality to be a factor in our experiment. Of the 20 participants, 17 had at least some previous spreadsheet experience. We also sought participants without any personal or professional programming experience in order to make our participants more representative of real end users. (It is fairly common these days for business and engineering students to take a high school or college programming class.) Only one participant had professional programming experience, which consisted of writing a few basic spreadsheet macros using Visual Basic during a summer internship. The background of the 20 participants is summarized in Table 6.1. In the education category of this table, "LA" encodes the number

of liberal arts participant, “Bus” encodes business participants, “Eng” encodes engineering participants, and “GPA” encodes the average grade point average of all 20 participants. In the spreadsheet and programming categories, “HS” encodes the number of participants that used spreadsheets (or programmed) in a high school class, “C” encodes use in a college class, “Per” encodes personal use, and “Pro” encodes professional use.

In order for participants to include the use of a fault localization technique in their testing actions, some technique had to be incorporated into the environment for use by the participants. Because of their successes in Chapter 4 and earlier empirical work [45, 58], we chose to incorporate the I-TC information base with the M-BL mapping into the environment described in Chapters 2 and 3. We then applied the testing actions collected using this technique across all information base and mapping combinations.

6.1.2 Materials

The experiment utilized two spreadsheet programs, Gradebook and Payroll (shown in Figures 3.1 and 6.1, respectively). To make our programs representative of real end-user spreadsheet programs, Gradebook was derived from an Excel spreadsheet

Gender		Education				Spreadsheet				Programming			
M	F	LA	Bus	Eng	GPA	HS	C	Per	Pro	HS	C	Per	Pro
11	9	6	11	3	3.19	8	16	9	8	3	8	0	1

TABLE 6.1: A summary of the participants’ general, educational background, previous spreadsheet experience, and previous programming experience.

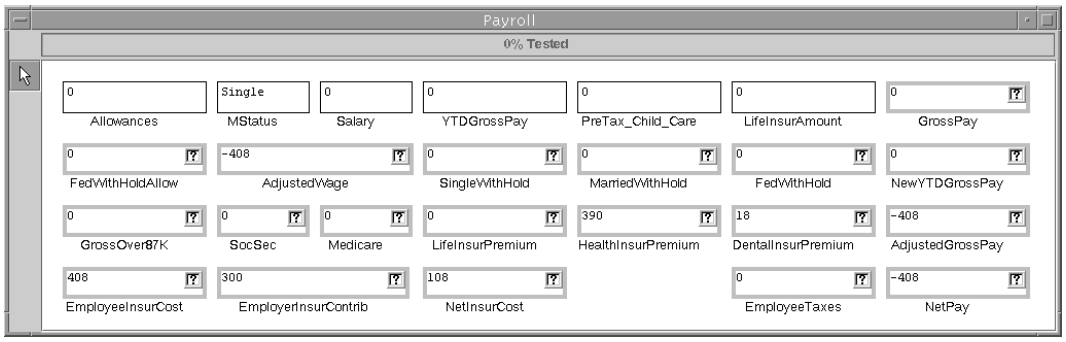

FIGURE 6.1: The Payroll task.

program of an (end-user) instructor, which we ported into an equivalent Forms/3 spreadsheet program. Payroll was a spreadsheet program designed by two Forms/3 researchers from a payroll description from a real company.

These spreadsheet programs were seeded with five faults created by real end users. To obtain these faults, we provided three separate end users with the following: (1) a “template” spreadsheet for each program with cells and cell names, but no cell formulas; and (2) a description of how each spreadsheet program should work, which included sample values and correct results for some cells. Each person was given as much time as he or she needed to design the spreadsheet program using the template and the description.

From the collection of faults left in these end users’ final spreadsheet programs, we chose five according to Allwood’s classification system [5]. Under Allwood’s system, mechanical faults include simple typographical errors or wrong cell references. Logical faults are mistakes in reasoning and are more difficult to correct than

mechanical faults. An omission fault is information that has never been entered into a cell formula, and is the most difficult to correct [5].

When classifying these faults, we realized that Allwood's scheme does not always clearly differentiate types of faults. For example, the scheme does not specify how to distinguish typographical mistakes (mechanical faults) from mistakes in reasoning (logical faults). In our study, if a seeded fault was a single incorrect character adjacent on the keyboard to the correct character (e.g., a 5 that should have been a 4), the fault was classified as a "mechanical" fault—the result of a typographical error. Faults were also classified as mechanical faults if they were due to mistakes in the placement of parentheses, erroneous operators, or incorrect cell references. If the fault was missing information, such as a missing cell reference, subexpression, or logical construct, it was classified as an "omission" fault. Otherwise, the fault was classified as a "logical" fault.

We seeded Gradebook with three mechanical faults, one logical fault, and one omission fault, and Payroll with two mechanical faults, two logical faults, and one omission fault. These faults are outlined in Tables 6.2 and 6.3. Payroll was intended to be the more difficult program due to its larger size, greater level of dataflow and intertwined dataflow relationships, and more difficult faults.

6.1.3 Procedures

After completing a background questionnaire, participants were given a brief tutorial to familiarize them with the environment. In the tutorial, participants performed actions on their own machines with guidance at each step. The tutorial taught use of WYSIWYT (checkmarks and associated feedback), but did not include any de-

Cell Name (*Fault Type*)	Faulty Formula	Correct Formula
Quiz_Avg (*Mechanical — Typographical*)	((Quiz1 + Quiz2 + Quiz3 + Quiz4 + Quiz5) - (min Quiz1 Quiz2 Quiz3 Quiz4 Quiz5)) / 5 else 0	((Quiz1 + Quiz2 + Quiz3 + Quiz4 + Quiz5) - (min Quiz1 Quiz2 Quiz3 Quiz4 Quiz5)) / 4 else 0
Midterm_Avg (*Mechanical — Parentheses*)	Midterm1_Perc + Midterm2 + Curved_Midterm3 - (min Midterm1_Perc Midterm2 Curved_Midterm3) / 2	(Midterm1_Perc + Midterm2 + Curved_Midterm3 - (min Midterm1_Perc Midterm2 Curved_Midterm3)) / 2
Course_Avg (*Mechanical — Operator*)	(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)
Exam_Avg (*Logical*)	(Midterm_Avg + Final_Percentage) / 3	(2 * Midterm_Avg + Final_Percentage) / 3
Curved_Midterm (*Omission*)	if Midterm3 $>$ 0 then 2	if Midterm3 $>$ 0 then Midterm3 + 2

TABLE 6.2: The faults seeded in the Gradebook task.

bugging or testing strategy content. We also did not teach 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. At the end of the tutorial, the participants were given five minutes to explore the program they were working on during the tutorial to allow them to work further with the features taught in the tutorial.

After the tutorial, participants were given the Gradebook and Payroll programs (tasks) with instructions to test and correct any errors found in the programs. The participants were also provided with two correct sequences of input and output for

Cell Name (*Fault Type*)	Faulty Formula	Correct Formula
MarriedWithHold (*Mechanical — Reference*)	if GrossPay ¡ 248 then 0 else (GrossPay - 248) * 0.10	if AdjustedWage ¡ 248 then 0 else (Adjusted-Wage - 248) * 0.10
AdjustedGrossPay (*Mechanical — Reference*)	GrossPay - PreTax_Child_Care - EmployeeInsurCost	GrossPay - PreTax_Child_Care - NetInsurCost
SingleWithHold (*Logical*)	if AdjustedWage ¡ 119 then 0 else (Adjusted-Wage - 248) * 0.10	if AdjustedWage ¡ 119 then 0 else (Adjusted-Wage - 119) * 0.10
SocSec (*Logical*) (*Omission*)	if GrossOver87K = 0 then (GrossPay * 0.062 * 0.0145) else (87000 * GrossPay * 0.062 * 0.0145)	if GrossOver87K = 0 then GrossPay * 0.062 else (GrossPay - GrossOver87K) * 0.062

TABLE 6.3: The faults seeded in the Payroll task. The SocSec cell has a fault in the "then" clause and a separate fault in the "else" clause. The fault in the "else" clause was classified as an omission fault because it was deemed that the clause was missing information regarding subtracting the amount over $87,000 from the gross pay before taking the tax of 6.2%.

each task to facilitate testing. The experiment was counterbalanced with respect to task order so as to distribute learning effects evenly. The tasks necessarily involved time limits—set at 20 minutes for Gradebook and 30 minutes for Payroll—to ensure participants worked on both spreadsheet programs, and to remove possible peer influence of some participants leaving early. To obtain the participants' testing actions during these two tasks, the actions by each participant were recorded into electronic transcripts.

6.1.4 Measures For Evaluation

As in Chapter 4, we define effectiveness as a technique's ability to correctly and visually differentiate the correct cells in a spreadsheet program from those cells that actually contain faults.

In this experiment, we considered two notions of effectiveness, resulting in two possible effectiveness metrics. The first possibility was to use the metric in Chapter 4, measuring the visual separation between all faulty cells and all correct cells of each spreadsheet program, regardless of whether those cells are colored or not. We abbreviate this metric Eff-All. The second possibility is to consider only those faulty and correct cells *that are actually colored* by the fault localization technique. By disregarding the cells that are not colored by a fault localization technique, this metric especially focuses on how well the technique prioritizes the sequence of the search by coloring certain cells darker than others. We abbreviate this metric Eff-Color.

As before, effectiveness is calculated by subtracting the average fault likelihood of the appropriate faulty cells from the average fault likelihood of the appropriate correct cells. Positive effectiveness is preferable, and a greater effectiveness indicates a better distinction between faulty and non-faulty cells. These effectiveness metrics are the dependent variable of our experiment, and are employed at every feedback point outlined in Section 5.3.

6.1.5 Threats to Validity

As in Chapter 4, we discuss the threats to this experiment's validity and, where possible, how we attempted to mitigate the impact of these threats on our results.

6.1.5.1 Threats to Internal Validity

The specific types of faults seeded in a program can affect fault localization results. To reduce this threat, as described in Section 6.1.2, we selected faults according to Allwood's classification scheme [5] to ensure that different types of faults were included.

As mentioned in Section 6.1.1, in order to apply the same test suites uniformly across all techniques, we had to obtain suites using a single information base and mapping, and we chose the I-TC information base and M-BL mapping. It is possible that the specific actions taken by participants, in response to fault localization feedback, would have varied had a different information base or mapping been chosen. This tradeoff was necessary in order to obtain uniform test suites, as we have already explained. Had we chosen a design that allowed for varying testing actions, we would have risked confounding the independent variable—information base or mapping selection—with a second variable of varying testing actions.

Another threat to the internal validity of the experiment is the possibility that participants may not have understood programs' functionality sufficiently to correct the faults. Also, the study's time limits could have interacted with the learning styles of some participants. However, as described in Section 6.1.3, one reason our study involved time limits was to eliminate another threat to internal validity: peer influence as the result of some participants leaving early.

