Interactive Fault Localization Techniques in a Spreadsheet Environment

Joseph R. Ruthruff, *Student Member*, *IEEE*, Margaret Burnett, *Senior Member*, *IEEE*, and Gregg Rothermel, *Member*, *IEEE*

Abstract—End-user programmers develop more software than any other group of programmers, using software authoring devices such as multimedia simulation builders, 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 paper investigates fault localization techniques in the spreadsheet domain, the most common type of end-user programming environment. We investigate a technique previously described in the research literature and two new techniques. We present the results of an empirical study to examine the impact of two individual 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.

Index Terms—Fault localization, debugging, end-user software engineering, spreadsheets, end-user programming.

1 Introduction

Not long ago, most software was developed 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, who often have no reason to be interested in or understand such theory and practice, write far more software than professional programmers. In fact, Scaffidi et al. [59] estimate that there were 80 million end-user programmers in American workplaces in 2005 and that this number will rise to 90 million by 2012, as compared to fewer than 3 million professional programmers [59].[1] The software created by these end users is highly diverse, including multimedia simulations, dynamic web pages, e-mail filtering rules, and spreadsheets.

Unfortunately, there is reason to believe that end users do not have adequate programming support. For example, Boehm and Basili [9] observe that 40-50 percent of the software created by end users contains nontrivial faults. There is research [8], [42], [43] revealing that spreadsheets, the most common type of software developed by end users, often contain faults. These faults can be serious, costing millions of dollars in some cases (e.g., [29], [42], [50]). Perhaps even more disturbing, spreadsheet developers often express unwarranted confidence in the quality of these programs [20], [43].

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

Software engineering researchers have long recognized the importance of fault localization strategies and have invested considerable effort into bringing fault localization techniques to professional programmers (e.g., [3], [11], [19], [22], [28], [31], [38], [46], [49], [63]) and similar efforts, directed at the needs of end users, could be worthwhile. However, significant differences exist between professional and end-user software development [54] and these differences have ramifications for fault localization techniques by acting as constraints on the types of strategies suitable for end users. Four differences of particular interest to this work are:

1. the fact that end users rarely have knowledge of software engineering theory and practices, which can be necessary to use fault localization techniques or understand their feedback,
2. the modeless and interactive nature of most end-user programming environments, which prevents a batch processing of information by techniques before displaying fault localization feedback,

1. Scaffidi et al. [59] identify specific "end-user subpopulations" such as "Managerial and Professional," "Technical, Sales, Administration," "Service," and so on. They derived their numbers using recent Bureau of Labor Statistics data and projections.

- *J.R. Ruthruff and G. Rothermel are with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588. E-mail: {ruthruff, grother}@cse.unl.edu.*
- *M. Burnett is with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331. E-mail: burnett@eecs.orst.edu.*

Manuscript received 19 July 2005; revised 29 Nov. 2005; accepted 30 Jan. 2006; published online 27 Apr. 2006.
Recommended for acceptance by J. Offutt.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0201-0705.

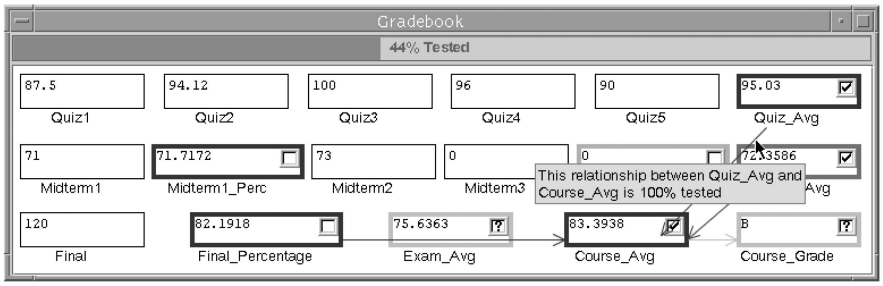

Fig. 1. An example of WYSIWYT in the Forms/3 spreadsheet environment.

3. a lack of organized testing infrastructures in end-user software development, thereby limiting the usefulness of techniques that require large bases of information to operate effectively, and
4. the unreliability of information used by techniques, which has been observed in previous studies involving end users [33], [34], [57], but has rarely been considered in fault localization techniques designed for professional programmers.

To date, little research has sought to bring fault localization strategies specifically to end-user programmers, especially in the context of these four differences. In prior work [57], we presented a fault localization technique for spreadsheet users that considers these differences. However, this technique maintains a large base of information with which to provide fault localization feedback and, thus, the costs of this technique may be too great as spreadsheet size increases. We therefore present algorithms for two new techniques for spreadsheet users that were designed with these cost concerns in mind.

Our preliminary, formative investigations [53], [57] into the effectiveness of this fault localization support show that it can help pinpoint the location of faults and can help end users find faults by leading them to employ more effective debugging strategies. However, our results also show that, at times, this support can be ineffective. Analysis of the cases in which the support was not effective suggests that there are actually *two distinct factors* involved in the design of a fault localization technique that could impact the technique's performance.

We have therefore designed and conducted a controlled experiment analyzing the role of these two factors in fault localization techniques. The first factor—information base—refers to the type of information stored and maintained by a technique in order to provide feedback and is commonly the subject of the research literature on fault localization. The second factor—mapping—refers to the way in which a technique maps the information into feedback, including both the calculation of this feedback using the information base and displaying the feedback in a manner that is compatible with the surrounding environment. Compared to the information base factor, this mapping factor has received scant attention in the research literature. Further, in our search of the literature, we find no previous work that has separated the impact of these two factors on a fault localization technique's effectiveness.

This work makes five primary contributions. The first contribution is a presentation of three unique fault localization techniques for spreadsheet users—two of which are presented for the first time in detail in this paper—that are designed to accommodate the previously mentioned differences between professional and end-user programming. The second contribution is a decomposition of fault localization techniques into two separate factors—information base and mapping—for the purposes of both their design and their evaluation. The third contribution is an experiment investigating the impact of both the information base and mapping factors on the effectiveness of fault localization techniques. Our results indicate that each factor significantly impacts technique effectiveness; thus, both must be considered when developing fault localization techniques. Furthermore, our experiment's design sheds insight into how interactive fault localization techniques should be evaluated in general. The fourth contribution involves adding to the end-user software engineering literature by providing empirical data on three fault localization information bases and three mappings that can be used in fault localization techniques for spreadsheet users. The fifth contribution is a corroboration of the growing body of evidence that end users make mistakes when performing interactive testing and debugging tasks, which has implications for the types of techniques that may be suitable for end-user programmers.

The remainder of this paper is organized as follows: Section 2 describes the end-user software engineering devices with which our fault localization techniques are integrated and discusses other related work; Section 3 presents three fault localization techniques for spreadsheet environments with algorithms and complexity analyses; Section 4 decomposes our techniques into the two factors of information base and mapping and discusses the role of these two factors in fault localization techniques in general; Section 5 describes the experiment to investigate the impact of these two factors; and Section 6 concludes.

2 Background

2.1 The WYSIWYT Testing Methodology

Our end-user software engineering research [14] aims to provide software development support to end-user programmers. One component in this strategy is the "What You See Is What You Test" (WYSIWYT) testing methodology [15], [51]. Our fault localization techniques are prototyped in the spreadsheet paradigm, in conjunction with the WYSIWYT testing methodology, so we briefly describe that methodology here.

Fig. 1 presents an example of WYSIWYT in Forms/3 [12], a spreadsheet language utilizing "free-floating" cells in

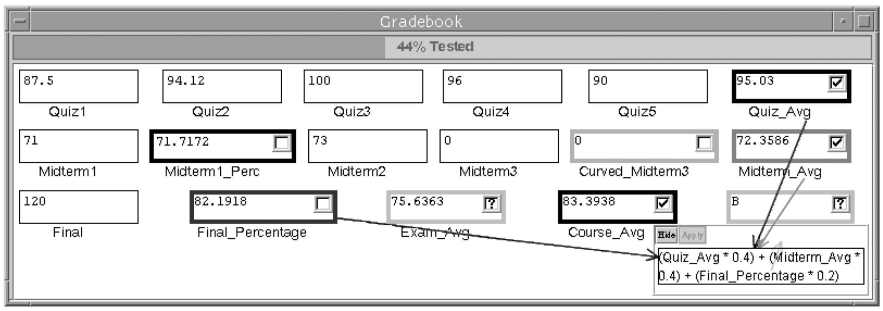

Fig. 2. An example of dataflow arrows at the subexpression level.

addition to traditional spreadsheet grids. Forms/3 is a research prototype that captures the essence of the spreadsheet paradigm. It does not have all the features of commercial systems such as Excel, but those systems do not have testing and debugging support to the extent available in Forms/3. Instead, Excel has heuristics that warn of some types of errors. Both systems also have arrows that point out dataflow. (We describe WYSIWYT's dataflow arrows shortly.)

WYSIWYT's underlying assumption is that, as a user incrementally develops a spreadsheet, he or she is also testing incrementally. Because the intended audience is end users, all communication is performed through visual devices. In WYSIWYT, untested cells that have nonconstant formulas are given a red border (light gray in this paper). (It is assumed that cells whose formulas are simply constants do not need to be tested.) For example, in Fig. 1, 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."

Tests can occur at *any* time—intermingled with formula edits, formula additions, 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* completes a successful "test." Such checkmarks can increase the "testedness" of cells, which is reflected by adding more blue to cell borders (more black in this paper). For example, in Fig. 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).[2]

WYSIWYT testedness colors reflect the use of a dataflow test adequacy criterion [37], [40], [47], which we term *du-adequacy* and define as follows: 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 pair 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* [23], is a test suite that exercises each du-pair in such a manner that it participates (dynamically) in the production of an output explicitly validated by the user.

In addition to providing feedback at the cell level, WYSIWYT provides users with feedback about testedness at two other granularities. A "percent testedness" indicator provides testedness feedback at the spreadsheet granularity by showing 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 spreadsheet increases; in Fig. 1, the spreadsheet is 44 percent tested. Testedness feedback is also available at a finer granularity through the colors of dataflow arrows, which can be shown at the cell level (in Fig. 1, the user has triggered dataflow arrows for the `Course_Avg` cell) or at the subexpression level (shown in Fig. 2). The system also provides testedness feedback through an intelligent explanation system [67], implemented via "on-demand" tooltips that display the testedness of any specified cell or dataflow relationship. In Fig. 1, the user has chosen to examine a black arrow leading into `Course_Avg`, which shows that the relationship between `Quiz_Avg` and `Course_Avg` is 100 percent tested.

WYSIWYT's purpose is to help identify failures and it has been empirically shown to be useful to both programmers and end users [36], [52]; 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 build on our WYSIWYT methodology. We describe these fault localization techniques in Section 3.

2.2 Related Work

Most fault localization research has been based on program slicing [66] and dicing [16], [38] techniques; Tip [62] surveys much 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], [35] and make use of that information using heuristics inspired by dicing.

2.2.1 Fault Localization for Professional Programmers

There has been a great deal of work on fault localization strategies for professional programmers (e.g., [3], [11], [19], [22], [28], [31], [38], [46], [49], [63]). For example, Agrawal et al. [3] present a technique, implemented as a tool called χslice, for locating faults in traditional programming

2. For color blind users, WYSIWYT displays red colors in light gray, blue colors in black, and "intermediate" colors between red and blue along a light-gray-to-black continuum.

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. [31] 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 [28] 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 [49] 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 immediately after each single program edit.

Pan and Spafford [41] developed a family of 20 heuristics appropriate for automated fault localization. These heuristics are based on the program statements exercised by passing and failing tests. Our strategies relate to three heuristics that involve 1) the set of all program points exercised by failed tests, 2) program points that are exercised by a large number of failed tests, and 3) cells that are exercised by failing tests and that are not exercised by passing tests.

Approaches similar to fault localization have been considered in domains other than software engineering. For example, in the domain of systems engineering, Failure Mode Effect Analysis (FMEA) [61] is a methodological approach to help engineers identify and propagate the effect of failure modes in products and processes. In the programming language domain, the propagation of type information in local type inference [45] has similarities to our fault localization techniques, especially the Nearest Consumers Technique in Section 3.2.2.

2.2.2 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 [7] 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 [64], [65] 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 [34] 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 [33], 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 a robustness feature built into our techniques (described in Section 3).