6.1.5.2 Threats to Construct Validity

It is possible that other metrics could better measure how well techniques provide fault localization feedback. To reduce this threat, we used two metrics to measure the effectiveness of our fault localization techniques.

We chose to measure at the First X-mark, Second X-mark, and Last Test points because they provide a snapshot of a techniques performance with minimal feedback and when the feedback was sufficient for the user to determine which formula to edit. Measuring effectiveness at other feedback points in debugging sessions could have yielded valuable information that is not captured by our experiment design. However, our analysis reveals that the number of X-marks (failures) placed in each session rarely exceeded two, thereby limiting the number of such points.

6.1.5.3 Threats to External Validity

Program representativeness is an issue facing our experiment. If the programs used in our experiment did not mimic those that real end users create, our results may not generalize. To reduce this threat, we selected "real-world" spreadsheet programs from a real end-user instructor and a real payroll description. Also, to better control experiment conditions and ensure that subjects could complete the tasks in the allotted time, our programs were not large. End user programs can be in varying sizes; however, our results should be taken in the context of this limitation. Future empirical work gathering additional empirical evidence using a greater range and number of programs would reduce this threat.

The ability to generalize our results may also be limited by our selection of faults. We attempted to address this issue by seeding "real-world" faults into our

tasks using the procedures outlined in Section 6.1.2. Also, to help control the threats to internal validity, we selected faults from the end users according to Allwood's classification scheme [5]. However, this came at a cost of some external validity, because the fault patterns of end users may differ from those introduced into our experiment tasks.

Finally, our experiment was conducted in the Forms/3 research language [15]. However, end users may debug differently in a different language. All of these external validity concerns can be addressed only through repeated studies, using different programs, faults, and languages.

6.1.5.4 Threats to Conclusional Validity

The power of our conclusions would be enhanced had we used more than two spreadsheet programs. One way we attempted to mitigate this concern was selecting programs with varying degrees of complexity. Our study's conclusions would also be strengthened had we used more than three information bases and mappings. For this reason, we choose information bases and mappings from three distinctly different techniques with varying costs.

6.2 Results

6.2.1 RQ1: The Information Base Factor

To investigate RQ1, which pertains to the different information bases' impact on technique effectiveness in isolation from the mapping factor, we compared the information bases' effectiveness three times, once under each mapping described in

Section 5.2. The comparisons were done at the three feedback points described in Section 5.3.

As a statistical vehicle for our analyses, we state the following (null) hypotheses:

H1: *There is no difference in the effectiveness of the three information bases with the M-TC mapping.*

H2: *There is no difference in the effectiveness of the three information bases with the M-BL mapping.*

H3: *There is no difference in the effectiveness of the three information bases with the M-NC mapping.*

The results using both the Eff-Color and Eff-All metrics are shown in Tables 6.4–6.6. We used the Friedman test to statistically analyze the data. This test is an alternative to the repeated measures ANOVA when the assumption of normality or equality is not met. (We did not run Friedman tests on the Second X-mark data due to the small sample sizes.)

Table 6.4(a) shows marginal significance (at the 0.10 level) at the Last Test of the Payroll task using the Eff-Color metric. Table 6.4(b) corroborates this finding, showing significance (at the 0.01 level) at the same point using the Eff-All metric. Therefore, we reject H1.

Similar trends were found using the M-BL mapping to isolate the information base factor. Table 6.5(a) shows marginal significance (at the 0.10 level) and 0.01 level significance at the First X-mark and Last Test, respectively, of Payroll, while Table 6.5(b) shows marginal significance at the Last Test of Payroll. Given these differences, especially at the Last Test of the larger Payroll task, we reject H2.

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.8948$) ($n = 18$)	**0.389 (0.502)**	0.259 (0.622)	**0.389 (0.502)**
Payroll ($p = 0.1211$) ($n = 13$)	0.000 (0.000)	-0.166 (0.404)	**0.039 (0.075)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.000 (1.000)	**0.166 (0.764)**	0.000 (1.000)
Payroll ($p = n/a$) ($n = 5$)	0.155 (0.458)	0.011 (0.122)	**0.183 (0.489)**

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.4389$) ($n = 18$)	-0.056 (0.539)	**0.004 (0.493)**	-0.038 (0.512)
Payroll ($p = 0.0608$) ($n = 13$)	0.127 (0.280)	-0.118 (0.476)	**0.210 (0.497)**

(a) The Eff-Color metric.

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.6065$) ($n = 18$)	**0.170 (0.156)**	0.130 (0.222)	**0.170 (0.156)**
Payroll ($p = 0.1637$) ($n = 13$)	-0.041 (0.160)	-0.046 (0.115)	**0.057 (0.157)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	**0.133 (0.231)**	**0.133 (0.340)**	**0.133 (0.231)**
Payroll ($p = n/a$) ($n = 5$)	**0.354 (0.402)**	0.175 (0.181)	0.342 (0.411)

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.8890$) ($n = 18$)	0.027 (0.204)	0.027 (0.204)	**0.034 (0.205)**
Payroll ($p = 0.0018$) ($n = 13$)	0.145 (0.388)	0.434 (0.338)	**0.437 (0.486)**

(b) The Eff-All metric.

TABLE 6.4: Isolating the information base factor with mapping M-TC. The mean (standard deviation) effectiveness values comparing the three information bases with the M-TC mapping are shown. The information base with the greatest average effectiveness is shown in bold. The "p" denotes p-values of the Friedman tests, and "n" denotes the number of subjects measured at each point.

Differences were even more pronounced using the M-NC mapping, as shown in Table 6.6. This was especially true using the Eff-Color metric (Table 6.6(a)) where, for both tasks, statistical differences were at the 0.10 level at the First X-mark, and at the 0.05 and 0.01 levels at the Last Test. We reject H3.

Although the Friedman test reveals only whether there is a difference among the three information bases in the measured settings, Tables 6.4–6.6 indicate that the I-NC information base, which is the basis of the inexpensive Nearest Consumers tech-

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.7165$) ($n = 18$)	0.831 (0.841)	0.859 (1.005)	**0.943 (0.923)**
Payroll ($p = 0.1000$) ($n = 13$)	0.294 (0.425)	0.347 (0.327)	**0.487 (0.397)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.333 (2.082)	**0.500 (1.803)**	**0.500 (1.803)**
Payroll ($p = n/a$) ($n = 5$)	0.372 (0.947)	0.359 (0.595)	**0.560 (0.830)**

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.4464$) ($n = 18$)	0.265 (1.116)	0.331 (1.063)	**0.376 (1.034)**
Payroll ($p = 0.0128$) ($n = 13$)	0.302 (0.602)	0.596 (0.532)	**0.768 (0.664)**

(a) The Eff-Color metric.

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.6065$) ($n = 18$)	0.351 (0.188)	0.368 (0.269)	**0.376 (0.185)**
Payroll ($p = 0.1353$) ($n = 13$)	0.026 (0.175)	0.098 (0.293)	**0.221 (0.249)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.350 (0.541)	**0.433 (0.404)**	**0.433 (0.404)**
Payroll ($p = n/a$) ($n = 5$)	0.621 (0.741)	**0.757 (0.593)**	0.699 (0.740)

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.3679$) ($n = 18$)	0.180 (0.303)	0.196 (0.384)	**0.231 (0.286)**
Payroll ($p = 0.0555$) ($n = 13$)	0.574 (0.505)	0.986 (0.788)	**0.878 (0.700)**

(b) The Eff-All metric.

TABLE 6.5: Isolating the information base factor with mapping M-BL. The mean (standard deviation) effectiveness values comparing the three information bases with the M-BL mapping are shown.

nique [56], may be the most effective of the three information bases—I-NC showed the highest average effectiveness at almost every point measured. Implications of this will be discussed in Section 6.3.

6.2.2 RQ2: The Mapping Factor

How important is mapping alone to technique effectiveness? The tables in Section 6.2.1 are suggestive in this regard. To statistically consider whether this factor had a

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.0923$) ($n = 18$)	1.146 (1.322)	1.394 (1.374)	**1.496 (1.385)**
Payroll ($p = 0.0695$) ($n = 13$)	0.690 (0.784)	0.543 (0.652)	**0.935 (0.796)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.500 (2.783)	0.833 (2.566)	**1.000 (2.646)**
Payroll ($p = n/a$) ($n = 5$)	0.693 (1.170)	0.542 (1.055)	**0.936 (1.217)**

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.0022$) ($n = 18$)	0.274 (1.519)	0.756 (1.477)	**0.791 (1.612)**
Payroll ($p = 0.0199$) ($n = 13$)	0.646 (0.859)	0.833 (0.879)	**1.268 (0.955)**

(a) The Eff-Color metric.

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.1324$) ($n = 18$)	0.491 (0.315)	0.579 (0.377)	**0.581 (0.244)**
Payroll ($p = 0.1382$) ($n = 13$)	0.135 (0.337)	0.141 (0.294)	**0.370 (0.412)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.483 (0.725)	0.650 (0.606)	**0.733 (0.643)**
Payroll ($p = n/a$) ($n = 5$)	0.939 (0.886)	**1.056 (0.981)**	**1.056 (1.079)**

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.0072$) ($n = 18$)	0.212 (0.459)	0.390 (0.492)	**0.427 (0.418)**
Payroll ($p = 0.1017$) ($n = 13$)	0.894 (0.699)	1.212 (0.976)	**1.246 (0.965)**

(b) The Eff-All metric.

TABLE 6.6: Isolating the information base factor with mapping M-NC. The mean (standard deviation) effectiveness values comparing the three information bases with the M-NC mapping are shown.

significant impact on effectiveness, we used the Friedman test to compare the mappings' effectiveness under each information base, for the following hypotheses:

H4: *There is no difference in the effectiveness of the I-TC information base with different mappings.*

H5: *There is no difference in the effectiveness of the I-BL information base with different mappings.*

H6: *There is no difference in the effectiveness of the I-NC information base with different mappings.*

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0031$) ($n = 18$)	0.389 (0.502)	0.831 (0.841)	**1.146 (1.322)**
Payroll ($p = 0.0060$) ($n = 13$)	0.000 (0.000)	0.294 (0.425)	**0.690 (0.784)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.000 (1.000)	0.333 (2.082)	**0.500 (2.784)**
Payroll ($p = n/a$) ($n = 5$)	0.155 (0.458)	0.372 (0.947)	**0.693 (1.170)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.1180$) ($n = 18$)	-0.056 (0.539)	0.265 (1.116)	**0.274 (1.519)**
Payroll ($p = 0.1220$) ($n = 13$)	0.128 (0.280)	0.302 (0.602)	**0.646 (0.859)**

(a) The Eff-Color metric.

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p < 0.0001$) ($n = 18$)	0.170 (0.156)	0.351 (0.188)	**0.491 (0.315)**
Payroll ($p = 0.2490$) ($n = 13$)	-0.041 (0.160)	0.026 (0.175)	**0.135 (0.337)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.133 (0.160)	0.350 (0.541)	**0.483 (0.725)**
Payroll ($p = n/a$) ($n = 5$)	0.354 (0.402)	0.621 (0.741)	**0.939 (0.886)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0285$) ($n = 18$)	0.027 (0.204)	0.180 (0.303)	**0.212 (0.459)**
Payroll ($p = 0.0025$) ($n = 13$)	0.434 (0.338)	0.574 (0.505)	**0.894 (0.699)**

(b) The Eff-All metric.

TABLE 6.7: Isolating the mapping factor with information base I-TC. The mean (standard deviation) effectiveness values comparing the three mappings with the I-TC information base are shown.

As Tables 6.7–6.9 show, for all three information bases, mapping M-NC was consistently the most effective. Even more, the Friedman tests reveal significant differences in technique effectiveness among the different mappings at the First X-mark and the Last Test points of measurement. These differences were almost always significant at the 0.05 level, and often significant at the 0.01 level. Clearly, H4, H5, and H6 must all be rejected.

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0004$) ($n = 18$)	0.259 (0.622)	0.859 (1.005)	**1.394 (1.374)**
Payroll ($p = 0.0016$) ($n = 13$)	-0.166 (0.404)	0.347 (0.327)	**0.543 (0.652)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.167 (0.764)	0.500 (1.803)	**0.833 (2.566)**
Payroll ($p = n/a$) ($n = 5$)	0.011 (0.122)	0.358 (0.595)	**0.542 (1.055)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0424$) ($n = 18$)	0.004 (0.493)	0.331 (1.063)	**0.756 (1.477)**
Payroll ($p = 0.0001$) ($n = 13$)	-0.118 (0.476)	0.596 (0.532)	**0.833 (0.878)**

(a) The Eff-Color metric.

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p < 0.0001$) ($n = 18$)	0.130 (0.222)	0.368 (0.269)	**0.579 (0.377)**
Payroll ($p = 0.3526$) ($n = 13$)	-0.046 (0.115)	0.098 (0.293)	**0.141 (0.294)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.133 (0.340)	0.433 (0.404)	**0.650 (0.606)**
Payroll ($p = n/a$) ($n = 5$)	0.175 (0.181)	0.757 (0.593)	**1.056 (0.981)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0056$) ($n = 18$)	0.027 (0.204)	0.196 (0.384)	**0.390 (0.492)**
Payroll ($p = 0.0001$) ($n = 13$)	0.145 (0.388)	0.986 (0.788)	**1.212 (0.976)**

(b) The Eff-All metric.

TABLE 6.8: Isolating the mapping factor with information base I-BL. The mean (standard deviation) effectiveness values comparing the three mappings with the I-BL information base are shown.

6.2.3 RQ3: Information Base Robustness

As our first step to investigate the pervasiveness of mistakes, we counted the number of incorrect testing decisions made in each subject (end-user test suite). In the context of our environment, this is either a WYSIWYT checkmark, signifying a correct value placed in a cell that really has an incorrect value; or an X-mark, signifying an incorrect value (a failure) placed in a cell that really has a correct value. In the Gradebook task, 8.99% of the checkmarks and 5.95% of the X-marks were incorrect. This trend continued in Payroll, where 20.62% of the checkmarks and

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0001$) ($n = 18$)	0.389 (0.502)	0.943 (0.923)	**1.496 (1.385)**
Payroll ($p = 0.0005$) ($n = 13$)	0.039 (0.075)	0.488 (0.397)	**0.935 (0.796)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.000 (1.000)	0.500 (1.803)	**1.000 (2.646)**
Payroll ($p = n/a$) ($n = 5$)	0.183 (0.489)	0.560 (0.830)	**0.936 (1.217)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0045$) ($n = 18$)	-0.039 (0.512)	0.376 (1.034)	**0.791 (1.612)**
Payroll ($p = 0.0036$) ($n = 13$)	0.210 (0.497)	0.768 (0.664)	**1.268 (0.955)**

(a) The Eff-Color metric.

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p < 0.0001$) ($n = 18$)	0.170 (0.156)	0.376 (0.185)	**0.581 (0.244)**
Payroll ($p = 0.0366$) ($n = 13$)	0.057 (0.157)	0.221 (0.249)	**0.369 (0.412)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.133 (0.231)	0.433 (0.404)	**0.733 (0.643)**
Payroll ($p = n/a$) ($n = 5$)	0.342 (0.411)	0.699 (0.740)	**1.056 (1.079)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0001$) ($n = 18$)	0.034 (0.205)	0.231 (0.286)	**0.427 (0.418)**
Payroll ($p = 0.0007$) ($n = 13$)	0.437 (0.486)	0.878 (0.700)	**1.246 (0.965)**

(b) The Eff-All metric.

TABLE 6.9: Isolating the mapping factor with information base I-BL. The mean (standard deviation) effectiveness values comparing the three mappings with the I-BL information base are shown.

3.33% of the X-marks were incorrect. These results corroborate the findings found in earlier formative studies [45, 58].

Clearly, the large number of incorrect testing decisions means that the information bases and mappings were corrupted with incorrect information. Given that such mistakes corrupt information bases, how did these mistakes impact an information base's effect on technique effectiveness? To investigate this, we measured effectiveness at each First X-mark, Second X-mark, and Last Test *that was in the context*

of at least one incorrect testing decision. We isolated information bases using the same procedure as in Section 6.2.1.

H7: *There is no difference in the effectiveness of the three information bases with the M-TC mapping when feedback is provided in the context of mistakes.*

H8: *There is no difference in the effectiveness of the three information bases with the M-BL mapping when feedback is provided in the context of mistakes.*

H9: *There is no difference in the effectiveness of the three information bases with the M-NC mapping when feedback is provided in the context of mistakes.*

As can be seen in Tables 6.10 and 6.11, there were no significant differences (at the 0.05 or 0.01 levels) among the techniques, so we cannot reject H7 or H8. However, in Table 6.12, at the last test of debugging sessions, the differences in each information base's effectiveness were marginally significant for Payroll, and significant (at the 0.01 level) for Gradebook. Therefore, we reject H9.

More important though, as shown in all three of these tables, all three information bases were able to provide effective feedback (indicated by positive values in the table) in most cases, even in the presence of user mistakes. (As one would expect, the mistakes appeared to have an impact on technique effectiveness. Although there was almost no change in effectiveness at the First X-mark feedback point due to incorrect testing decisions, by the Last Test feedback point, many effectiveness measures were adversely affected.) This ability to provide effective feedback in these scenarios may be in part due to the help of our robustness feature. However, further research would be required in order to test this possibility.

Another interesting trend in the data is that, once again, the I-NC information base usually showed the highest average effectiveness, even in the context of these

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.8669$) ($n = 13$)	**0.769 (1.013)**	0.436 (1.013)	**0.769 (1.013)**
Payroll ($p = 0.2231$) ($n = 10$)	0.000 (0.000)	-0.202 (0.458)	**0.051 (0.083)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.333 (1.528)	**0.500 (1.323)**	0.333 (1.528)
Payroll ($p = n/a$) ($n = 3$)	0.292 (0.505)	0.067 (0.115)	**0.314 (0.543)**

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.3679$) ($n = 13$)	-0.308 (0.630)	**-0.187 (0.673)**	-0.285 (0.608)
Payroll ($p = 0.1030$) ($n = 10$)	0.149 (0.286)	-0.195 (0.465)	**0.159 (0.394)**

(a) The Eff-Color metric.

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.5134$) ($n = 13$)	**0.385 (0.215)**	0.283 (0.295)	**0.385 (0.215)**
Payroll ($p = 0.1211$) ($n = 10$)	-0.081 (0.364)	-0.050 (0.227)	**0.155 (0.335)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	**0.350 (0.409)**	**0.350 (0.541)**	**0.350 (0.409)**
Payroll ($p = n/a$) ($n = 3$)	**0.454 (0.606)**	0.347 (0.134)	0.394 (0.592)

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.8669$) ($n = 13$)	0.065 (0.385)	0.056 (0.392)	**0.076 (0.395)**
Payroll ($p = 0.0727$) ($n = 10$)	**0.704 (0.439)**	0.403 (0.307)	0.609 (0.478)

(b) The Eff-All metric.

TABLE 6.10: Isolating the information base factor with mapping M-TC for feedback points that were in the context of at least one incorrect testing decision. The mean (standard deviation) effectiveness values comparing the three information bases with the M-TC mapping are shown.

testing mistakes. We discuss possible reasons for this superior robustness in Section 6.3.

6.2.4 RQ4: Mapping Robustness

In the context of at least one incorrect testing decision, some of the differences in information base effectiveness tended to disappear. Would the same trend hold true for the mapping factor?

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.7165$) ($n = 13$)	0.821 (0.798)	0.859 (1.032)	**0.974 (0.915)**
Payroll ($p = 0.1309$) ($n = 10$)	0.262 (0.416)	0.317 (0.296)	**0.500 (0.398)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.333 (2.082)	**0.500 (1.803)**	**0.500 (1.803)**
Payroll ($p = n/a$) ($n = 3$)	0.743 (0.908)	0.492 (0.709)	**0.775 (0.999)**

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.4464$) ($n = 13$)	0.036 (1.092)	0.128 (1.037)	**0.190 (1.007)**
Payroll ($p = 0.0539$) ($n = 10$)	0.311 (0.587)	0.488 (0.394)	**0.698 (0.610)**

(a) The Eff-Color metric.

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.6065$) ($n = 13$)	0.363 (0.143)	0.386 (0.268)	**0.397 (0.132)**
Payroll ($p = 0.2053$) ($n = 10$)	0.024 (0.192)	0.087 (0.319)	**0.267 (0.260)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.350 (0.541)	0.433 (0.404)	**0.433 (0.404)**
Payroll ($p = n/a$) ($n = 3$)	**0.729 (0.956)**	0.699 (0.746)	0.706 (1.026)

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.3679$) ($n = 13$)	0.159 (0.289)	0.182 (0.405)	**0.229 (0.267)**
Payroll ($p = 0.1988$) ($n = 10$)	0.584 (0.541)	**0.923 (0.681)**	0.827 (0.649)

(b) The Eff-All metric.

TABLE 6.11: Isolating the information base factor with mapping M-BL for feedback points that were in the context of at least one incorrect testing decision. The mean (standard deviation) effectiveness values comparing the three information bases with the M-BL mapping are shown.