There is other work that can help end users find faults. S2 [58] 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. Clermont and Mittermeir [17], [18] define methods for aggregating cells based on various properties in order to find structure in spreadsheets. Igarashi et al. [30] 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], [6], [24] 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 [39] 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 [13], [67].

3 Three Interactive Fault Localization Techniques

This section 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 and complexity analyses.

3.1 Overview and Goals

Fault localization support generally attempts to help programmers locate the causes of failures in two ways: 1) by indicating the areas in a program that the user should search when attempting to locate a fault, thereby *reducing the user's search space* from the entire program to a small subset, and 2) by indicating the areas in this search space that are most likely to contain a fault, thereby *prioritizing the sequence of the user's search* through this space.

In our particular prototype, which follows the spreadsheet paradigm, 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." X-marks trigger a *fault likelihood* estimation for each cell (with a nonconstant formula) that might have contributed to the failure. As its name suggests, fault likelihood estimates the possibility that a fault resides in the formula of a cell. In this paper, fault likelihood estimations are in the form of categorical levels such as "Low," "Medium," and "High." (Quantitative ranges such as 0-100 percent are not used because our techniques can never be 100 percent certain that a fault does, or does not, reside in a cell's

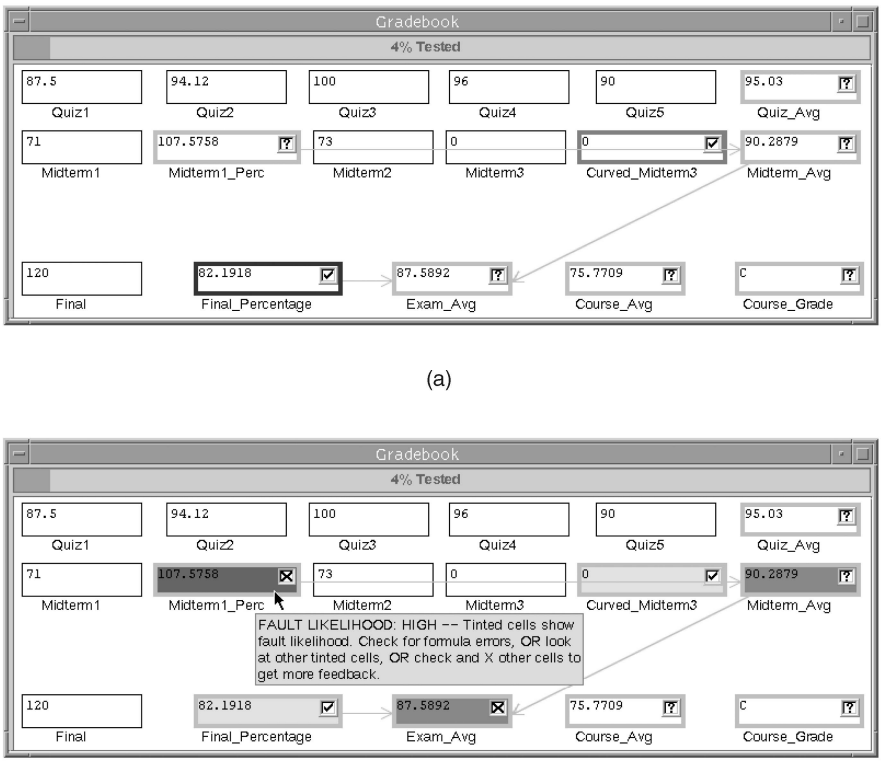

Fig. 3. (a) A `Gradebook` program at an early stage of testing. (b) The user notices incorrect values in `Exam_Avg` and `Midterm1_Perc`—the values are obviously too high—and places an X-mark in each cell's decision box. The user then hovers over the resulting dark highlighting in `Midterm1_Perc` to see an explanation of what the dark highlighting indicates.

formula.) Fault likelihood, updated for each affected 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 article).[3] This serves as our first goal of fault localization: reducing the user's search space.

Fault likelihood grows as additional X-marks are placed on spreadsheet cells. As the fault likelihood of a particular 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. (Similarly, as fault likelihood shrinks due to fewer X-marks or additional checkmarks, suspect cells are highlighted in lighter shades of red or gray.) 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 worked the `Gradebook` program to the stage shown in Fig. 3a. 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`.[4] 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 nonzero 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 in Fig. 3b. The `Midterm1_Perc` has an estimated fault likelihood of "High" (indicated by the darkest highlighting), `Midterm_Avg`, and `Exam_Avg` cells have an estimated fault likelihood of "Medium" (indicated by a medium highlighting), and the `Curved_Midterm3` and `Final_Percentage` cells have an estimated fault likelihood of "Very Low" (indicated by the lightest highlighting). The testedness borders of the cells with nonzero fault likelihood have faded to draw visual attention to the fault likelihood component of the cell.[5]

3.2 Three Fault Localization Techniques

Computing exact fault likelihood values for cells in order to display these fault localization colors, of course, is not possible. Instead, we combine heuristics with deductions made by analyzing the source code (formulas) and the user's tests to estimate fault likelihood. Because these techniques are meant for highly interactive visual environments, the computational costs of the techniques are especially important.

We now describe the algorithms for three fault localization techniques, including the information used by each

3. For color-blind users, as with WYSIWYT, the colors representing the fault likelihood of cells are displayed along a light-gray-to-black continuum.

4. In fact, in Fig. 3, the fault is an incorrect mathematical operation in the `Midterm1_Perc` formula (not shown in the figure).

5. Recent work [56] has suggested that certain facets of debugging may be improved by not fading these borders.

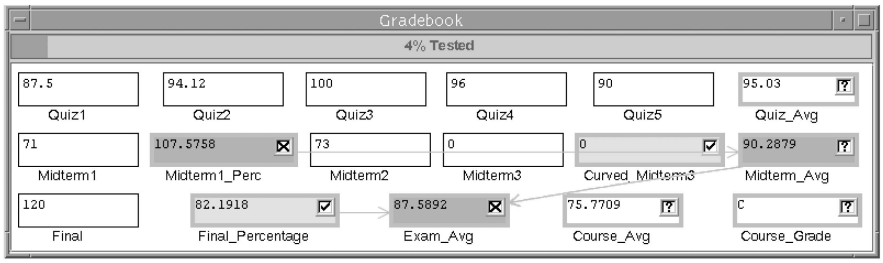

Fig. 4. The `Gradebook` program with the Blocking Technique.

technique for making deductions, the algorithms each technique uses or maintains in drawing further inferences from these deductions, and how much each technique's reasoning costs. In these descriptions, we 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 [66], 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 (with an X-mark or checkmark) on c or any of c's consumers.

3.2.1 The Blocking Technique

The Blocking Technique considers the dataflow relationships existing in the testing decisions that *reach* a cell c—hereafter, we refer to such reaching decisions as *unblocked*—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. Program dicing [16], [38] uses an approach that is similar to our Blocking Technique.

Fig. 4 demonstrates the Blocking Technique on the `Gradebook` spreadsheet 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 `Midterm_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. (In addition, had the `Midterm1_Perc` or `Exam_Avg` cells had consumers with checkmarks for the current set of input values, then the X-marks in these two cells would have blocked the effects of these checkmarks.)

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 the program 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 to or modify a nonconstant cell's formula (analogous to changing the program's logic).[6]

Changing Test Cases. Suppose that the user changes a constant cell's value and n nonconstant spreadsheet cells are affected. Any testing decisions on these n cells would have to be removed, although the effects of these decisions on fault likelihood information are preserved. (Testing decisions are also removed when a cell is affected by a change in the spreadsheet's program logic.) The cost of removing testing decisions on n cells is therefore $O(n)$. However, this cost is no more than that of the cell traversal that is already required by most spreadsheet evaluation engines to update the values of affected cells. Furthermore, the Blocking Technique can be implemented to piggy-back off of the evaluation engine's cell traversal to remove the testing decisions on affected cells while their values are being updated. This would result in only one $O(n)$ pass over the affected cells, rather than two $O(n)$ passes.

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 Algorithm 1) 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 responsible for updating data structures. The third pass, which is explained in Section 3.2.1, is responsible for mapping the information base of the technique into a fault likelihood level for each cell in `updateGUIList` and updating the user interface accordingly.

Algorithm 1 The `MarkPlaced` subroutine for the Blocking Technique's algorithm and the accompanying `BuildCellList` subroutine and `UpdateBlockingInfo` function. `MarkPlaced` is called when a testing decision is made on or removed from a cell. ("**by Ref**" indicates parameters passed by reference.)

```
1: procedure MarkPlaced (markedCell, aTest)
2:   let markedCell's current testing decision be aTest
3:   BuildCellList(markedCell, topoCellList
           by Ref)
4:   remove first element from topoCellList
```

6. 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.

```
5:    let updateGUIList = UpdateBlockingInfo
          (topoCellList, markedCell, aTest)
6:    estimate fault likelihood and update user interface for
          every cell in updateGUIList
7:  end procedure
8:
9:  procedure BuildCellList(aCell, topoCellList
          by Ref)
10:   if aCell has not yet been visited by BuildCellList
          then
11:     for all aUpwardCell that dynamically affect aCell
          do
12:       BuildCellList(aUpwardCell, topoCellList
          by Ref)
13:     end for
14:     add aCell to beginning of topoCellList
15:   end if
16: end procedure
17:
18: function UpdateBlockingInfo(topoCellList,
          markedCell, aTest)
19:   add markedCell to updateGUIList
20:   if aTest was added to markedCell by user then
21:     record that aTest blocks all tests reaching
          markedCell, and add aTest to markedCell
22:   else
23:     {aTest was removed}
24:     unblock tests formerly blocked by aTest and
          remove aTest from markedCell
25:   end if
26:   propagate markedCell's tests to create dynamically
          affecting cells' workLists
27:   for all sortedCell in topoCellList do
28:     add sortedCell to updateGUIList
29:     if sortedCell currently has a testing decision (i.e.,
          an X-mark or checkmark) then
30:       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
31:     end if
32:     copy sortedCell's workList changes to
          sortedCell's tests
33:     propagate sortedCell's workList changes to the
          workLists of affecting cells
34:   end for
35:   return updateGUIList
36: end function
```

The job of BuildCellList 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 topoCellList, which is returned to MarkPlaced. 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 UpdateBlockingInfo, so it is discarded in line 4.

UpdateBlockingInfo is responsible for maintaining the "blocking" characteristics of the Blocking Technique. Its job is to propagate the testing decision to the cellList returned by BuildCellList 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.[7]

The blocked and unblocked tests for markedCell are propagated to the workList of each cell dynamically affecting markedCell in line 26. Lines 27-31 process the topologicalcellList containing the dynamic backward slice that was previously created in BuildCellList. The workList of each cell in this list with a testing decision is checked for newly propagated tests that may be blocked by the cell's current testing decision. When this is complete, the changes are applied to each cell's information base in line 32 and propagated to affecting cells in line 33. 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 c be a cell in which a mark was placed, let p be the number of c's producers (cells in c's dynamic backward slice), 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. In the MarkPlaced algorithm, BuildCellList (line 3) is simply a depth-first search and, therefore, has $O(p+e)$ runtime complexity. UpdateBlockingInfo (line 5) iterates over the p producing cells to propagate tests to the worklists of the producing cells using the e edges,and then to process those worklists. When propagating tests to worklists (line 26), 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 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 $O((p+e)*m^2)$.

Changing the Spreadsheet Logic. When the user edits a nonconstant formula or changes a constant formula into a nonconstant formula, the spreadsheet's logic is changed. In this case, the NewFormula algorithm, which is outlined in Algorithm 2, 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

7. Temporary worklists are used, rather than propagating markCell's tests straight to an 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 every sortedCell in topoCellList.

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).

Algorithm 2 The `NewFormula` routine for the Blocking Technique and the accompanying `UndoTestEffects` function.

```
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 the cell 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 WYSIWYT
          testedness on aMarkedCell from these tests
21:   end if
22:   call UndoTestEffects(aCellToUpdate,
          editedCell, allEditedCellTests) for all
          aCellToUpdate statically affecting
          aMarkedCell and add returned lists to
          updateGUIList
23:   return updateGUIList
24: end function
```

In Algorithm 2, the `UndoTestEffects` function uses line 22 to perform 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 the `editedCell` that was given the new formula. `UndoTestEffects` removes all testing decisions to which the `editedCell` contributes in line 17, as well as the effects of those decisions on affecting cells in line 20. (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 depth-first search avoids visiting the same cell's formula data structure twice during a single invocation of `NewFormula`.