H10: *There is no difference in the effectiveness of the three information bases with the M-TC mapping when feedback is provided in the context of mistakes.*

H11: *There is no difference in the effectiveness of the three information bases with the M-BL mapping when feedback is provided in the context of mistakes.*

H12: *There is no difference in the effectiveness of the three information bases with the M-NC mapping when feedback is provided in the context of mistakes.*

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.1251$) ($n = 13$)	1.103 (1.274)	1.423 (1.484)	**1.564 (1.377)**
Payroll ($p = 0.1095$) ($n = 10$)	0.683 (0.789)	0.452 (0.550)	**0.947 (0.800)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.500 (2.784)	0.833 (2.566)	**1.000 (2.646)**
Payroll ($p = n/a$) ($n = 3$)	**1.278 (1.001)**	0.850 (1.202)	1.235 (1.522)

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.0024$) ($n = 13$)	-0.105 (1.364)	0.577 (1.566)	**0.587 (1.647)**
Payroll ($p = 0.0665$) ($n = 10$)	0.698 (0.841)	0.667 (0.709)	**1.163 (0.939)**

(a) The Eff-Color metric.

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.1837$) ($n = 13$)	0.513 (0.198)	0.595 (0.382)	**0.603 (0.210)**
Payroll ($p = 0.1390$) ($n = 10$)	0.145 (0.367)	0.121 (0.286)	**0.427 (0.441)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.483 (0.275)	0.650 (0.606)	**0.733 (0.643)**
Payroll ($p = n/a$) ($n = 3$)	**1.120 (1.150)**	1.044 (1.232)	1.058 (1.500)

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.0078$) ($n = 13$)	0.177 (0.406)	0.391 (0.539)	**0.438 (0.411)**
Payroll ($p = 0.4423$) ($n = 10$)	0.937 (0.774)	1.113 (0.837)	**1.174 (0.959)**

(b) The Eff-All metric.

TABLE 6.12: Isolating the information base factor with mapping M-NC for feedback points that were in the context of at least one incorrect testing decision. The mean (standard deviation) effectiveness values comparing the three information bases with the M-NC mapping are shown.

In the context of at least one incorrect testing decision, Table 6.13 shows significant differences in effectiveness using difference mappings. This trend continued in Tables 6.14 and 6.15. These differences were almost always significant at the 0.05 or 0.01 levels, so we must reject H10, H11, and H12.

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.2813$) ($n = 13$)	0.769 (1.013)	0.821 (0.798)	**1.103 (1.274)**
Payroll ($p = 0.0536$) ($n = 10$)	0.000 (0.000)	0.262 (0.416)	**0.683 (0.789)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.333 (1.528)	0.333 (2.082)	**0.500 (2.784)**
Payroll ($p = n/a$) ($n = 3$)	0.292 (0.505)	0.743 (0.909)	**1.278 (1.001)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.2847$) ($n = 13$)	-0.308 (0.630)	**0.036 (1.092)**	-0.105 (1.364)
Payroll ($p = 0.1128$) ($n = 10$)	0.149 (0.286)	0.311 (0.587)	**0.698 (0.841)**

(a) The Eff-Color metric.

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0038$) ($n = 13$)	0.385 (0.215)	0.363 (0.143)	**0.513 (0.198)**
Payroll ($p = 0.0275$) ($n = 10$)	-0.081 (0.364)	0.024 (0.192)	**0.145 (0.367)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.350 (0.409)	0.350 (0.541)	**0.483 (0.725)**
Payroll ($p = n/a$) ($n = 3$)	-0.081 (0.364)	0.024 (0.192)	**0.145 (0.367)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.1905$) ($n = 13$)	0.065 (0.385)	0.159 (0.289)	**0.177 (0.406)**
Payroll ($p = 0.0622$) ($n = 10$)	0.704 (0.439)	0.584 (0.541)	**0.938 (0.774)**

(b) The Eff-All metric.

TABLE 6.13: Isolating the mapping factor with information base I-TC for feedback points that were in the context of at least one incorrect testing decision. The mean (standard deviation) effectiveness values comparing the three mappings with the I-TC information base are shown.

6.3 Discussion

6.3.1 The Information Base Factor

The results regarding RQ1 showed that the information base factor can make a significant difference in technique effectiveness. This result is in keeping with tradition. We also found that the information bases' differences in effectiveness were most pronounced at the end of debugging sessions, most likely due to the increased testing information available at the end of a session, allowing the techniques a

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0052$) ($n = 13$)	0.436 (1.013)	0.859 (1.032)	**1.423 (1.484)**
Payroll ($p = 0.0204$) ($n = 10$)	-0.202 (0.458)	0.317 (0.296)	**0.452 (0.550)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.500 (1.323)	0.500 (1.803)	**0.833 (2.566)**
Payroll ($p = n/a$) ($n = 3$)	0.067 (0.115)	0.492 (0.709)	**0.850 (1.202)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.1985$) ($n = 13$)	-0.187 (0.673)	0.128 (1.037)	**0.577 (1.566)**
Payroll ($p = 0.0013$) ($n = 10$)	-0.195 (0.465)	0.488 (0.709)	**0.667 (0.709)**

(a) The Eff-Color metric.

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0016$) ($n = 13$)	0.283 (0.295)	0.386 (0.268)	**0.595 (0.382)**
Payroll ($p = 0.2366$) ($n = 10$)	-0.050 (0.227)	0.087 (0.319)	**0.121 (0.286)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.350 (0.541)	0.433 (0.404)	**0.650 (0.606)**
Payroll ($p = n/a$) ($n = 3$)	0.347 (0.134)	0.699 (0.746)	**1.044 (1.232)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0401$) ($n = 13$)	0.056 (0.392)	0.182 (0.539)	**0.391 (0.539)**
Payroll ($p = 0.0004$) ($n = 10$)	0.403 (0.307)	0.923 (0.681)	**1.113 (0.837)**

(b) The Eff-All metric.

TABLE 6.14: Isolating the mapping factor with information base I-BL for feedback points that were in the context of at least one incorrect testing decision. The mean (standard deviation) effectiveness values comparing the three mappings with the I-BL information base are shown.

greater opportunity to differentiate themselves from each other. However, a surprise was that effectiveness did not always get better as debugging sessions progressed—in the case of Gradebook, all nine information bases and mappings consistently got worse. We believe this may relate to the mistakes the users made in their testing, a point we will return to shortly. The implied importance of the information base factor indicates that researchers could serve end users, and the software they create, by investing effort into devising information bases for end-user fault localization

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0040$) ($n = 13$)	0.769 (1.013)	0.974 (0.915)	**1.564 (1.377)**
Payroll ($p = 0.0084$) ($n = 10$)	0.051 (0.083)	0.500 (0.398)	**0.947 (0.800)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.333 (1.528)	0.500 (1.803)	**1.000 (2.646)**
Payroll ($p = n/a$) ($n = 3$)	0.314 (0.543)	0.775 (0.999)	**1.235 (1.522)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0354$) ($n = 13$)	-0.285 (0.608)	0.190 (1.007)	**0.587 (1.664)**
Payroll ($p = 0.0450$) ($n = 10$)	0.159 (0.394)	0.698 (0.610)	**1.163 (0.939)**

(a) The Eff-Color metric.

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0008$) ($n = 13$)	0.385 (0.215)	0.397 (0.132)	**0.603 (0.210)**
Payroll ($p = 0.0283$) ($n = 10$)	0.155 (0.335)	0.267 (0.260)	**0.427 (0.441)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.350 (0.409)	0.433 (0.404)	**1.000 (0.643)**
Payroll ($p = n/a$) ($n = 3$)	0.394 (0.592)	0.706 (1.026)	1.058 (1.500)

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0115$) ($n = 13$)	0.076 (0.395)	0.229 (0.267)	**0.438 (0.411)**
Payroll ($p = 0.0450$) ($n = 10$)	0.609 (0.478)	0.827 (0.649)	**1.174 (0.959)**

(b) The Eff-All metric.

TABLE 6.15: Isolating the mapping factor with information base I-NC for feedback points that were in the context of at least one incorrect testing decision. The mean (standard deviation) effectiveness values comparing the three mappings with the I-NC information base are shown.

techniques, just as has been done for professional programmers' fault localization techniques.

Another surprise was the superior effectiveness of the I-NC information base. The first surprising aspect of this result is the fact that this information base is the *least computationally expensive* of the three we compared. The second surprising aspect of this result is that the I-NC information base is the information base least like those of many traditional fault localization techniques, which tend to use counts

of passed and failed tests (as does I-TC) or dicing-like approaches (as does I-BL) to generate feedback.

In generalizing this experience, the first lesson to researchers may be that the most expensive and intelligent information base *may not always be the most effective*. In fact, from a cost-effectiveness standpoint, a simple, inexpensive technique may be sufficient for end users programming in spreadsheet environments. Second, researchers may find that techniques employing non-traditional measures generate the most effective feedback in the spreadsheet paradigm. Future research may shed some insights into whether similar lessons are indicated in other end-user programming paradigms.

We were surprised at the role of the information base factor in the presence of user mistakes (RQ3). We had expected that this factor would be the most important factor in providing quality feedback in the presence of these mistakes. Contrary to expectations, when using the M-TC and M-BL, the information base factor often did not make a significant difference in effectiveness. This comes *despite the importance of the information base factor from the investigation in RQ1*. A likely reason is that the information bases themselves are corrupted by such mistakes. This corruption may generally mitigate any differences among information bases. But because significant differences were found when using the M-NC mapping to isolate this factor, future research is needed to determine the importance of the information base factor in the presence of user mistakes by using a greater number of information base and mapping combinations on a wider variety of subjects.

6.3.2 The Mapping Factor

Turning to RQ2, the role of mapping in the fault localization techniques' performance was quite pronounced. The significant differences occurred despite only small distinctions among the way the three mappings were done (described in Section 5.2).

This result has two implications. First, regarding the design of fault localization techniques, our results suggest that because mapping plays such a critical role, great care should be exercised in selecting what mapping to include in a fault localization technique. The second implication concerns the evaluation of fault localization techniques. Since the information base and mapping factors had significant, *independent* roles in the techniques' effectiveness, our results suggest that evaluating each factor separately is necessary in order to obtain accurate information as to the effectiveness of a fault localization technique. In fact, researchers may find that some mappings consistently perform better than others, as we found in this particular experiment with M-NC. To our knowledge, this work is the first to formally suggest the possible importance of evaluating each factor of a fault localization technique separately.

The mapping factor continued to play a significant role in technique effectiveness when considering situations where at least one incorrect testing decision had been made (RQ4). This result may not seem terribly surprising given the results for RQ2, but it is quite surprising given the results for RQ3, where the information base factor often did not make a significant difference.

The immediate consequence of these results is a reinforcement of the importance of the mapping factor, and the care that researchers should take when choos-

ing a mapping for their own fault localization techniques. In fact, these results suggest that the mapping factor may be even more important than the information base factor. Further research, of course, is needed to place additional confidence in this statement.

CHAPTER 7

CONCLUSIONS AND IMPLICATIONS FOR FUTURE WORK

End-user programmers are writing an unprecedented number of programs, due in large part to the significant effort put forth to bring programming power to end users. Unfortunately, this effort had not been supplemented by a comparable effort to increase the correctness of these often faulty programs, especially with respect to the debugging tasks that end-user programmers inevitably must perform.

To begin to address this need, we have been working towards bringing fault localization techniques to end users. This thesis presents algorithms for three fault localization techniques, two of which are introduced for the first time in this thesis. These fault localization techniques aim to reduce a user's search space in debugging, and also to prioritize the sequence of the search through that space. This is accomplished by estimating the likelihood that each program point (in our prototype, spreadsheet cells) contains a fault that contributed to failures observed by the user, and communicating this feedback to the user using simple visual mechanisms. The three techniques presented in this thesis have varying costs associated with making these estimations, providing researchers and developers of end-user programming environments with cost-effectiveness considerations regarding which technique(s) to pursue.

To investigate how well these previously unevaluated techniques provide fault localization feedback, and to find ways that we could improve the techniques, we

conducted a formative experiment to measure the effectiveness of each technique at the end of testing—after all information has been gathered—and very early: just after the first failure is observed. Measuring feedback at the end of testing is often considered in fault localization research; however, measuring early feedback, which is often not considered, is important because it is at this point that a technique would first give a user visual feedback.

This investigation indicated significant differences among the effectiveness of our techniques in both the cases of early feedback and at the end of testing. Also, because previous research [45, 58] has indicated that end users make mistakes when interactively testing and debugging, we investigated technique effectiveness in the presence of mistakes in testing information. We found that there were often significant differences in the effectiveness of the fault localization feedback provided by our techniques in the presence of mistakes. This indicates that developers of future fault localization techniques should consider how well those techniques provide quality feedback in the presence of mistakes.

In seeking to improve our techniques' effectiveness in light of this experiment's findings, we found evidence that there exist two orthogonal factors in fault localization techniques that can play important roles in providing fault localization feedback. To tease apart the differences between these factors and their impact on the effectiveness of fault localization feedback, we conducted a second empirical study to isolate the impact of these factors in fault localization. This study indicated that both the information base and the mapping factor can each have a significant impact on the effectiveness of fault localization techniques. In fact, the mapping factor may play an even more critical role in technique effectiveness than the information base factor, especially in the presence of incorrect testing information.

This thesis has several implications for future work into bringing interactive fault localization techniques to end-user programmers. First, throughout the two empirical studies in thesis, we found differences in the effectiveness of our three fault localization techniques. Our computationally expensive techniques, designed to harness as much testing information as possible, were at times outperformed by our less expensive technique. Designers of future fault localization techniques may find that some expensive fault localization techniques may not be as cost-effective as some other less expensive techniques.

Second, this thesis evaluated fault localization techniques' effectiveness very early in debugging, when the techniques must provide initial feedback to the user. Because prior fault localization research usually evaluates techniques only at the point of maximal system reasoning potential—when the system has its best (and only) chance of producing correct feedback—a contribution of this thesis is to break from traditional fault localization research by highlighting the need to evaluate techniques early in the interactive debugging, when very little information may be available on which to base fault localization feedback.

Third, this thesis has reinforced the need to evaluate interactive fault localization techniques in the presence of mistakes and incorrect testing information. This thesis has also corroborated earlier findings [45, 58] that such mistakes are commonly made by end users performing interactive testing and debugging in spreadsheet environments. This contribution is important because it goes against another traditional assumption in fault localization research: that all information provided to the technique is correct.

Finally, our results suggest that the mapping factor may have a significant impact on technique effectiveness more often than the information base factor. The

importance of the mapping factor provides yet another contrast to traditional fault localization research, which has often focused solely on ways to bring bigger, better, and faster information bases to fault localization techniques.

BIBLIOGRAPHY

[1] R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, Rome, Italy, September 26–29, 2004 (to appear).

[2] H. Agrawal and J.R. Horgan. Dynamic program slicing. In *Proceedings of the ACM SIGPLAN '90 Conference on Programming Language Design and Implementation*, pages 246–256, June 1990.

[3] H. Agrawal, J.R. Horgan, S. London, and W.E. Wong. Fault localization using execution slices and dataflow tests. In *Proceedings of the IEEE Sixth International Symposium on Software Reliability Engineering*, pages 143–151, Toulouse, France, October 1995.

[4] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi. A type system for statically detecting spreadsheet errors. In *Proceedings of the* 18^{th} *IEEE International Conference on Automated Software Engineering*, pages 174–183, Montreal, Quebec, October 6–10, 2003.

[5] C. Allwood. Error detection processes in statistical problem solving. *Cognitive Science*, 8(4):413–437, 1984.

[6] ANSI/IEEE. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE, New York, 1983.

[7] T. Antoniu, P.A. Steckler, S. Krishnamurthi, E. Neuwirth, and M. Felleisen. Validating the unit correctness of spreadsheet programs. In *Proceedings of the* 26^{th} *International Conference on Software Engineering*, pages 439–448, Edinburgh, Scotland, May 23–28, 2004.

[8] J. Atwood, M. Burnett, R. Walpole, E. Wilcox, and S. Yang. Steering programs via time travel. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 4–11, Boulder, Colorado, September 3–6, 1996.

[9] Y. Ayalew and R. Mittermeir. Spreadsheet debugging. In *Proceedings of the European Spreadsheet Risks Interest Group*, Dublin, Ireland, July 24–25, 2003.

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

[11] B. Boehm and V.R. Basili. Software defect reduction Top 10 list. *Computer*, 34(1):135–137, January 2001.

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

[13] D. Brown, M. Burnett, G. Rothermel, H. Fujita, and F. Negoro. Generalizing WYSIWYT visual testing to screen transition languages. In *Proceedings of the IEEE Symposium on Human-Centric Computing Languages and Environments*, pages 203–210, Auckland, New Zealand, October 28–31, 2003.

[14] P. Bunus and P. Fritzson. Semi-automatic fault localization and behavior verification for physical system simulation models. In *Proceedings of the* 18^{th} *IEEE International Conference on Automated Software Engineering*, pages 253–258, Montreal, Quebec, October 6–10, 2003.

[15] M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of Functional Programming*, 11(2):155–206, March 2001.

[16] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace. End-user software engineering with assertions in the spreadsheet paradigm. In *Proceedings of the* 25^{th} *International Conference on Software Engineering*, pages 93–103, Portland, OR, May 3–10, 2003.

[17] M. Burnett, C. Cook, and G. Rothermel. End-user software engineering. In *Communications of the ACM*, September 2004 (to appear).

[18] M. Burnett and H. Gottfried. Graphical definitions: Expanding spreadsheet languages through direct manipulation and gestures. *ACM Transactions on Computer-Human Interaction*, 5(1):1–33, March 1998.

[19] M. Burnett, G. Rothermel, and C. Cook. An integrated software engineering approach for end-user programmers. In H. Lieberman, F. Paterno, and V. Wulf, editors, *End User Development*. Kluwer Academic Publishers, 2004 (to appear).

[20] M. Burnett, A. Sheretov, and G. Rothermel. Scaling up a 'What You See Is What You Test' methodology to spreadsheet grids. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 30–37, Tokyo, Japan, September 13–16, 1999.

[21] T.Y. Chen and Y.Y. Cheung. On program dicing. *Software Maintenance: Research and Practice*, 9(1):33–46, January–February 1997.

[22] C. Cook, M. Burnett, and D. Boom. A bug's eye view of immediate visual feedback in direct-manipulation programming systems. In *Proceedings of Empirical Studies of Programmers: Seventh Workshop*, pages 20–41, Alexandria, VA, October 1997.

[23] C. Corritore, B. Kracher, and S. Wiedenbeck. Trust in the online environment. In *HCI International*, volume 1, pages 1548–1552, New Orleans, LA, August 2001.

[24] R.A. DeMillo, H. Pan, and E.H. Spafford. Critical slicing for software fault localization. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 121–134, San Diego, CA, January 8–10, 1996.

[25] E. Duesterwald, R. Gupta, and M.L. Soffa. Rigorous data flow testing through output influences. In *Proceedings of the* 2^{nd} *Irvine Software Symposium*, pages 131–145, Irvine, CA, March 1992.

[26] M. Erwig and M. Burnett. Adding apples and oranges. In *Proceedings of the* 4^{th} *International Symposium on Practical Aspects of Declarative Languages*, pages 173–191, January 2002.

[27] M. Fisher, M. Cao, G. Rothermel, C.R. Cook, and M.M. Burnett. Automated test case generation for spreadsheets. In *Proceedings of the* 24^{th} *International Conference on Software Engineering*, pages 141–151, Orlando, Florida, May 19–25, 2002.

[28] M. Fisher II, D. Jin, G. Rothermel, and M. Burnett. Test reuse in the spreadsheet paradigm. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering*, pages 257–268, November 12–15, 2002.

[29] M. Francel and S. Rugaber. Fault localization using execution traces. In *Proceedings of the ACM* 30^{th} *Annual Southeast Regional Conference*, pages 69–76, Raleigh, North Carolina, 1992.

[30] 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.

[31] T. Igarashi, J.D. Mackinlay, B.-W. Chang, and P.T. Zellweger. Fluid visualization of spreadsheet structures. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 118–125, Halifax, Nova Scotia, September 1–4, 1998.

[32] J.A. Jones, M.J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of the* 24^{th} *International Conference on Software Engineering*, pages 467–477, Orlando, Florida, May 19–25, 2002.

[33] M. Karam and T. Smedley. A testing methodology for a dataflow based visual programming language. In *Proceedings of the IEEE Symposium on Human-Centric Computing Languages and Environments*, pages 280–287, Stresa, Italy, September 5–7, 2001.

[34] A.J. Ko and B.A. Myers. Development and evaluation of a model of programming errors. In *Proceedings of the IEEE Symposium on Human-Centric Computing Languages and Environments*, pages 7–14, Auckland, New Zealand, October 28–31, 2003.

[35] A.J. Ko and B.A. Myers. Designing the Whyline: A debugging interface for asking questions about program failures. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 151–158, Vienna, Austria, April 24–29, 2004.

[36] B. Korel and J. Laski. Dynamic slicing of computer programs. *Journal of Systems and Software*, 13(3):187–195, November 1990.

[37] V. Krishna, C. Cook, D. Keller, J. Cantrell, C. Wallace, M. Burnett, and G. Rothermel. Incorporating incremental validation and impact analysis into spreadsheet maintenance: An empirical study. In *Proceedings of the International Conference on Software Maintenance*, pages 72–81, Florence, Italy, November 2001.