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[8] consisting of c's static backward slice 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 avoids visiting 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). 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 $O(e + p * m^2)$.

Mapping Information to Estimated Fault Likelihood. The Blocking Technique maps its base of information into fault localization feedback as follows: 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 $NumBlockedPassedTests(c)$ ($NBPT$) and $NumUnblockedPassedTests(c)$ ($NUPT$) have 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." This property ensures that every cell that might have contributed to the computation of an incorrect value will be assigned some nonzero 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.

If the previous two conditions do not hold, the Blocking Technique maps this testing information into an estimated fault likelihood for c using (1):

$$\text{fault likelihood}(c) = \max(1, 2 * NUFT - NUPT). \quad (1)$$

This estimated fault likelihood is transformed into a "0-5" fault likelihood level using the scheme presented in Table 1.[9] We evaluate the effectiveness of this mapping in Section 5.

The formula in (1) was chosen to ensure that, as the $NUFT$ of c increases, the fault likelihood of c increases, that is, fault likelihood is proportional to the number of unblocked X-marks in which c has participated. In contrast, as the $NUPT$ of c increases, the fault likelihood of c decreases, that is, fault likelihood is inversely proportional

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

9. Note that the "Very Low" fault likelihood level is not included in Table 1. Rather, that level is reserved exclusively for the previously mentioned robustness feature.

TABLE 1
The Mapping Used by the Blocking Technique to Transform Numerical Values to Discrete Fault Likelihood Levels

Computed FL	FL Level	Description
0	0	"None"
1–2	2	"Low"
3–4	3	"Medium"
5–9	4	"High"
10–∞	5	"Very High"

The level of 1 (not shown in this table) corresponds to the "Very Low" fault likelihood used by the robustness feature.

TABLE 2
The Mapping Used by the Nearest Consumers Technique to Transform Numerical Values to Discrete Fault Likelihood Levels

Computed FL	FL Level	Description
0	0	"None"
1	1	"Very Low"
2	2	"Low"
3	3	"Medium"
4	4	"High"
5–∞	5	"Very High"

to the number of unblocked checkmarks in which c has participated. ($NUFT$ was given a higher weight than $NUPT$ to prevent an equal number of X-marks and checkmarks from canceling each other out.) Also, if c or any of its consumers have at least one failed test, then the fault likelihood of c will be greater than zero. This is because the technique assigns c "Very Low" fault likelihood if none of these failed tests are reachable and, otherwise, uses (1), which ensures a fault likelihood greater than zero through its "max" operation.

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. These interactive activities may occur for very large spreadsheets, potentially making responsiveness an issue. For example, one corpus of spreadsheets assembled from the Internet found spreadsheets having as many as 26,434 formulas with references to other cells [26]. In these types of situations, the cost of fault localization techniques, such as the Blocking Technique, during these interactive activities 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 the first way this technique 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). Cell 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.

The Nearest Consumers Technique was used to create Fig. 3 on the same spreadsheet 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 Fig. 3 and Fig. 4 show that the `Final_Percentage` and `Curved_Midterm3` cells have the lowest estimated fault likelihood of all cells, while the other three cells have higher estimated fault likelihood. However, notice that these differences between the former two cells and the latter three cells have been exaggerated by the Nearest Consumers Technique at the bottom of Fig. 3. Clearly, the Nearest Consumers Technique does not always compute the same results as the Blocking Technique.

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.

Making a Testing Decision. When a testing decision is made, the Nearest Consumers Technique attempts to loosely mimic the Blocking Technique at a low cost using the algorithm outlined in Algorithm 3. Nearest Consumers begins by computing the numerical fault likelihood of cell c. In line 8 in the `MarkPlaced_Helper` subroutine, this numerical value is the average fault likelihood of c's direct consumers. This value is then mapped to a discrete fault likelihood level in line 9, as outlined in Table 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 previously visited by the technique and, so, this estimation for each cell should occur immediately.

Algorithm 3 The `MarkPlaced` subroutine for the Nearest Consumers Technique. `MarkPlaced_Helper` performs a breadth-first search of the producing cells of `markedCell`.

```
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 level
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?, retainLowFL?) for all
        directProd in aCell's direct producers
20  end procedure
```

The AdjustFaultLikelihood subroutine, which is called in line 10 of Algorithm 3, adjusts the fault likelihood of aCell based on trends it observes in the current testing decisions. This subroutine implements five rules; instead of presenting the algorithm for the subroutine, we present these five rules in the upcoming discussion.

The first three rules potentially adjust the fault likelihood of aCell that was initially estimated in lines 8-9 of Algorithm 3. It sequentially checks the conditions of each rule and applies 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 (previously calculated in line 8 of Algorithm 3), 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 and $avgFL(DC) >$ "None," then c is assigned a fault likelihood of "Very Low" to attempt to mimic the Blocking Technique's behavior of having checkmarks block tests.

Rule 2. If c has a failure (X-mark), but $avgFL(DC) <$ "Medium," then c is assigned a fault likelihood of "Medium." This assignment attempts to mimic the Blocking Technique's behavior of having X-marks block tests by preventing cells with 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 "None."

Rule 3. The fault likelihood from $avgFL(DC)$ is incremented one level whenever there are more X-marks than checkmarks in DC, 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.

After adjusting the fault likelihood of aCell through Rules 1-3, MarkPlaced_Helper applies two final rules to the fault likelihood in lines 11-18 of Algorithm 3. 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 helps to mimic the behavior 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 likelihood or to lower estimations. This rule helps mimic the behavior of the Blocking Technique by pruning off cells contributing to correct values.

The Nearest Consumer Technique also enforces the robustness feature 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 Algorithm 3. 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 producing 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 current 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.

Changing the Spreadsheet Logic. Unlike in the Blocking Technique, Nearest Consumers does not have to remove the effects of all testing decisions to which an edited nonconstant 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

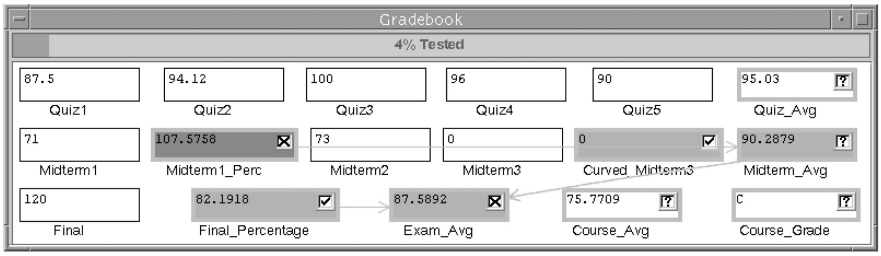

Fig. 5. The `Gradebook` program with the Test Count Technique.

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 AC, including c. Because the forward slice is traversed, all cells in AC 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 AC 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 done by calling the `NewFormula` procedure in Algorithm 4. Because this procedure requires a single breadth-first traversal up c's static backward slice, its runtime complexity is $O(p+d)$, where d is the number of direct consumers connected to the p producers in the backward slice.

Algorithm 4 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 Algorithm 3.

```
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
           editedCell's producers
4: end procedure
```

Mapping Information to Estimated Fault Likelihood. Table 2 shows that the Nearest Consumers Technique uses the same fault likelihood levels as the Blocking Technique, albeit with different numbers mapping to different levels. (This is due to the fact that Nearest Consumers uses averages to make fault likelihood estimations, rather than multiplications and subtractions.) Moreover, Nearest Consumers attempts to loosely mimic the Blocking Technique 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 earlier).

The `MarkPlaced_Helper` procedure in Algorithm 3 is responsible for averaging the fault likelihood of cells' direct consumers and adjusting this calculation based on the five rules. This mapping can be summarized as follows: Let DC, $avgFL(DC)$, xm, and cm have definitions similar to those earlier. Using Rule 3, an adjustment $z=1$ is made to the average fault likelihood of DC if any of the following conditions are true: 1) $xm>1$ and $cm=0$, 2) $xm>cm$ and $cm>0$, or 3) $xm>cm$ and c has an X-mark; otherwise, $z=0$. The fault likelihood of c is then calculated as follows:

$$\text{fault likelihood}(c) = avgFL(DC) + z. \tag{2}$$

The four exceptions to this calculation correspond to Rules 1, 2, 4, and 5; we refer readers to the previous discussion of those rules for details.

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 checkmarks) and *failed tests* (indicated via X-marks) 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. TARANTULA [31], a fault localization technique for traditional programming languages, uses an approach that is similar to our Test Count Technique.

An example of the previously introduced `Gradebook` spreadsheet with the Test Count Technique is provided in Fig. 5. 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."

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 mimic the blocking behavior of the Blocking Technique. Although the technique is similar to the Blocking Technique by making fault likelihood inversely proportional to the number of checkmarks in which the cell participates, the "Low" fault likelihood level is the lowest level 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.

This technique came about by leveraging algorithms and data structures that were written for another purpose—to support semi-automated test reuse [25] (regression testing) in the spreadsheet paradigm. The detailed algorithms and data structures used for test reuse purposes are detailed in that work [25]. Here, we present the algorithms relevant to fault localization support for the three interactions that could trigger action from the Test Count Technique.

Changing Test Cases. Changing the input values of the spreadsheet 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 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 $O(n * m)$.

Making a Testing Decision. The placement or removal of a testing decision triggers Algorithm 5. The algorithm adds or removes the appropriate testing decision from the cell under consideration in line 3 and then propagates that decision to the test histories of the cells in the dynamic backward slice of the marked cell in line 4. Finally, as the user interface is updated for these cells, a fault likelihood estimation is made for each cell c in line 5. Equation (3), which is used to make this estimation, is presented soon.

Algorithm 5 The `MarkPlaced` subroutine for the Test Count Technique.

```
1: procedure MarkPlaced (markedCell, aTest)
2:   let markedCell's current testing decision be aTest
3:   add/remove aTest to/from markedCell's test
         history as warranted
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
```

Using the test reuse algorithms [25], 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 spreadsheet and p is the number of c's producers.

Changing the Spreadsheet Logic. Changing cell c's nonconstant formula to a different formula, which triggers Algorithm 6, requires all saved information about c's tests and those of its consumers to be considered obsolete, which takes place in lines 2-3. For the same reason, the fault likelihood of c and its consumers must all be reinitialized to zero, which is performed in line 4.

Algorithm 6 The `NewFormula` subroutine for the Test Count Technique.

```
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
```

Using the test reuse methodology [25], all of the related testing information can be updated in $O(t * m * max(u, cost\ of\ set\ operations))$, 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 is the maximum number of uses for any cell in the spreadsheet.

TABLE 3
The Mapping Used by the Test Count Technique to Transform Numerical Values into Discrete Fault Likelihood Levels

Computed FL	FL Level	Description
0	0	"None"
1–2	1	"Low"
3–4	2	"Medium"
5–9	3	"High"
10–∞	4	"Very High"

Mapping Information to Estimated Fault Likelihood. The Test Count Technique mimics much of the behavior of the Blocking Technique. However, because it does not track the dataflow relationships between cells and testing decisions, it cannot maintain the "blocking" characteristics of the Blocking Technique. Instead, fault likelihood estimations are made purely by observing the number of X-marks and checkmarks in each cell's test history.

Let $NumFailingTests(c)$ (*NFT*) be the number of X-marks affected by c and let $NumSuccessfulTests(c)$ (*NST*) be the number of checkmarks affected by 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 using (3):

$$\text{fault likelihood}(c) = \max(1, 2 * NFT - NST). \quad (3)$$

This calculation is mapped to one of four possible fault likelihood levels using the scheme shown in Table 3.

Similarly to (1) in Section 3.2.1, (3) makes the fault likelihood of c proportional to the number of X-marks in which c has participated and inversely proportional to the number of checkmarks in which c has participated. The Test Count Technique also ensures that the fault likelihood of c will be greater than zero, for reasons similar to those of the Blocking Technique.

4 Two Factors Influencing Fault Localization Techniques

Our preliminary investigations [53], [57] into the effectiveness of our three fault localization techniques showed that the techniques can help pinpoint the locations of faults and can improve the debugging strategies of end-user programmers. However, post-hoc analysis indicated that there may be two distinct factors involved in the effectiveness of fault localization feedback. We therefore separated our fault localization techniques into these two factors, as follows:

- *Information Base.* To support the behavior of a fault localization technique, information must be stored and maintained either by the technique or by the surrounding environment. To abstract away implementation or algorithmic details such as data structures, we use this term to refer only to the *type* of information used and the circumstances under which it is maintained.
- *Mapping.* Mappings transform information bases into fault localization feedback. This transformation

involves reasoning about a cell's fault likelihood using the information provided by the information base and then transforming that calculation into appropriate feedback for the user.

In fact, we believe that any fault localization approach that includes some form of reporting or feedback to a human involves these two factors. For example, TARANTULA [31] uses a set of passed and failed tests and coverage information indicating the program points contributing to each test as its information base. Its mapping uses this information to calculate 1) a color representing each statement's participation in testing and 2) the technique's confidence in the correctness of each color.

There are some similarities between our notion of "information base" and "mapping" factors in fault localization techniques with the "model" and "view" in model-view-controller architectures. In these architectures, the model often consists of maintained data and the underlying algorithms, while the view dictates the way in which things appear to the user. A difference is that, in fault localization techniques, the reasoning that maps the information to fault localization values lies in our mappings, whereas this would lie in the "model" in model-view-controller architectures.

4.1 Information Bases

To support the behavior of a fault localization technique, information must be stored and maintained by the technique (or the surrounding environment). Most of the previous research into fault localization techniques has focused on approaches to maintaining different types of information and the cost of those approaches.

Each of our three techniques maintains a unique base of information that is used to achieve the behavior described in Section 3. The algorithms governing the maintenance of this information were described in Sections 3.2.1-3.2.3. We now draw from that presentation to summarize each technique's information base, which we refer to as `I-TC`, `I-BL`, and `I-NC` for the remainder of this paper.

- *Test Count* (`I-TC`). This technique's information base maintains, for each cell c, the set of passed and failed tests that dynamically execute c. Mappings estimate the fault likelihood of spreadsheet cells using this "history" of all testing decisions—both decisions for previous and current input values (constant spreadsheet cell values)—and how these decisions impact the producing cells that dynamically contribute to each decision (i.e., dynamic backward slices). The size of `I-TC` grows with respect to both spreadsheet and test suite size.
- *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 tracks the dataflow relationships between each cell, using this information to allow tests, under certain circumstances, to "block" other tests from reaching certain cells. Mappings estimate the fault likelihood for cells using this history of all testing decisions and the decisions' respective dataflow information and how these decisions impact their dynamically producing cells. However, because of the technique's blocking behavior, in many cases, testing decisions only play a role in the fault likelihood of the cells in each decision's program dice [38]. (The situations where this is not the case are those where our robustness feature in Section 3.2.1 prevents checkmarks that block X-marks from completely removing cells from this "dice.") Because of the overhead necessary to track dataflow and blocking information, `I-BL` is more computationally expensive than `I-TC`.
- *Nearest Consumers* (`I-NC`). Rather than maintaining a history of *all* previous testing decisions, as do the previous two information bases, the `I-NC` information base tracks only 1) the fault likelihood of each cell in the spreadsheet prior to a new testing decision and 2) the current testing decision for each cell *affected by the current test case* (i.e., the current set of input values), including that current test case. Since each of these components requires only constant space for each cell in the spreadsheet, the information base grows with respect to spreadsheet size only.

Because the context of our experiment 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 in an empirical setting is to learn whether a modest information base such as `I-NC` can compete in effectiveness with the other two more expensive information bases.

4.2 Mappings

The manner in which fault localization techniques draw from their information bases to produce feedback is through the mapping factor. Mappings transform information bases into fault localization feedback that fulfills the goals of the fault localization technique. In the case of our three techniques, the goals that the mappings must fulfill are outlined at the beginning of Section 3.

For any fault localization technique, many mappings are possible: Not only are there a large number of different ways to transform an information base into feedback, but the characteristics of this feedback, such as the number of different levels in the feedback (e.g., in our techniques, this is the number of discrete fault likelihood levels), can vary. Thus, it would not be feasible to compare all possible mappings and doing so is not even warranted until we determine whether the mapping factor alone can significantly impact a technique's effectiveness. Surprisingly, in our search of the research literature, we can find no previous work that has investigated this possibility.

We use the mappings of our own three techniques as a vehicle for investigating the importance of mapping as an independent factor. The mappings of each technique were described in Sections 3.2.1-3.2.3 and in (1)-(3). We now draw from that presentation to summarize each technique's mapping, which we refer to as `M-TC`, `M-BL`, and `M-NC` for the remainder of this paper.

- *Test Count* (M-TC). The Test Count Technique's mapping ensures that the fault likelihood of a cell c is inversely proportional to the number of c's passed tests and directly proportional to the number of c's failed tests. It maps 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 increase 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.
- *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 Section 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 make it viable to both increase and decrease fault likelihood values as c's direct consumers' fault likelihood values 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 [57] 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 (described in Section 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 incorrectly placed checkmarks cannot cause a cell that is involved in at least one correctly placed X-mark to be removed from a user's search space.

5 Experiment

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) as, for example, we ourselves have done in previous studies [53], [57] reporting various results regarding the use of fault localization by end users. An undesirable consequence of this is that the effectiveness (or ineffectiveness) of a technique may be wrongly attributed to the technique's information base, when, instead, it may be due to the mapping. To investigate these two individual factors in fault localization techniques in the context of an interactive end-user programming environment, we conducted a new, controlled experiment, with the following two research questions:

RQ1: To what extent do differences in information bases affect the effectiveness of spreadsheet fault localization techniques?

RQ2: To what extent do differences in mappings affect the effectiveness of spreadsheet fault localization techniques?

Previous empirical work [33], [34], [57] has indicated that end users make mistakes during their interactive testing and debugging tasks and any differences between the information bases and mappings of fault localization techniques may be exaggerated or diminished when isolating situations where techniques must operate in the presence of unreliable information. Further, researchers have only recently begun to consider these types of situations when evaluating interactive debugging devices for end users. To investigate each factor's role in technique effectiveness in the presence of such unreliable information, we devised two additional research questions:

RQ3: To what extent does inaccurate information affect information bases and the effectiveness of spreadsheet fault localization techniques?

RQ4: To what extent does inaccurate information affect mappings and the effectiveness of spreadsheet fault localization techniques?

5.1 Design

In formulating our experiment, we considered three 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. Although this methodology would allow us to elicit test suites from real end users, it has a significant disadvantage: Each technique that we would compare would be given different testing actions (i.e., different participants would place different testing actions, in the form of WYSIWYT X-marks and checkmarks; thus, each technique would work with different testing actions). This would make it impossible to ensure that differences in test suites were not confounding any results, thereby preventing us from addressing our research questions regarding whether differences in fault localization factors alone can impact the effectiveness of fault localization techniques.

A methodology that avoids this flaw involves following 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 a third methodology that draws on advantages from both of the foregoing approaches, 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 objects of analysis of our experiment.

TABLE 4
A Summary of the Participants' General, Educational Background, Previous Spreadsheet Experience, and Previous Programming Experience

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

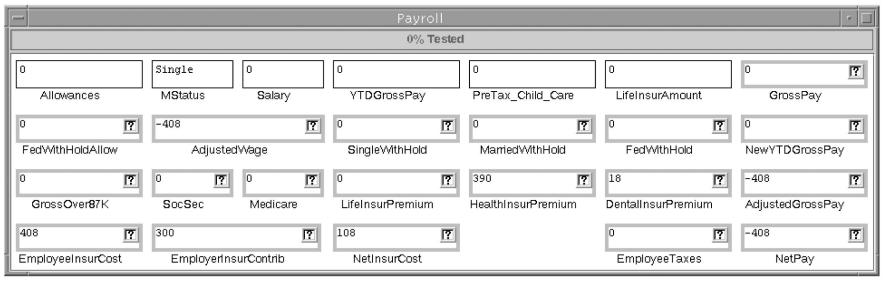

Fig. 6. The `Payroll` task.

5.1.1 Participants

To obtain the necessary test suites, we recruited 20 students (18 undergraduate students and two graduate students) from Oregon State University. We sought 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 programming experience in a professional setting, 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 4. In the education category of this table, "LA" indicates the number of liberal arts participants, "Bus" indicates business participants, "Eng" indicates engineering participants, and "GPA" indicates 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 use in a professional setting.

5.1.2 Materials

The experiment utilized two spreadsheets, `Gradebook` and `Payroll` (shown in Fig. 3 and Fig. 6, respectively). To make our spreadsheets representative of those used by real end-user programmers, `Gradebook` was derived from an Excel spreadsheet of an (end-user) instructor, which we ported into an equivalent Forms/3 spreadsheet. `Payroll` was a spreadsheet designed by two Forms/3 researchers from a payroll description from a real company.

These spreadsheets were seeded with five faults created by actual end users. To obtain these faults, we provided three separate end users with the following: 1) a "template" spreadsheet with cells and cell names, but no cell formulas and 2) a description of how each spreadsheet 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 using the template and the description.

From the collection of faults left in these end users' final spreadsheets, we chose five that provided coverage of the categories in Panko's classification system [43], which is based on Allwood's classification system [5]. Under Panko's system, mechanical faults include simple typographical errors or incorrect cell references. Logical faults are mistakes in reasoning and are more difficult to detect and correct than mechanical faults. An omission fault is information that has never been entered into a cell formula and is the most difficult to detect [43].

When classifying these faults, we realized that Panko's and Allwood's schemes do not always clearly differentiate types of faults. For example, the schemes do 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 Table 5 and Table 6. `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.

TABLE 5
The Faults Seeded in the `Gradebook` Task

Cell Name (*Fault Type*)	Faulty Formula	Correct Formula
Quiz_Avg (*Mechanical: Typographical*)	((Quiz1 + Quiz2 + Quiz3 + Quiz4 + Quiz5) - (min Quiz1 Quiz2 Quiz3 Quiz4 Quiz5)) / 5	((Quiz1 + Quiz2 + Quiz3 + Quiz4 + Quiz5) - (min Quiz1 Quiz2 Quiz3 Quiz4 Quiz5)) / 4
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 else 0	if Midterm3 > 0 then Midterm3 + 2 else 0

TABLE 6
The Faults Seeded in the `Payroll` Task

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 (AdjustedWage – 248) * 0.10
AdjustedGrossPay (*Mechanical: Reference*)	GrossPay – PreTax_Child_Care – EmployeeInsurCost	GrossPay – PreTax_Child_Care – NetInsurCost
SingleWithHold (*Logical*)	if AdjustedWage < 119 then 0 else (AdjustedWage – 248) * 0.10	if AdjustedWage < 119 then 0 else (AdjustedWage – 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

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 percent.

5.1.3 Variables and Measures

As a dependent variable, we require a measure of a fault localization technique's effectiveness. Many such measures are possible. In this experiment, our goal is to study the ability of techniques to point out faults by applying *identical* test suites uniformly. Thus, we define *effectiveness* as a technique's ability to correctly and visually differentiate the correct cells in a spreadsheet from those cells that actually contain faults. The better a technique visually distinguishes a program's faulty cells from its correct cells, the more effective the technique.

In this experiment, we considered two notions of effectiveness, resulting in two effectiveness metrics. For our first notion, we measure effectiveness in terms of the *visual separation* between the faulty cells and the correct cells of each spreadsheet, which is the result of subtracting the average fault likelihood of the colored correct cells from the average fault likelihood of the colored faulty cells. (Subtraction is used instead of calculating a ratio because the color choices form an ordinal, not a ratio, scale.) Our previous work [57] indicated that users usually restrict their attention during debugging only to the cells indicated in the fault localization feedback (i.e., the colored cells). Given this finding, this first effectiveness measure focuses only on the cells colored by the technique.