[38] J. Laski and B. Korel. A data flow oriented program testing strategy. *IEEE Transactions on Software Engineering*, 9(3):347–354, May 1983.

[39] J.R. Lyle and M. Weiser. Automatic program bug location by program slicing. In *Proceedings of the* 2^{nd} *International Conference on Computers and Applications*, pages 877–883, 1987.

[40] R.C. Miller and B.A. Myers. Outlier finding: Focusing user attention on possible errors. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 81–90, November 2001.

[41] S.C. Ntafos. On required element testing. *IEEE Transactions on Software Engineering*, 10(6):1984, November 1984.

[42] H. Pan and E. Spafford. Toward automatic localization of software faults. In *Proceedings of the* 10^{th} *Pacific Northwest Software Quality Conference*, October 1992.

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

[44] R. Panko. What we know about spreadsheet errors. *Journal on End User Computing*, pages 15–21, Spring 1998.

[45] 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. In *Proceedings of the IEEE Symposium on Human-Centric Computing Languages and Environments*, pages 15–22, Auckland, New Zealand, October 28–31, 2003.

[46] B. Pytlik, M. Renieris, S. Krishnamurthi, and S.P. Reiss. Automated fault localization using potential invariants. In *Proceedings of the* 5^{th} *International Workshop on Automated and Algorithmic Debugging*, pages 273–276, Ghent, Belgium, September 8–10, 2003.

[47] S. Rapps and E.J. Weyuker. Selected software test data using data flow information. *IEEE Transactions on Software Engineering*, 11(4):367–375, April 1985.

[48] O. Raz, P. Koopman, and M. Shaw. Semantic anomaly detection on online data sources. In *Proceedings of the* 24^{th} *International Conference on Software Engineering*, pages 302–312, Orlando, FL, May 19–25, 2002.

[49] J. Reichwein and M.M. Burnett. An integrated methodology for spreadsheet testing and debugging. Technical Report 04-60-02, Oregon State University, Corvallis, OR, January 2004. http://eecs.oregonstate.edu/library/?call=2004-6. Last accessed: July 19^{th}, 2004.

[50] J. Reichwein, G. Rothermel, and M. Burnett. Slicing spreadsheets: An integrated methodology for spreadsheet testing and debugging. In *Proceedings of the* 2^{nd} *Conference on Domain Specific Languages*, pages 25–38, Austin, Texas, October 3–5, 1999.

[51] M. Renieris and S.P. Reiss. Fault localization with nearest neighbor queries. In *Proceedings of the* 18^{th} *IEEE International Conference on Automated Software Engineering*, pages 30–39, Montreal, Canada, October 6–10, 2003.

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

[53] G. Rothermel, M. Burnett, L. Li, C. Dupuis, and A. Sheretov. A methodology for testing spreadsheets. *ACM Transactions on Software Engineering and Methodology*, 10(1):110–147, January 2001.

[54] G. Rothermel, L. Li, C. Dupuis, and M. Burnett. What You See Is What You Test: A methodology for testing form-based visual programs. In *Proceedings of the* 20^{th} *International Conference on Software Engineering*, pages 198–207, Kyoto, Japan, April 19–25, 1998.

[55] K.J. Rothermel, C.R. Cook, M.M. Burnett, J. Schonfeld, T.R.G. Green, and G. Rothermel. WYSIWYT testing in the spreadsheet paradigm: An empirical evaluation. In *Proceedings of the* 22^{nd} *International Conference on Software Engineering*, pages 230–239, Limerick, Ireland, June 4–11, 2000.

[56] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Prabhakararao, M. Fisher II, and M. Main. End-user software visualizations for fault localization. In *Proceedings of the ACM Symposium on Software Visualization*, pages 123–132, San Diego, CA, June 11–13, 2003.

[57] J.R. Ruthruff, A. Phalgune, L. Beckwith, M. Burnett, and C. Cook. Rewarding "good" behavior: End-user debugging and rewards. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, Rome, Italy, September 26–29, 2004 (to appear).

[58] J.R. Ruthruff, S. Prabhakararao, J. Reichwein, C. Cook, E. Creswick, and M. Burnett. Interactive, visual fault localization support for end-user programmers. *Journal of Visual Languages and Computing*, 2004 (to appear).

[59] J. Sajaniemi. Modeling spreadsheet audit: A rigorous approach to automatic visualization. *Journal on Visual Languages and Computing*, 11(1):49–82, February 2000.

[60] F. Tip. A survey of program slicing techniques. *Journal on Programming Languages*, 3(3):121–189, 1995.

[61] J.M. Voas. Software testability measurement for assertion placement and fault localization. In *Proceedings of the International Workshop on Automated and Algorithmic Debugging*, pages 133–144, 1995.

[62] E. Wagner and H. Lieberman. An end-user tool for e-commerce debugging. In *Proceedings of the International Conference on Intelligent User Interfaces*, page 331, Miami, Florida, January 12–15, 2003.

[63] E.J. Wagner and H. Lieberman. Supporting user hypotheses in problem diagnosis on the web and elsewhere. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 30–37, Funchal, Madeira Island, January 13–16, 2004.

[64] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, July 1984.

[65] A. Wilson, M. Burnett, L. Beckwith, O. Granatir, L. Casburn, C. Cook, M. Durham, and G. Rothermel. Harnessing curiosity to increase correctness in end-user programming. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 305–312, Fort Lauderdale, FL, April 5–10, 2003.

[66] C. Wohlin, P. Runeson, M. Host, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering*. Kluwer Academic Publishers, Boston, MA, 2000.

APPENDICES

APPENDIX A

TUTORIAL FOR EXPERIMENT #1

We are conducting this research study so we can learn about how people test and find errors in spreadsheets.

The people involved in this study are Professors Burnett, Cook, and Rothermel, and students Dan Keller, and Josh Cantrell.

For today's experiment, you will be learning about testing spreadsheets in the Forms/3 environment. I'll lead you through a brief tutorial, and then you will have a few experimental tasks to work on.

Participation in this study is completely voluntary. If you choose to continue in this study, you will receive 5 points Extra Credit or $10.00.

There are no risks associated with this study. The benefits are the reward you choose and helping us to shape a future direction in computer science. There will be no record tying you to your work performed during the study. Your data will be assigned a random number, and no record will tie you to that number after the data has been collected. You may choose to withdraw at any time.

Please don't discuss the experiment with anyone. We are doing sessions all week and would prefer the students coming later not have any advance knowledge.

Any questions about this research study and its procedures may be directed to any of the following people involved with this study.

- Dan Keller — keller@cs.orst.edu
- Josh Cantrell — cantrjos@cs.orst.edu
- Dr. Burnett — burnett@cs.orst.edu
- Dr. Cook — cook@cs.orst.edu

Any other questions can be directed to the IRB Coordinator at 737-3437 or irb@orst.edu.

As we go through this tutorial, I want you to actually DO the steps I'm describing. When I say "click", I will always mean click the left mouse button once unless I specify otherwise. Pay attention to your computer while you do the steps, and look at my slide to confirm you result.

You have a "Forms/3 Quick Reference Card" that you may refer to during the study. Feel free to add notes to it during this tutorial. If you have a question, raise your hand and my assistant will help you.

For each spreadsheet that we will be looking at, you will have a sheet of paper describing what the spreadsheet is supposed to do. Read the description of the PurchaseBudget spreadsheet now.

Now open the PurchaseBudget spreadsheet by clicking on the button labeled PurchaseBudget at the bottom of the screen.

There are some obvious differences between Forms/3 and other spreadsheets. For example, Forms/3 does not use a fixed grid of cells (PAUSE) and we can give the cells useful names like Pens and TotalCost (point to the cells on the spreadsheet). Just like other spreadsheets, you see a value associated with each cell.

You can select a cell by clicking on it. Click on the Pens cell (PAUSE).

The Pens cell now has a selection outline indicating you have selected it. You can move the cell by holding the mouse button down over the cell and dragging. You can also resize the cell by dragging one of the black boxes on the selection outline. Try moving and resizing the Pens cell.

You can look at the formula for a cell by clicking on the down arrow button. Click on the down arrow for PenTotalCost.

It calculates the total cost of the pens being ordered by multiplying the number of pens added by the cost of each pen. You can display more than one formula at a time. Click on the down arrow for Pens to look at its formula.

Pens is an input cell, so its formula is simply a value. During this study, you will be able to change the value in input cells but you cannot change the formulas in output cells. Change the value of Pens to 70 by clicking in the formula window, deleting the old value and entering the new value, 70. Press apply to save your changes. When you press apply, you should see the value of Pens and some other cells change to reflect the new value. Hide the formulas for Pens and PenTotalCost by clicking on the hide button in the formula windows.

Earlier I said that you would be testing some spreadsheets and looking for errors. What this means is that you will try some different combinations of inputs, look at the corresponding output values, and tell the system if the value for output cells are correct or incorrect.

You have probably noticed by now that output cells have colored borders and decision boxes in the upper right corner. These tools are to help you with your testing. The colored borders indicate how well you have tested a cell. A red border means completely untested, a blue border means fully tested and a purple border

indicates that a cell is only partially tested. The “bluer” the purple, the more tested the cell is.

The decision box is used to communicate your testing decisions to the system. If you look at the decision box for all the output cells, they have a question mark in them. A question mark indicates that you haven’t made a decision for inputs like these. You make a testing decision by looking the value displayed in an output cell and deciding whether it is right or wrong given the current input values. If the output is correct for the given inputs, you tell the system it is correct by clicking on the decision box for the cell.

Let’s do an example of recording a decision. Look at the value in the PenTotal-Cost cell. Does the value in PenTotalCost seem to be correct? You can check the description of the spreadsheet to help you determine if the cell is calculating the correct value. The description says that PenTotalCost should calculate the total cost of the pens you are going to buy.

Since you are ordering 70 pens and the cost the per pen is 3, the PenTotal-Cost should be 210. The value looks correct, so tell the system the value is correct by clicking on the decision box. Remember, the computer does not know if your decision is right or wrong, it just records your decision.

You should have noticed that three things happened:

1. The decision box displays a checkmark, showing that your decision was recorded. A checkmark means that the system has recorded that you decided the current value is correct.

2. The border color for PenTotalCost turned blue. Remember a blue border means the cell is tested.

3. The Percent Tested indicator located in the top margin changed. The percent indicator lets you know how much of the entire spreadsheet you have tested. The color of the percent tested indicator follows the same convention as cell borders. Red means untested, blue means fully tested and shades of purple between red and blue indicate various degrees of partial testedness. As you test more cells, the percent tested will increase. When all the cells are fully tested, the percent tested indicator will be 100%

If you make a mistake and clicked on a wrong decision box, or if you didn't notice the changes in the colors or Percent Tested indicator, you can undo the decision you just made. Now, click on the PenTotalCost decision box again to remove your decision.

Notice that the checkmark changed back to a question mark, the border turned back to red and the percent tested indicator went back to 0.

Now click on the decision checkbox in the PenTotalCost cell again.

Since the rest of the cells still have question marks, let's make decisions for them too.

Look at the value of the PaperTotalCost cell. PaperTotalCost calculates the total cost of the paper on order. Based on the current inputs, does the value in PaperTotalCost seem to be correct?

You are ordering 400 units of paper and the cost per unit is 4. So 1600 seems to be correct. Tell the system that this value of PaperTotalCost is correct.

Look at the value for PenQCheck. This cell lets you know if you are ordering enough pens. Does this value seem to be correct?

The minimum number of pens you need is 68. You will have 70 pens after the order, so the value for the cell seems to be correct. Tell the system this value of PenQCheck is correct.

Look at the value for PaperQCheck. This cell lets you know if you are ordering enough paper. Does this value seem to be correct?

The minimum amount of paper you need is 400. You are ordering 400, so that should be enough. However, the PaperQCheck cell says that you do not have enough paper. That means this cell has an error in it. When you find a cell with a wrong value you need to tell the system the value is wrong. To tell the system the value is wrong, right click on the decision box for the cell. Go ahead and tell the system this value for PaperQCheck is wrong.

When you tell the system a value is wrong, an X appears in the decision box.

If you tell the system a value is wrong and you shouldn't have, you can undo your decision by right clicking on the decision box again. Right click on PaperQCheck to undo the decision. Notice the X turns back into a question mark. Tell the system that the value for PaperQCheck is wrong again.

Usually when you find an error in spreadsheet, you would want it fix it. However, for this study it is sufficient to indicate that the value is wrong. If you find a wrong value, you should indicate that it is wrong and continue with your testing. There may be more than one wrong value for a cell. Also, if you find a wrong value, you may not be able to get up to 100% testedness.

Now look at the value for the TotalCost cell. This cell calculates the total cost of the units on order. Does this value seem to be correct?

You are spending 210 on pens and 1600 on paper, so 1810 seems to be correct. Tell the system this value is correct.

Look at the value for the cell BudgetOK?. This cell checks to see that the total cost is within the alloted budget. Does this value seem to be correct?

Your budget is 2000 and the total cost is 1810, so it seems that your budget is okay. Tell the system this value is correct.

Now we have tested every cell. However, some cells are purple and the percent tested indicator is only at 50%.

Remember that I said that when the cell is purple it is only partially tested. That means there are some more situations that we haven't tried yet. Let's change an input value to look at another situation. Change the value of Pens to 0.

Notice that the decision box for PenQCheck is displaying a question mark, indicating that this is a new situation for that cell. Also notice that the decision boxes for PenTotalCost, TotalCost, and BudgetOK? turned into a blank. A blank for a cell's decision box means that you have already tested a situation like this before and telling the system about this value won't increase the testedness of the spreadsheet.

Let's look at the value for PenQCheck, since we haven't made a decision for this situation yet. PenQCheck says that we don't have enough pens. Does this seem to be correct?

The minimum number of pens we need is 68 and we are ordering 0. This is not enough pens, so the value seems to be correct. Tell the system this value is correct.

Now let's look at the PaperQCheck cell. You can look at the formula to help figure out what part hasn't been tested. Open the formula to see what part of PaperQCheck hasn't been tested.

The value of the cell is "not enough paper", so we must be in the "then" part of the formula. To get to "else" part, Paper must be greater than 400.

Let's change the value of Paper, so there will be enough paper. Open the formula for Paper and change the value to 410. Remember to press the apply button. Hide the formula when you are done.

Notice that the decision box for PaperQCheck is displaying a question mark, meaning that you haven't tested a situation like this one. Look at the value in PaperQCheck. Does it seem to be correct? There is enough paper now, so the value is correct. Tell the system the value is correct for PaperQCheck.

TotalCost and BudgetOK? are still purple, which means that they are both only partially tested.

To help you decide which parts haven't been tested, Forms/3 provides arrows so you can see the relationships between cells. You can turn on the arrows for a cell by right clicking on the cell. Right click on the BudgetOK? cell.

You can see arrows from Pens, Paper, and TotalCost pointing to the BudgetOK? cell, because BudgetOK? uses all those values in its calculations.

The arrows follow the same color scheme as the cell borders. A red arrow indicates an untested relationship, a blue arrow indicates a fully tested relationship, and a purple arrow indicates a partially tested situation. Opening the formula for a cell allows you to see some more detail with the arrows. Open the formula for BudgetOK?.

Look at the arrow pointing from TotalCost to BudgetOK? It is purple, so that means that the relationship between the two cells is only partly tested. Notice also that the arrow is pointing to the if statement where TotalCost is used. BudgetOK? says that the budget is currently okay, so we are in the else part of the if statement. To go to the then part of the if, the TotalCost must be greater than 2000. We need to change the inputs to get to that new situation. Turn off arrows for BudgetOK? by

right clicking on it again. Hide the formula for BudgetOK? by clicking on the hide button.

We need to have the TotalCost be greater than 2000. Let's change the number of Pens to 1000.

Notice that there is a question mark in the decision box for BudgetOK?, telling us that this is a new situation. Does the value of BudgetOK? seem to be correct? The TotalCost is 4,640 which is more than 2000 so "over budget" seems to be correct. Tell the system this value is correct.

TotalCost is still purple, so let's see if we can figure out what part hasn't been tested yet. Open the formula for TotalCost. TotalCost is displaying the sum of PenTotalCost and PaperTotalCost, so it is in the else part. To get to the then part of the if, Pens or Paper must be less than 0. Let's change Paper to -1.

Now look at the value for TotalCost. Does it seem to be correct? It does, so tell the system this value is correct.

Notice that PaperQCheck also has a question mark in its decision box. Does the value for PaperQCheck seem to be correct? It does, so tell the system it is correct.

Notice that BudgetOK? also has a question mark in its decision box. Does the value for BudgetOK? seem to be correct? It does, so tell the system the value is correct.

BudgetOK? is still purple so that means there is at least one more untested situation for it. Open the formulas for TotalCost and BudgetOK? by clicking on their down arrows. Now turn on arrows for BudgetOK? by right clicking on it.

There are two arrows from TotalCost to BudgetOK?: one is red and one is blue. The blue arrow indicates a relationship that has been tested. The red arrow indicates a relationship that has not been tested.

The red arrow starts at the “then” part of TotalCost. That means that Pens or Paper must be less than zero. The red arrow ends at the first else part of BudgetOK? To get to the first else part of BudgetOK?, Pens and Paper must both be greater than or equal to zero. Therefore the red arrow represents a relationship where Pens and Paper must be less than zero and greater than or equal to zero at the same time. This is impossible and gives us an impossible situation that can’t be tested.

On your experimental tasks there may or may not be impossible situations, so do your best to test as many situations as you can. Remember to tell the system whether the value is correct or incorrect for the output cells in each situation.

APPENDIX B

MATERIALS FOR EXPERIMENT #1

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.

Input Cells

Pens	The number of boxes of pens you ordered.
Paper	The number of reams of paper you ordered.

Output Cells

PenTotalCost	The total cost of ordering the boxes of pens.
PaperTotalCost	The total cost of ordering the reams of paper.
PenQCheck	Tells you if you're not ordering enough pens.
PaperQCheck	Tells you if you're not ordering enough paper.
TotalCost	The total cost of your order.
BudgetOK?	Tells you if you spent more than your budget, or provided invalid quantities for Pens or Paper.

FIGURE B.1: The task description of the PurchaseBudget spreadsheet program from Chapters 4 and 6. This program was used in the tutorials for Experiment #1 and Experiment #2.

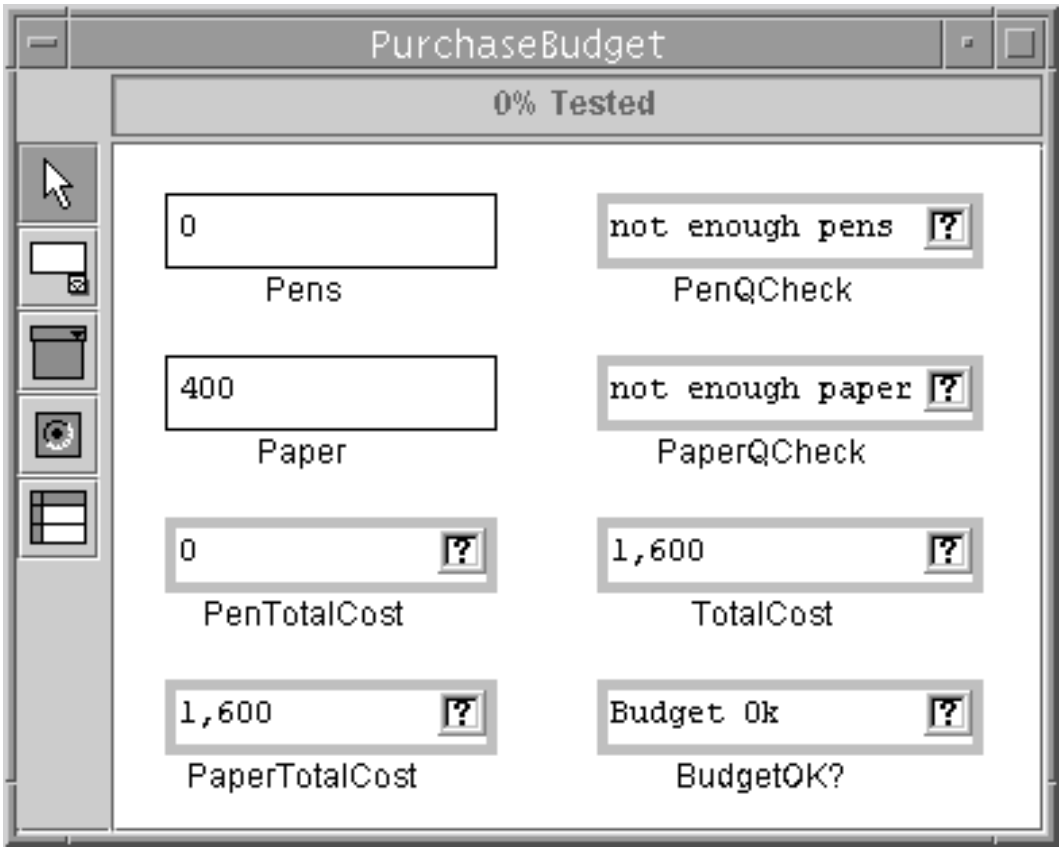

FIGURE B.2: The PurchaseBudget spreadsheet program used in the tutorial of Experiment #1 and Experiment #2 in Chapters 4 and 6.