Let C be the set of all cells in a spreadsheet, let $fault(c)$ be a function that maps a cell $c \in C$ to zero if c does not contain a fault and one if c contains one or more faults, let $FL(c)$ be the fault likelihood of any given cell $c \in C$, and let $avgFL(C)$ be the average fault likelihood of the set of cells C. The effectiveness (`Eff-Color`) of a technique according to this first measure is measured in the following way:

$$\begin{aligned} C_f &= \{c \mid c \in C, fault(c) = 1, FL(c) > 0\}, \\ C_{nf} &= \{c \mid c \in C, fault(c) = 0, FL(c) > 0\} \\ \texttt{Eff-Color} &= avgFL(C_f) - avgFL(C_{nf}). \end{aligned} \tag{4}$$

Our second effectiveness measure is similar to the first, except that, rather than considering only the colored faulty and correct cells, the second measure considers *all* faulty and correct cells—that is, the result of subtracting the average fault likelihood of all correct cells from the average fault likelihood of all faulty cells. Since all faulty and correct cells are considered, this metric also has the effect of rewarding techniques with a smaller search space size. This is because faulty cells with no fault likelihood will have the effect of lowering the average fault likelihood of faulty cells, thereby lowering effectiveness, while correct cells with no fault likelihood will have the effect of lowering the average fault likelihood of correct cells, thereby increasing effectiveness.

Let C, $fault(c)$, $FL(c)$, and $avgFL(C)$ have the same definitions as in `Eff-Color`. The effectiveness (`Eff-All`) of a technique according to this measure is measured in the following way:

$$\begin{aligned} C_f &= \{c \mid c \in C, fault(c) = 1\}, \\ C_{nf} &= \{c \mid c \in C, fault(c) = 0\} \\ \texttt{Eff-All} &= avgFL(C_f) - avgFL(C_{nf}). \end{aligned} \tag{5}$$

Positive effectiveness is preferable for both effectiveness measures and a greater effectiveness implies a better distinction between faulty and nonfaulty cells.

To investigate whether differences in information bases and mappings can impact the effectiveness of fault localization techniques, we separately manipulate each

factor for the research question under consideration. From our selection of an information base or a mapping in a fault localization technique, we can measure the effectiveness of the technique's feedback according to our dependent variables in order to gauge whether changes in the manipulated factor cause a significant difference in the quality of the feedback. The selected information bases in RQ1 and RQ3, or mappings in RQ2 and RQ4, are therefore the independent variables of our experiment.

Without considering the attributes that "bind" a mapping to an information base, the essence of the differences among our three mappings is two shared characteristics: 1) the number of possible fault likelihood values and 2) an "initial" value used to start assigning fault likelihood feedback. When we manipulate the two factors to apply different information bases to different mappings, we refer to applying only these two characteristics of one mapping to another to create an entirely new mapping for that technique. We do not attempt to tease apart the influences of these two characteristics, but simply consider them together to learn whether changing the mapping factor of a technique can significantly impact that technique's effectiveness.

5.1.4 Points of Evaluation

At what point should effectiveness be measured? Most previous work focuses on "traditional" techniques for professional programmers that perform 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 an appropriate point to measure these types of 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.

In principle, we could measure effectiveness at every point at which a user receives feedback. However, it is not statistically viable to utilize every such point, because many will not be reached by numbers of users sufficient to support comparisons. Therefore, we elect to measure at just 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 measure at the third X-marks, fourth X-marks, and so on, but the participants 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. Once a user edits the "source code" (formula), downstream fault localization information becomes obsolete and is discarded.

We emphasize that the need to evaluate at multiple 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. Otherwise, the experiment may be overlooking important data reported by the technique.

5.1.5 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 debugging 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 spreadsheet 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` spreadsheets (tasks) with instructions to test and correct any errors found in the spreadsheets. The participants were also provided with two correct sequences of input and output for 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 spreadsheets 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.

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 earlier empirical work [53], [57], we chose to use the `I-TC` information base with the `M-BL` mapping. We then applied the testing actions collected using this technique across all information base and mapping combinations.

5.1.6 Threats to Validity

Every experiment has threats to the validity of its results and these threats must be considered in order to assess the meaning and impact of results. (Wohlin et al. [68] provide a general discussion of validity evaluation and a classification of validity threats.) This section discusses potential threats to the validity of our experiment and, where possible, how we attempted to mitigate the impact of these threats on our results.

Threats to External Validity. Threats to external validity limit the extent to which results can be generalized.

If the spreadsheets used in our experiment did not represent those that real end users create, our results may not generalize. To reduce this threat, we obtained "real-world" spreadsheets from an actual end user who taught a course at the university where this work was performed and an actual payroll description. Also, to better control experiment conditions and ensure that participants could complete the tasks in the allotted time,[10] our spreadsheets were not large. While end-user spreadsheets can be of varying size, including both large and small spreadsheets, our results should be interpreted in the context of this limitation. Future empirical work gathering additional empirical evidence using a greater range and number of spreadsheets would reduce this threat, although such studies could require days for very large, industrial-sized spreadsheets and would therefore have to be conducted in uncontrolled, nonlaboratory settings, thereby sacrificing the degree of control that this experiment sought to achieve.

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 5.1.2. Also, to help control the threats to internal validity, we selected faults from the end users according to classification schemes [5], [43]. 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 spreadsheet environment [12]. However, end users may debug differently in different environments.

All of these external validity concerns can be addressed only through repeated studies, using different spreadsheet tasks, faults, and spreadsheet environments.

Threats to Internal Validity. Threats to internal validity are other factors that may be responsible for an experiment's results.

The specific types of faults seeded in a program can affect fault localization results. To reduce this threat, as described in Section 5.1.2, we selected faults according to Panko's classification scheme [43] to ensure that different types of faults were included.

As mentioned in Section 5.1.5, 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. However, this trade-off 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 the functionality of the spreadsheets that they worked on sufficiently to correct the faults. Also, the participants could have learned at different rates either during our tutorial or during the tasks themselves and this learning factor could have played a role in our results. Finally, the study's time limits could have interacted with the learning styles of some participants. However, as described in Section 5.1.5, 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.

Threats to Construct Validity. Threats to construct validity question whether the measures in an experiment's design adequately capture the effects that they should or that they were intended to.

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 technique's 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.

5.2 Results

5.2.1 RQ1: The Information Base Factor

To investigate RQ1, which pertains to the impact of the information base factor on techniques' effectiveness, in isolation from the mapping factor, we compared the information bases' effectiveness three times, once under each mapping described in Section 4.2. The comparisons were done at the three feedback points, based on debugging sessions, described in Section 5.1.4. In total, there were 18 debugging sessions in the `Gradebook` task and 13 in the `Payroll` task. (A second X-mark was placed in five of the 18 sessions for `Gradebook` and in three of the 13 sessions for `Payroll`.)

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 Table 7. Each table has six subtables, (a)-(f), which depict the effectiveness of the feedback for the

10. Two hours were required for each participant to learn the mechanics of the spreadsheet environment, work on the two moderately-sized tasks, and complete postsession questionnaires.

TABLE 7
Isolating the Information Base Factor with the Specified Mapping: (a) Mapping M-TC, Metric Eff-Color, (b) Mapping M-TC, Metric Eff-All, (c) Mapping M-BL, Metric Eff-Color, (d) Mapping M-BL, Metric Eff-All, (e) Mapping M-NC, Metric Eff-Color, and (f) Mapping M-NC, Metric Eff-All

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.8948$)	**0.389** **(0.502)**	0.259 (0.622)	**0.389** **(0.502)**
Payroll ($n = 13$) ($p = 0.1211$)	0.000 (0.000)	-0.166 (0.404)	**0.039** **(0.075)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	0.000 (1.000)	**0.166** **(0.764)**	0.000 (1.000)
Payroll ($n = 5$) ($p = n/a$)	0.155 (0.458)	0.011 (0.122)	**0.183** **(0.489)**
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.4389$)	-0.056 (0.539)	**0.004** **(0.493)**	-0.038 (0.512)
Payroll ($n = 13$) ($p = 0.0608$)	0.127 (0.280)	-0.118 (0.476)	**0.210** **(0.497)**

(a)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.6065$)	**0.170** **(0.156)**	0.130 (0.222)	**0.170** **(0.156)**
Payroll ($n = 13$) ($p = 0.1637$)	-0.041 (0.160)	-0.046 (0.115)	**0.057** **(0.157)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	**0.133** **(0.231)**	**0.133** **(0.340)**	**0.133** **(0.231)**
Payroll ($n = 5$) ($p = n/a$)	**0.354** **(0.402)**	0.175 (0.181)	0.342 (0.411)
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.8890$)	0.027 (0.204)	0.027 (0.204)	**0.034** **(0.205)**
Payroll ($n = 13$) ($p = 0.0018$)	0.145 (0.388)	0.434 (0.338)	**0.437** **(0.486)**

(b)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.7165$)	0.831 (0.841)	0.859 (1.005)	**0.943** **(0.923)**
Payroll ($n = 13$) ($p = 0.1000$)	0.294 (0.425)	0.347 (0.327)	**0.487** **(0.397)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	0.333 (2.082)	**0.500** **(1.803)**	**0.500** **(1.803)**
Payroll ($n = 5$) ($p = n/a$)	0.372 (0.947)	0.359 (0.595)	**0.560** **(0.830)**
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.4464$)	0.265 (1.116)	0.331 (1.063)	**0.376** **(1.034)**
Payroll ($n = 13$) ($p = 0.0128$)	0.302 (0.602)	0.596 (0.532)	**0.768** **(0.664)**

(c)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.6065$)	0.351 (0.188)	0.368 (0.269)	**0.376** **(0.185)**
Payroll ($n = 13$) ($p = 0.1353$)	0.026 (0.175)	0.098 (0.293)	**0.221** **(0.249)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	0.350 (0.541)	**0.433** **(0.404)**	**0.433** **(0.404)**
Payroll ($n = 5$) ($p = n/a$)	0.621 (0.741)	**0.757** **(0.593)**	0.699 (0.740)
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.3679$)	0.180 (0.303)	0.196 (0.384)	**0.231** **(0.286)**
Payroll ($n = 13$) ($p = 0.0555$)	0.574 (0.505)	0.986 (0.788)	**0.878** **(0.700)**

(d)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.0923$)	1.146 (1.322)	1.394 (1.374)	**1.496** **(1.385)**
Payroll ($n = 13$) ($p = 0.0695$)	0.690 (0.784)	0.543 (0.652)	**0.935** **(0.796)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	0.500 (2.783)	0.833 (2.566)	**1.000** **(2.646)**
Payroll ($n = 5$) ($p = n/a$)	0.693 (1.170)	0.542 (1.055)	**0.936** **(1.217)**
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.0022$)	0.274 (1.519)	0.756 (1.477)	**0.791** **(1.612)**
Payroll ($n = 13$) ($p = 0.0199$)	0.646 (0.859)	0.833 (0.879)	**1.268** **(0.955)**

(e)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.1324$)	0.491 (0.315)	0.579 (0.377)	**0.581** **(0.244)**
Payroll ($n = 13$) ($p = 0.1382$)	0.135 (0.337)	0.141 (0.294)	**0.370** **(0.412)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	0.483 (0.725)	0.650 (0.606)	**0.733** **(0.643)**
Payroll ($n = 5$) ($p = n/a$)	0.939 (0.886)	**1.056** **(0.981)**	**1.056** **(1.079)**
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 18$) ($p = 0.0072$)	0.212 (0.459)	0.390 (0.492)	**0.427** **(0.418)**
Payroll ($n = 13$) ($p = 0.1017$)	0.894 (0.699)	1.212 (0.976)	**1.246** **(0.965)**

(f)

specified metric, information base, and mapping at each point of evaluation. The mean (standard deviation) effectiveness values comparing the three information bases are shown. The information base with the greatest average effectiveness is shown in bold. We used the Friedman test [60] 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.) The "p" denotes p-values of the Friedman tests and "n" denotes the number of observations compared at each point. (Prior to this experiment, we defined the 0.05 level of significance as a prerequisite for rejecting hypotheses. However, we separately denote, in the text, p-values reaching the 0.01 level of significance to provide a more rich accounting of the results. We also point out significance at the 0.10 level for similar reasons; however, we do not consider this significance level to be sufficient for rejecting hypotheses.)

Before discussing the results of the Friedman tests in Table 7, we describe one table in detail to illustrate our results. In Table 7a, the mapping M-TC is held and the three information bases are varied; this has the effect of isolating the impact of the information base factor. The mean effectiveness of the three resulting techniques, according to the Eff-Color metric, are shown for the Gradebook and Payroll tasks at each point of measurement described in Section 5.1.4. For each task at each evaluation point, the technique(s) with the best effectiveness are shown in bold. On the leftmost column under the task name, the number of debugging sessions reaching each evaluation point is shown as n; for example, in Table 7a, there were 18 sessions that had a "First X-mark" and a "Last Test" for Gradebook and three sessions that had a "Second X-mark." The p-values of the Friedman tests for each set of observations are also shown in this area; for example, the p-value for Payroll at the "Last Test" of the debugging sessions is 0.0608.

Table 7a shows marginal significance (at the 0.10 level) at the Last Test of the Payroll task using the Eff-Color metric. Table 7b 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 7c 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 7d 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.

Differences were even more pronounced using the M-NC mapping, as shown in Table 7e and Table 7f. This was especially true using the Eff-Color metric (Table 7e) 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, Table 7 indicates that the I-NC information base, which is the basis of the inexpensive Nearest Consumers technique, may be the most effective of the three information bases—I-NC showed the highest average effectiveness at almost every point measured. The implications of this are discussed in Section 5.3.

5.2.2 RQ2: The Mapping Factor

How important is mapping alone to technique effectiveness? The tables in Section 5.2.1 are suggestive in this regard. To statistically consider whether this factor had a 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* three mappings *with the* I-TC *information base.*

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

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

As Table 8 shows, 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.

5.2.3 RQ3: Information Base Robustness

As our first step in investigating the pervasiveness of mistakes, we counted the number of incorrect testing decisions made in each 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 percent of the checkmarks and 5.95 percent of the X-marks were incorrect. This trend continued in Payroll, where 20.62 percent of the checkmarks and 3.33 percent of the X-marks were incorrect. These results corroborate the findings of an earlier formative study [57], where many more incorrect checkmarks were placed than incorrect X-marks. In that previous study, we found evidence suggesting that our end-user participants placed checkmarks if they thought that a cell's value could possibly be correct. In both that previous study and this paper's experiment, it appears that this liberal use of checkmarks resulted in many incorrect testing decisions.

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 the effectiveness at each First X-mark, Second X-mark, and Last Test *that was in the context of at least one incorrect testing decision*; this occurred for 13 of the 18 debugging sessions in the Gradebook task and for 10 of the 13 sessions in Payroll. (In the debugging sessions in which a second X-mark was placed, all three such sessions in Gradebook took place in the presence of at least one incorrect testing decision, as did three of the five sessions for Payroll.) We isolated information bases using the same procedure as in Section 5.2.1.

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.*

TABLE 8
Isolating the Mapping Factor with the Specified Information Base: (a) Information Base I-TC, Metric Eff-Color, (b) Information Base I-TC, Metric Eff-All, (c) Information Base I-BL, Metric Eff-Color, (d) Information Base I-BL, Metric Eff-All, (e) Information Base I-NC, Metric Eff-Color, and (f) Information Base I-NC, Metric Eff-All

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	0.389	0.831	**1.146**
($p = 0.0031$)	(0.502)	(0.841)	**(1.322)**
Payroll ($n = 13$)	0.000	0.294	**0.690**
($p = 0.0060$)	(0.000)	(0.425)	**(0.784)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$)	0.000	0.333	**0.500**
($p = n/a$)	(1.000)	(2.082)	**(2.784)**
Payroll ($n = 5$)	0.155	0.372	**0.693**
($p = n/a$)	(0.458)	(0.947)	**(1.170)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	-0.056	0.265	**0.274**
($p = 0.1180$)	(0.539)	(1.116)	**(1.519)**
Payroll ($n = 13$)	0.128	0.302	**0.646**
($p = 0.1220$)	(0.280)	(0.602)	**(0.859)**

(a)

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	0.170	0.351	**0.491**
($p < 0.0001$)	(0.156)	(0.188)	**(0.315)**
Payroll ($n = 13$)	-0.041	0.026	**0.135**
($p = 0.2490$)	(0.160)	(0.175)	**(0.337)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$)	0.133	0.350	**0.483**
($p = n/a$)	(0.160)	(0.541)	**(0.725)**
Payroll ($n = 5$)	0.354	0.621	**0.939**
($p = n/a$)	(0.402)	(0.741)	**(0.886)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	0.027	0.180	**0.212**
($p = 0.0285$)	(0.204)	(0.303)	**(0.459)**
Payroll ($n = 13$)	0.434	0.574	**0.894**
($p = 0.0025$)	(0.338)	(0.505)	**(0.699)**

(b)

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	0.259	0.859	**1.394**
($p = 0.0004$)	(0.622)	(1.005)	**(1.374)**
Payroll ($n = 13$)	-0.166	0.347	**0.543**
($p = 0.0016$)	(0.404)	(0.327)	**(0.652)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$)	0.167	0.500	**0.833**
($p = n/a$)	(0.764)	(1.803)	**(2.566)**
Payroll ($n = 5$)	0.011	0.358	**0.542**
($p = n/a$)	(0.122)	(0.595)	**(1.055)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	0.004	0.331	**0.756**
($p = 0.0424$)	(0.493)	(1.063)	**(1.477)**
Payroll ($n = 13$)	-0.118	0.596	**0.833**
($p = 0.0001$)	(0.476)	(0.532)	**(0.878)**

(c)

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	0.130	0.368	**0.579**
($p < 0.0001$)	(0.222)	(0.269)	**(0.377)**
Payroll ($n = 13$)	-0.046	0.098	**0.141**
($p = 0.3526$)	(0.115)	(0.293)	**(0.294)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$)	0.133	0.433	**0.650**
($p = n/a$)	(0.340)	(0.404)	**(0.606)**
Payroll ($n = 5$)	0.175	0.757	**1.056**
($p = n/a$)	(0.181)	(0.593)	**(0.981)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	0.027	0.196	**0.390**
($p = 0.0056$)	(0.204)	(0.384)	**(0.492)**
Payroll ($n = 13$)	0.145	0.986	**1.212**
($p = 0.0001$)	(0.388)	(0.788)	**(0.976)**

(d)

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	0.389	0.943	**1.496**
($p = 0.0001$)	(0.502)	(0.923)	**(1.385)**
Payroll ($n = 13$)	0.039	0.488	**0.935**
($p = 0.0005$)	(0.075)	(0.397)	**(0.796)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$)	0.000	0.500	**1.000**
($p = n/a$)	(1.000)	(1.803)	**(2.646)**
Payroll ($n = 5$)	0.183	0.560	**0.936**
($p = n/a$)	(0.489)	(0.830)	**(1.217)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	-0.039	0.376	**0.791**
($p = 0.0045$)	(0.512)	(1.034)	**(1.612)**
Payroll ($n = 13$)	0.210	0.768	**1.268**
($p = 0.0036$)	(0.497)	(0.664)	**(0.955)**

(e)

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	0.170	0.376	**0.581**
($p < 0.0001$)	(0.156)	(0.185)	**(0.244)**
Payroll ($n = 13$)	0.057	0.221	**0.369**
($p = 0.0366$)	(0.157)	(0.249)	**(0.412)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$)	0.133	0.433	**0.733**
($p = n/a$)	(0.231)	(0.404)	**(0.643)**
Payroll ($n = 5$)	0.342	0.699	**1.056**
($p = n/a$)	(0.411)	(0.740)	**(1.079)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 18$)	0.034	0.231	**0.427**
($p = 0.0001$)	(0.205)	(0.286)	**(0.418)**
Payroll ($n = 13$)	0.437	0.878	**1.246**
($p = 0.0007$)	(0.486)	(0.700)	**(0.965)**

(f)

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 Table 9a, Table 9b, Table 9c, and Table 9d, there were no significant differences (at the 0.05 or 0.01 levels) among the techniques, so we cannot reject H7 or

TABLE 9
Isolating the Information Base Factor with the Specified Mapping at Feedback Points that Were in the Context of at Least One Incorrect Testing Decision: (a) Mapping M-TC, Metric Eff-Color, (b) Mapping M-TC, Metric Eff-All, (c) Mapping M-BL, Metric Eff-Color, (d) Mapping M-BL, Metric Eff-All, (e) Mapping M-NC, Metric Eff-Color,and (f) Mapping M-NC, Metric Eff-All

(a)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.8669$)	**0.769** **(1.013)**	0.436 (1.013)	**0.769** **(1.013)**
Payroll ($n = 10$) ($p = 0.2231$)	0.000 (0.000)	-0.202 (0.458)	**0.051** **(0.083)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	0.333 (1.528)	**0.500** **(1.323)**	0.333 (1.528)
Payroll ($n = 3$) ($p = n/a$)	0.292 (0.505)	0.067 (0.115)	**0.314** **(0.543)**
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.3679$)	-0.308 (0.630)	**-0.187** **(0.673)**	-0.285 (0.608)
Payroll ($n = 10$) ($p = 0.1030$)	0.149 (0.286)	-0.195 (0.465)	**0.159** **(0.394)**

(b)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.5134$)	**0.385** **(0.215)**	0.283 (0.295)	**0.385** **(0.215)**
Payroll ($n = 10$) ($p = 0.1211$)	-0.081 (0.364)	-0.050 (0.227)	**0.155** **(0.335)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	**0.350** **(0.409)**	**0.350** **(0.541)**	**0.350** **(0.409)**
Payroll ($n = 3$) ($p = n/a$)	**0.454** **(0.606)**	0.347 (0.134)	0.394 (0.592)
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.8669$)	0.065 (0.385)	0.056 (0.392)	**0.076** **(0.395)**
Payroll ($n = 10$) ($p = 0.0727$)	**0.704** **(0.439)**	0.403 (0.307)	0.609 (0.478)

(c)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.7165$)	0.821 (0.798)	0.859 (1.032)	**0.974** **(0.915)**
Payroll ($n = 10$) ($p = 0.1309$)	0.262 (0.416)	0.317 (0.296)	**0.500** **(0.398)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	0.333 (2.082)	**0.500** **(1.803)**	**0.500** **(1.803)**
Payroll ($n = 3$) ($p = n/a$)	0.743 (0.908)	0.492 (0.709)	**0.775** **(0.999)**
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.4464$)	0.036 (1.092)	0.128 (1.037)	**0.190** **(1.007)**
Payroll ($n = 10$) ($p = 0.0539$)	0.311 (0.587)	0.488 (0.394)	**0.698** **(0.610)**

(d)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.6065$)	0.363 (0.143)	0.386 (0.268)	**0.397** **(0.132)**
Payroll ($n = 10$) ($p = 0.2053$)	0.024 (0.192)	0.087 (0.319)	**0.267** **(0.260)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	0.350 (0.541)	0.433 (0.404)	**0.433** **(0.404)**
Payroll ($n = 3$) ($p = n/a$)	**0.729** **(0.956)**	0.699 (0.746)	0.706 (1.026)
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.3679$)	0.159 (0.289)	0.182 (0.405)	**0.229** **(0.267)**
Payroll ($n = 10$) ($p = 0.1988$)	0.584 (0.541)	**0.923** **(0.681)**	0.827 (0.649)

(e)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.1251$)	1.103 (1.274)	1.423 (1.484)	**1.564** **(1.377)**
Payroll ($n = 10$) ($p = 0.1095$)	0.683 (0.789)	0.452 (0.550)	**0.947** **(0.800)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	0.500 (2.784)	0.833 (2.566)	**1.000** **(2.646)**
Payroll ($n = 3$) ($p = n/a$)	**1.278** **(1.001)**	0.850 (1.202)	1.235 (1.522)
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.0024$)	-0.105 (1.364)	0.577 (1.566)	**0.587** **(1.647)**
Payroll ($n = 10$) ($p = 0.0665$)	0.698 (0.841)	0.667 (0.709)	**1.163** **(0.939)**

(f)

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.1837$)	0.513 (0.198)	0.595 (0.382)	**0.603** **(0.210)**
Payroll ($n = 10$) ($p = 0.1390$)	0.145 (0.367)	0.121 (0.286)	**0.427** **(0.441)**
Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($n = 3$) ($p = n/a$)	0.483 (0.275)	0.650 (0.606)	**0.733** **(0.643)**
Payroll ($n = 3$) ($p = n/a$)	**1.120** **(1.150)**	1.044 (1.232)	1.058 (1.500)
Last Test			
	I-TC	I-BL	I-NC
Gradebook ($n = 13$) ($p = 0.0078$)	0.177 (0.406)	0.391 (0.539)	**0.438** **(0.411)**
Payroll ($n = 10$) ($p = 0.4423$)	0.937 (0.774)	1.113 (0.837)	**1.174** **(0.959)**

H8. However, in Table 9e and Table 9f, 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 tables) 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 testing mistakes. We discuss possible reasons for this superior robustness in Section 5.3.