Change

Suppose you have a large jar of pennies that you want to take to the bank to be counted. You want to know how many dollars, quarters, dimes, nickels, and pennies that the bank will give you for your jar of pennies.

This spreadsheet calculates the **minimum** number of dollars, quarters, dimes, nickels, and pennies that are of equal value to your jar of pennies. For example, the bank will not give you two nickels, since they could give you one dime instead.

Input Cell

jar_of_pennies	The number of pennies in your jar.

Output Cell

Dollars	The number of dollars you'll get.
Quarters	The number of quarters you'll get.
Dimes	The number of dimes you'll get.
Nickels	The number of nickels you'll get.
Pennies	The number of pennies you'll get.

Task

You are to thoroughly test the Forms/3 spreadsheet and mark any errors you find by placing an "X" in the decision box of the cell with the error. Do not attempt to correct any errors you find.

FIGURE B.3: The task description of the Change spreadsheet program from Chapter 4.

Grade

The grade for a class you are taking is based on four quizzes and one extra-credit assignment. You can determine what your grade will be in the class once you know your scores for the quizzes and the extra-credit assignment. If any score on a quiz is greater than 100, then the spreadsheet will report an error.

The grade is calculated in the Forms/3 spreadsheet as follows:

1.The quiz scores are averaged together.
2.Extra credit is computed as follows:
    5 points of extra credit for a score greater than 25 on the extra-credit assignment.
    3 points of extra credit for a score of 20 to 25 on the extra-credit assignment.
    0 points of extra credit for scores less than 20 on the extra-credit assignment.
3.The total score is the sum of the quiz average and the extra credit points.
4.The letter grade is computed from the total score.

90 - 100 =	A
80 - 89 =	B
70 - 79 =	C
60 - 69 =	D
0 - 59 =	F

Input Cells

quiz1	Your score for quiz one.
quiz2	Your score for quiz two.
quiz3	Your score for quiz three.
quiz4	Your score for quiz four.
ExtraCredit	Your score on the extra-credit assignment

Output Cells

Avg	Averages of your four quiz scores.
EC_Award	The number of extra-credit points you will get: 5, 3, or 0.
ErrorsExist?	Lets you know if you accidentally entered a quiz score of greater than 100
TotalScore	The total score you are getting for the class; is the sum of Avg and EC_Award.
LetterGrade	The letter grade you will be getting for the class.

Task

You are to thoroughly test the Forms/3 spreadsheet and mark any errors you find by placing an "X" in the decision box of the cell with the error. Do not attempt to correct any errors you find.

FIGURE B.4: The task description of the Grade spreadsheet program from Chapter 4.

APPENDIX C

TUTORIAL FOR EXPERIMENT #2

Hi, my name is Joseph R. Ruthruff, and I will be leading you through today's study.

The other people involved in this study are Dr. Margaret Burnett, Dr. Gregg Rothermel, 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. Gregg Rothermel grother@cs.orst.edu

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

Before we begin, I'd like to ask if anyone in here is color blind. 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 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 first page of the description of the PurchaseBudget spreadsheet now.

(Read aloud with them)

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 dells 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. Consequently, we may not be confident that the cell's value is correct. 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 examine the TotalCost cell. 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 pop-up 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.

We can see that the value of this TotalCost cell is zero. Is this value 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 sum of PenTotalCost and Paper-TotalCost." Well, both PenTotalCost and PaperTotalCost are zero. Since zero plus zero is zero, TotalCost appears to have the correct value.

The previous pop-up message told us to left-click if the cell's value is correct. So let's left-click this decision box for TotalCost—again, that's the box in the Total-Cost cell with the question mark. 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 with another click in the same decision box. Try it by left-clicking on that checkmark in TotalCost's decision box. (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 cell's border in the same way as before. What does the message say? It 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 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, okay, 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 PaperTotal-Cost 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.

Let's work on getting another cell fully tested. Look at the value of the Paper-QCheck cell. Is this value correct? Let's read the second paragraph at the top of the spreadsheet description. (read it.) Well, with a value of zero in the paper cell, and a value of 21 in the PaperOnHand cell, we have 21 sheets of paper, which is not enough to fill our shelves. Since the PaperQCheck cell says "not enough paper", its value is correct. Now how do I indicate that a cell's value is correct? I forgot. Let's rest our mouse over the question mark on PaperQCheck's decision box. Oh yeah, it says "Left-click if the cell's value is correct, and right-click if the cell's value is incorrect. These decisions help test and find errors." 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 pop-up message. So let's left-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. Can anyone read what the pop-up message says? (wait for hand) That's right, 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.

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 left-clicking on this arrow button.

Hey, a formula box popped up. We can change the value of this box to change the value of the Paper cell. Let's try changing the value to 50. Change the number 0 in this formula box to a 50 and hit the Apply button.

Now look at the decision box of the PaperQCell. It is blank. I don't remember what that means, so let's rest my mouse over the decision box of this PaperQCell. 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 big value. Move your mouse back to the formula box for the Paper cell, change its value to 500, and left-click the Apply button. Now push the Hide button on this formula box.

Now look at the PaperQCell. 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 500 in the Paper cell and 21 in the PaperOnHand cell, I have 521 paper on stock. Is this enough paper? I'm going to go back and read my spreadsheet description to find out. (read second paragraph aloud again) So I only need 400 reams of paper, but I have 521. So this is enough. And the PaperQCell says "paper quantity ok". Well, this is correct, so let's left-click on the PaperQCheck cell's decision box. All right! The border changed to blue, and even more, the spreadsheet is now 56% tested.

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. Pens is already zero, so we don’t have to worry about that input cell. Wait, Paper is 400 in this example spreadsheet, but my value is still 500. So let’s change the value of this Paper to 400 so that it matches the example spreadsheet in my description. How do I do this? Left-click on the arrow button at the bottom of the Paper cell, and change the 500 to a 400, and push the Apply button. I think I’m done with this formula, so let’s hide it by left-clicking on the “Hide” button. Moving on, in this example correct spreadsheet, PensOnHand is 25, and PaperOnHand is 21. Oh good, my spreadsheet already has these values, so I don’t have to change anything.

Now, according to this example spreadsheet in our description, 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?

Move your mouse to the decision box with the question mark in it and hold it there until a message pops up. What does it say? The message tells us that if the cell’s value is correct to go ahead and left-click and if it is wrong to right-click. Well, this value is wrong, so go ahead and right-click on this decision box.

Hey, wait a second, 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 the correctness of the TotalCost cell. For the current set of inputs, the Total cost should be 1600. But our total cost cell says

2800. That means the value associated with the TotalCost cell is “Wrong”. What can we do? Well, TotalCost does have a question mark in the decision box, so since the value is wrong, we can right-click in the decision box to place an X-mark. Let’s do that now. Take a few seconds to explore anything that might have changed by moving your mouse over the items and viewing the pop-up 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.

Okay, so we have three cells with wrong values. There is at least one bug in a formula somewhere on this spreadsheet that is causing these three cells to have incorrect values. I’m going to give you a couple minutes to look for this bug and fix it. To do this, find the cells that you think might have bugs in their formulas. You can then open the formula of these suspect cells and check to see if the formula matches what your spreadsheet description says it should do. If you find something wrong with the formula, fix it.

So take a couple minutes, and try to find this bug.

(PAUSE for 2 minutes)

Okay, let’s make sure that everyone found the bug by going over it together. I’m going to start by looking in the PaperTotalCost cell. So let’s open its formula. Hmmm, okay, so it is taking the value of the Paper cell and multiplying it by 7. But wait, 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.) Ah ha, the cost of paper is four dollars, but this cell is using a cost of seven. This is wrong. So I’m going to change the 7 in this formula to a 4, and left-click the Apply button to finalize my changes.

Hey wait, my total spreadsheet testedness at the top of my window went down to 28%! What happened? Well, since I corrected the formula, Forms/3 had to discard some of my 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 35%; I got some of my testedness back.

Okay, now I'm going to give you a few more minutes to explore the rest of this spreadsheet. Look at the bottom of the description. It says, "Test the spreadsheet to see if it works correctly, and correct any errors you find." Why don't you go ahead and do that.

Remember, if you are curious about any aspect of the system, you can hover your mouse over the item and read the pop-up. Also, you might find those check-marks and X-marks to be useful. You can place checkmarks in the decision box of correct cells by left-clicking, and X-marks in the decision box of incorrect cells by right-clicking.

Go ahead now and try to finish this task. I'll give you a few minutes.

APPENDIX D

MATERIALS FOR EXPERIMENT #2

Background Questionnaire

1. Gender (circle your selection): Male / Female
2. Major or Educational Background: _____
3. Year or Degree Completed: Fresh. Soph. Jun. Sen. Post Bac. Grad.
4. Cumulative GPA: _____
5. 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? _____
6. 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? _____
7. Have you participated in any previous Forms/3 experiments? Yes / No
8. Is English your primary language? Yes / No

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

FIGURE D.1: The background questionnaire of the experiment described in Chapter 6.

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 lowest quiz score 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 99 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 lowest midterm score is dropped. The average midterm score is then the average of the two highest 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 Percentage

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 percentage, in accordance with the following scale:

94% and up: A	82% - 86% : B	72% - 76% : C	62% - 66% : D
89% - 94% : A-	79% - 82% : B-	69% - 72% : C-	59% - 62% : D-
86% - 89% : B+	76% - 79% : C+	66% - 69% : D+	Below 59% : F

FIGURE D.2: The first page of the task description of the Gradebook spreadsheet program from Chapter 6.

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)	90
Midterm2	96
Midterm3 (Original)	80
Midterm_Average	93.45
Final	129
Final_Percentage	88.36
Course_Percentage	94.65
Course_Grade	A

Mary Smith	**Report Card**
Quiz1	0
Quiz2	88.24
Quiz3	85
Quiz4	87
Quiz5	100
Quiz_Average	90.06
Midterm1 (Original)	48
Midterm2	61
Midterm3 (Original)	66
Midterm_Average	64.5
Final	106
Final_Percentage	72.6
Course_Percentage	76.34
Course_Grade	C+

FIGURE D.3: The second page of the task description of the Gradebook spreadsheet program from Chapter 6. This page contains two example, correct "report cards" that participants were allowed to use to facilitate testing of the spreadsheet task.

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.

FIGURE D.4: The first page of the task description of the Payroll spreadsheet program from Chapter 6.

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	

FIGURE D.5: The second page of the task description of the Payroll spreadsheet program from Chapter 6. This page contains two example, correct “payroll stubs” that participants were allowed to use to facilitate testing of the spreadsheet task.