5.2.4 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 for the mapping factor?

H10. *There is no difference in the effectiveness of the* three mappings *with the* I-TC *information base when feedback is provided in the context of mistakes.*

H11. *There is no difference in the effectiveness of the* three mappings *with the* I-BL *information base when feedback is provided in the context of mistakes.*

H12. *There is no difference in the effectiveness of the* three mappings *with the* I-NC *information base when feedback is provided in the context of mistakes.*

In the context of at least one incorrect testing decision, Table 10a and Table 10b show significant differences in effectiveness using difference mappings. This trend continued in Table 10c, Table 10d, Table 10e, and Table 10f. These differences were almost always significant at the 0.05 or 0.01 levels, so we reject H10, H11, and H12.

5.3 Discussion

5.3.1 The Information Base Factor

Our results regarding RQ1 showed that the information base factor can make a significant difference in technique effectiveness. This result is in keeping with traditional understanding. 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 greater opportunity to differentiate themselves from each other. However, a surprise was that effectiveness did not always improve as debugging sessions progressed—in the case of Gradebook, the feedback produced by 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 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 aspect of this result is the fact that this information base is the *least computationally expensive* of the three we compared. The second aspect of this result is that the I-NC information base is the information base least like those employed in 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 nontraditional 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 mappings, the information base factor often did not make a significant difference in effectiveness. This occurs *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 test suites.

5.3.2 The Mapping Factor

Turning to RQ2, the role of mapping in the fault localization techniques' performance was quite pronounced. While we found two significant differences at the 0.05 level and three differences at the 0.01 level in RQ1 using both effectiveness measures, our investigation of RQ2 yielded three differences at the 0.05 level and *17* differences at the 0.01 level. These significant differences occurred despite only small distinctions among the way the three mappings were done (i.e., the number of fault likelihood values and the "initial" value).

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

TABLE 10
Isolating the Mapping Factor with the Specified Information Base for Feedback Points in the Context of at Least One Incorrect Testing Decision: (a) Information Base I-TC, Metric Eff-Color, (b) Information Base I-TC, Metric Eff-All, (c) Information Base I-BL, Metric Eff-Color, (d) Information Base I-BL, Metric Eff-All, (e) Information Base I-NC, Metric Eff-Color, and (f) Information Base I-NC, Metric Eff-All

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.2813$)	0.769 (1.013)	0.821 (0.798)	**1.103 (1.274)**
Payroll ($n = 10$) ($p = 0.0536$)	0.000 (0.000)	0.262 (0.416)	**0.683 (0.789)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$) ($p = n/a$)	0.333 (1.528)	0.333 (2.082)	**0.500 (2.784)**
Payroll ($n = 3$) ($p = n/a$)	0.292 (0.505)	0.743 (0.909)	**1.278 (1.001)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.2847$)	-0.308 (0.630)	**0.036 (1.092)**	-0.105 (1.364)
Payroll ($n = 10$) ($p = 0.1128$)	0.149 (0.286)	0.311 (0.587)	**0.698 (0.841)**

(a)

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.0038$)	0.385 (0.215)	0.363 (0.143)	**0.513 (0.198)**
Payroll ($n = 10$) ($p = 0.0275$)	-0.081 (0.364)	0.024 (0.192)	**0.145 (0.367)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$) ($p = n/a$)	0.350 (0.409)	0.350 (0.541)	**0.483 (0.725)**
Payroll ($n = 3$) ($p = n/a$)	-0.081 (0.364)	0.024 (0.192)	**0.145 (0.367)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.1905$)	0.065 (0.385)	0.159 (0.289)	**0.177 (0.406)**
Payroll ($n = 10$) ($p = 0.0622$)	0.704 (0.439)	0.584 (0.541)	**0.938 (0.774)**

(b)

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.0052$)	0.436 (1.013)	0.859 (1.032)	**1.423 (1.484)**
Payroll ($n = 10$) ($p = 0.0204$)	-0.202 (0.458)	0.317 (0.296)	**0.452 (0.550)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$) ($p = n/a$)	0.500 (1.323)	0.500 (1.803)	**0.833 (2.566)**
Payroll ($n = 3$) ($p = n/a$)	0.067 (0.115)	0.492 (0.709)	**0.850 (1.202)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.1985$)	-0.187 (0.673)	0.128 (1.037)	**0.577 (1.566)**
Payroll ($n = 10$) ($p = 0.0013$)	-0.195 (0.465)	0.488 (0.709)	**0.667 (0.709)**

(c)

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.0016$)	0.283 (0.295)	0.386 (0.268)	**0.595 (0.382)**
Payroll ($n = 10$) ($p = 0.2366$)	-0.050 (0.227)	0.087 (0.319)	**0.121 (0.286)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$) ($p = n/a$)	0.350 (0.541)	0.433 (0.404)	**0.650 (0.606)**
Payroll ($n = 3$) ($p = n/a$)	0.347 (0.134)	0.699 (0.746)	**1.044 (1.232)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.0401$)	0.056 (0.392)	0.182 (0.539)	**0.391 (0.539)**
Payroll ($n = 10$) ($p = 0.0004$)	0.403 (0.307)	0.923 (0.681)	**1.113 (0.837)**

(d)

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.0040$)	0.769 (1.013)	0.974 (0.915)	**1.564 (1.377)**
Payroll ($n = 10$) ($p = 0.0084$)	0.051 (0.083)	0.500 (0.398)	**0.947 (0.800)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$) ($p = n/a$)	0.333 (1.528)	0.500 (1.803)	**1.000 (2.646)**
Payroll ($n = 3$) ($p = n/a$)	0.314 (0.543)	0.775 (0.999)	**1.235 (1.522)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.0354$)	-0.285 (0.608)	0.190 (1.007)	**0.587 (1.664)**
Payroll ($n = 10$) ($p = 0.0450$)	0.159 (0.394)	0.698 (0.610)	**1.163 (0.939)**

(e)

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.0008$)	0.385 (0.215)	0.397 (0.132)	**0.603 (0.210)**
Payroll ($n = 10$) ($p = 0.0283$)	0.155 (0.335)	0.267 (0.260)	**0.427 (0.441)**
Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($n = 3$) ($p = n/a$)	0.350 (0.409)	0.433 (0.404)	**1.000 (0.643)**
Payroll ($n = 3$) ($p = n/a$)	0.394 (0.592)	0.706 (1.026)	**1.058 (1.500)**
Last Test			
	M-TC	M-BL	M-NC
Gradebook ($n = 13$) ($p = 0.0115$)	0.076 (0.395)	0.229 (0.267)	**0.438 (0.411)**
Payroll ($n = 10$) ($p = 0.0450$)	0.609 (0.478)	0.827 (0.649)	**1.174 (0.959)**

(f)

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.

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 at times 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 choosing 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.

6 Conclusions and Implications for Future Work

End-user programmers are writing a substantial 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 significant effort to increase the correctness of these often faulty programs. The lack of research in this area is especially evident with respect to supporting the debugging tasks that end-user programmers inevitably must perform.

To help address this need, this paper presents algorithms for three fault localization techniques for end users programming in spreadsheet environments. These techniques are designed to accommodate the differences between professional and end-user programming and have varying costs in order to provide fault localization feedback. We also discussed two separate factors in fault localization techniques that can play important roles in providing fault localization feedback. To investigate the impact of these factors on the effectiveness of fault localization feedback, we conducted an experiment. This experiment indicated that each can have a significant impact on the effectiveness of fault localization techniques. To our knowledge, this work is the first to formally suggest the possible importance of evaluating each factor of a fault localization technique separately.

We believe that the fault localization techniques described in this paper can be extended to other end-user programming paradigms. The information bases of our approaches are coupled with the WYSIWYT testing methodology, which has been shown to be generalizable to other spreadsheet systems, such as Excel [27], to other programming paradigms, such as the dataflow paradigm using Prograph [32], and to the screen transition paradigm using Lyee [10]. Our mappings can be adapted as needed to map these information bases into suitable fault localization feedback in the given programming paradigm. For example, in Prograph [21], fault likelihood calculations would be made for nodes rather than spreadsheet cells. In summary, any paradigm capable of supporting WYSIWYT is capable of supporting our fault localization techniques. Further, our techniques could be implemented in paradigms without the support of WYSIWYT, provided that the environment provides some mechanism for maintaining testing information provided by an oracle such as an end user.

Repeated empirical study is needed, of course, to investigate this experiment's research questions in other settings. Nevertheless, this paper has several implications for future work into bringing interactive fault localization techniques to end-user programmers. First, we found significant differences in the effectiveness of the information bases of our three techniques. These information bases have varying costs associated with providing fault localization feedback, indicating that researchers and developers of end-user programming environments may face varying cost-effectiveness considerations when selecting information bases for their own techniques. For example, designers of future fault localization techniques may find that some expensive information bases may not be as cost-effective as other less expensive information bases. We believe that researchers may benefit from experimenting with various information bases for their own interactive fault localization techniques in order to find the most cost-effective choice, especially if responsiveness is an issue in their settings.

Second, this paper evaluates interactive techniques at the various points throughout debugging in which fault localization feedback occurs, including early in debugging when very little information may be available on which to base feedback. This differs from most research regarding traditional fault localization techniques by considering points other than those of maximal system reasoning potential—when the system has its best (and only) chance of producing correct feedback.

Third, this paper reinforces the need to design and evaluate interactive techniques while considering the possibility of providing feedback in the presence of inaccurate information. The fault localization algorithms presented in this paper have robustness features to attempt to mitigate the impact of inaccurate information. The experiment in this paper also corroborates a growing body of evidence [33], [34], [57] that mistakes are commonly made by end users performing interactive testing and debugging tasks. This contribution, which goes against another traditional assumption in fault localization research —that all information provided to the technique is correct—is important because many techniques leverage testing information to provide feedback. We believe that researchers should design their information bases and mappings with the possibility of mistakes in mind. Researchers should also empirically study their techniques' resiliency to such mistakes to better inform their design decisions. We have recently conducted such follow-up work on our own techniques [44], where we found beneficial adjustments to our mappings and formulated a set of empirically-based recommendations for handling mistakes in fault localization techniques for spreadsheets.

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 better information bases to fault localization techniques. We suggest that researchers design and empirically evaluate various mappings in order to determine the best choice for their own end-user fault localization techniques. Furthermore, for fault localization techniques focusing on professional programmers, it may sometimes be possible to improve the quality of a

technique's feedback with simple modifications to the mapping factors of these techniques rather than by investing resources into developing entirely new information bases.

ACKNOWLEDGMENTS

The authors thank the participants in their experiment. James Reichwein developed the first version of the Blocking Technique [57]. Marc Fisher II developed the first version of the Test Count Technique [25]. Curtis Cook provided advice on our statistical analyses. The authors thank the anonymous reviewers of this paper for comments that improved the content of the work. This work was supported in part by the EUSES Consortium via US National Science Foundation grant ITR-0325273. This paper is a revised and expanded version of a paper presented at the 27th International Conference on Software Engineering, May 2005 [55]. This work was performed at Oregon State University.

REFERENCES

[1] R. Abraham and M. Erwig, "Header and Unit Inference for Spreadsheets through Spatial Analyses," *Proc. IEEE Int'l Symp. Visual Languages and Human-Centric Computing,* pp. 165-172, Sept. 2004.

[2] H. Agrawal and J.R. Horgan, "Dynamic Program Slicing," *Proc. ACM SIGPLAN 1990 Conf. Programming Language Design and Implementation,* pp. 246-256, June 1990.

[3] H. Agrawal, J.R. Horgan, S. London, and W.E. Wong, "Fault Localization Using Execution Slices and Dataflow Tests," *Proc. Sixth IEEE Int'l Symp. Software Reliability Eng.,* pp. 143-151, Oct. 1995.

[4] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi, "A Type System for Statically Detecting Spreadsheet Errors," *Proc. 18th IEEE Int'l Conf. Automated Software Eng.,* pp. 174-183, Oct. 2003.

[5] C.M. Allwood, "Error Detection Processes in Statistical Problem Solving," *Cognitive Science,* vol. 8, no. 4, pp. 413-437, Oct.-Dec. 1984.

[6] T. Antoniu, P.A. Steckler, S. Krishnamurthi, E. Neuwirth, and M. Felleisen, "Validating the Unit Correctness of Spreadsheet Programs," *Proc. 26th Int'l Conf. Software Eng.,* pp. 439-448, May 2004.

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

[8] M. Betts and A.S. Horowitz, "Oops! Audits Find Errors in 49 Out of 54 Spreadsheets," *Computerworld,* p. 47, May 2004.

[9] B. Boehm and V.R. Basili, "Software Defect Reduction Top 10 List," *Computer,* vol. 34, no. 1, pp. 135-137, Jan. 2001.

[10] D. Brown, M. Burnett, G. Rothermel, H. Fujita, and F. Negoro, "Generalizing WYSIWYT Visual Testing to Screen Transition Languages," *Proc. IEEE Symp. Human-Centric Computing, Languages, and Environments,* pp. 203-210, Oct. 2003.

[11] P. Bunus and P. Fritzson, "Semi-Automatic Fault Localization and Behavior Verification for Physical System Simulation Models," *Proc. 18th IEEE Int'l Conf. Automated Software Eng.,* pp. 253-258, Oct. 2003.

[12] M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang, "Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm," *J. Functional Programming,* vol. 11, no. 2, pp. 155-206, Mar. 2001.

[13] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace, "End-User Software Engineering with Assertions in the Spreadsheet Paradigm," *Proc. 25th Int'l Conf. Software Eng.,* pp. 93-103, May 2003.

[14] M. Burnett, C. Cook, and G. Rothermel, "End-User Software Engineering," *Comm. ACM,* vol. 47, no. 9, pp. 53-58, Sept. 2004.

[15] M. Burnett, A. Sheretov, and G. Rothermel, "Scaling Up a 'What You See Is What You Test' Methodology to Spreadsheet Grids," *Proc. IEEE Symp. Visual Languages,* pp. 30-37, Sept. 1999.

[16] T.Y. Chen and Y.Y. Cheung, "On Program Dicing," *Software Maintenance: Research and Practice,* vol. 9, no. 1, pp. 33-46, Jan.-Feb. 1997.

[17] M. Clermont, "Analyzing Large Spreadsheet Programs," *Proc. 10th Working Conf. Reverse Eng.,* pp. 306-315, Nov. 2003.

[18] M. Clermont and R. Mittermeir, "Auditing Large Spreadsheet Programs," *Proc. Int'l Conf. Information Systems Implementation and Modelling,* pp. 87-97, Apr. 2003.

[19] H. Cleve and A. Zeller, "Locating Causes of Program Failures," *Proc. 27th Int'l Conf. Software Eng.,* pp. 342-351, May 2005.

[20] C. Cook, M. Burnett, and D. Boom, "A Bug's Eye View of Immediate Visual Feedback in Direct-Manipulation Programming Systems," *Proc. Empirical Studies of Programmers: Seventh Workshop,* pp. 20-41, Oct. 1997.

[21] P.T. Cox, F.R. Giles, and T. Pietrzykowski, "Prograph: A Step towards Liberating Programming from Textual Conditioning," *Proc. IEEE Workshop Visual Languages,* pp. 150-156, Oct. 1989.

[22] R.A. DeMillo, H. Pan, and E.H. Spafford, "Critical Slicing for Software Fault Localization," *Proc. Int'l Symp. Software Testing and Analysis,* pp. 121-134, Jan. 1996.

[23] E. Duesterwald, R. Gupta, and M.L. Soffa, "Rigorous Data Flow Testing through Output Influences," *Proc. Second Irvine Software Symp.,* pp. 131-145, Mar. 1992.

[24] M. Erwig and M. Burnett, "Adding Apples and Oranges," *Proc. Fourth Int'l Symp. Practical Aspects of Declarative Languages,* pp. 173-191, Jan. 2002.

[25] M. Fisher II, D. Jin, G. Rothermel, and M. Burnett, "Test Reuse in the Spreadsheet Paradigm," *Proc. IEEE Int'l Symp. Software Reliability Eng.,* pp. 257-268, Nov. 2002.

[26] M. Fisher II and G. Rothermel, "The EUSES Spreadsheet Corpus: A Shared Resource for Supporting Experimentation with Spreadsheet Dependability Mechanisms," *Proc. First Workshop End-User Software Eng.,* pp. 47-51, May 2005.

[27] M. Fisher II, G. Rothermel, T. Creelan, and M. Burnett, "Scaling a Dataflow Testing Methodology to the Multiparadigm World of Commercial Spreadsheets," Technical Report TR-UNL-CSE-2005-0003, Univ. of Nebraska-Lincoln, Sept. 2005.

[28] M. Francel and S. Rugaber, "Fault Localization Using Execution Traces," *Proc. ACM 30th Ann. Southeast Regional Conf.,* pp. 69-76, Apr. 1992.

[29] D.S. Hilzenrath, "Finding Errors a Plus, Fannie Says; Mortgage Giant Tries to Soften Effect of $1 Billion in Mistakes," *The Washington Post,* 31 Oct. 2003.

[30] T. Igarashi, J.D. Mackinlay, B.W. Chang, and P.T. Zellweger, "Fluid Visualization of Spreadsheet Structures," *Proc. IEEE Symp. Visual Languages,* pp. 118-125, Sept. 1998.

[31] J.A. Jones, M.J. Harrold, and J. Stasko, "Visualization of Test Information to Assist Fault Localization," *Proc. 24th Int'l Conf. Software Eng.,* pp. 467-477, May 2002.

[32] M. Karam and T. Smedley, "A Testing Methodology for a Dataflow Based Visual Programming Language," *Proc. IEEE Symp. Human-Centric Computing, Languages, and Environments,* pp. 280-287, Sept. 2001.

[33] A.J. Ko and B.A. Myers, "Development and Evaluation of a Model of Programming Errors," *Proc. IEEE Symp. Human-Centric Computing, Languages, and Environments,* pp. 7-14, Oct. 2003.

[34] A.J. Ko and B.A. Myers, "Designing the Whyline: A Debugging Interface for Asking Questions about Program Failures," *Proc. ACM Conf. Human Factors in Computing Systems,* pp. 151-158. Apr. 2004.

[35] B. Korel and J. Laski, "Dynamic Slicing of Computer Programs," *J. Systems and Software,* vol. 13, no. 3, pp. 187-195, Nov. 1990.

[36] 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," *Proc. IEEE Int'l Conf. Software Maintenance,* pp. 72-81, Nov. 2001.

[37] J. Laski and B. Korel, "A Data Flow Oriented Program Testing Strategy," *IEEE Trans. Software Eng.,* vol. 9, no. 3, pp. 347-354, May 1993.

[38] J.R. Lyle and M. Weiser, "Automatic Program Bug Location by Program Slicing," *Proc. Second Int'l Conf. Computers and Applications,* pp. 877-883, June 1987.

[39] R.C. Miller and B.A. Myers, "Outlier Finding: Focusing User Attention on Possible Errors," *Proc. ACM Symp. User Interface Software and Technology,* pp. 81-90, Nov. 2001.

[40] S.C. Ntafos, "On Required Element Testing," *IEEE Trans. Software Eng.*, vol. 10, no. 6, pp. 795-803, Nov. 1984.
[41] H. Pan and E. Spafford, "Toward Automatic Localization of Software Faults," *Proc. 10th Pacific Northwest Software Quality Conf.*, Oct. 1992.
[42] R. Panko, "Finding Spreadsheet Errors: Most Spreadsheet Errors Have Design Flaws that May Lead to Long-Term Miscalculation," *Information Week*, p. 100, May 1995.
[43] R. Panko, "What We Know about Spreadsheet Errors," *J. End User Computing*, pp. 15-21, Spring 1998.
[44] A. Phalgune, C. Kissinger, M. Burnett, C. Cook, L. Beckwith, and J.R. Ruthruff, "Garbage In, Garbage Out? An Empirical Look at Oracle Mistakes by End-User Programmers," *Proc. 2005 IEEE Symp. Visual Languages and Human-Centric Computing*, pp. 45-52, Sept. 2005.
[45] B.C. Pierce and D.N. Turner, "Local Type Inference," *ACM Trans. Programming Languages and Systems*, vol. 22, no. 1, pp. 1-44, Jan. 2000.
[46] B. Pytlik, M. Renieris, S. Krishnamurthi, and S.P. Reiss, "Automated Fault Localization Using Potential Invariants," *Proc. Fifth Int'l Workshop Automated and Algorithmic Debugging*, pp. 273-276, Sept. 2003.
[47] S. Rapps and E.J. Weyuker, "Selected Software Test Data Using Data Flow Information," *IEEE Trans. Software Eng.*, vol. 11, no. 4, pp. 367-375, Apr. 1985.
[48] O. Raz, P. Koopman, and M. Shaw, "Semantic Anomaly Detection on Online Data Sources," *Proc. 24th Int'l Conf. Software Eng.*, pp. 302-312, May 2002.
[49] M. Renieris and S.P. Reiss, "Fault Localization with Nearest Neighbor Queries," *Proc. 18th IEEE Int'l Conf. Automated Software Eng.*, pp. 30-39, Oct. 2003.
[50] G. Robertson, "Officials Red-Faced by $24m Gaffe: Error in Contract Bid Hits Bottom Line of TransAlta Corp.," *Ottawa Citizen*, 5 June 2003.
[51] G. Rothermel, M. Burnett, L. Li, C. Dupuis, and A. Sheretov, "A Methodology for Testing Spreadsheets," *ACM Trans. Software Eng. and Methodology*, vol. 10, no. 1, pp. 110-147, Jan. 2001.
[52] 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," *Proc. 22nd Int'l Conf. Software Eng.*, pp. 230-239, June 2000.
[53] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Prabhakararao, M. Fisher II, and M. Main, "End-User Software Visualizations for Fault Localization," *Proc. ACM Symp. Software Visualization*, pp. 123-132, June 2003.
[54] J.R. Ruthruff and M. Burnett, "Six Challenges in Supporting End-User Debugging," *Proc. First Workshop End-User Software Eng.*, pp. 57-61, May 2005.
[55] J.R. Ruthruff, M. Burnett, and G. Rothermel, "An Empirical Study of Fault Localization for End-User Programmers," *Proc. 27th Int'l Conf. Software Eng.*, pp. 352-361, May 2005.
[56] J.R. Ruthruff, A. Phalgune, L. Beckwith, M. Burnett, and C. Cook, "Rewarding 'Good' Behavior: End-User Debugging and Rewards," *Proc. IEEE Symp. Visual Languages and Human-Centric Computing*, pp. 115-122, Sept. 2004.
[57] J.R. Ruthruff, S. Prabhakararao, J. Reichwein, C. Cook, E. Creswick, and M. Burnett, "Interactive, Visual Fault Localization Support for End-User Programmers," *J. Visual Languages and Computing*, vol. 16, nos. 1-2, pp. 3-40, Feb.-Apr. 2005.
[58] J. Sajaniemi, "Modeling Spreadsheet Audit: A Rigorous Approach to Automatic Visualization," *J. Visual Languages and Computing*, vol. 11, no. 1, pp. 49-82, Feb. 2000.
[59] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the Numbers of End Users and End User Programmers," *Proc. 2005 IEEE Symp. Visual Languages and Human-Centric Computing*, pp. 207-214, Sept. 2005.
[60] S. Siegel and N.J. Castellan Jr., *Non-Parametric Statistics for the Behavioral Sciences.* Boston: McGraw-Hill, 1998.
[61] D.H. Stamatis, *Failure Mode Effect Analysis: FMEA from Theory to Execution*, second ed. ASQ Quality Press, June 2003.
[62] F. Tip, "A Survey of Program Slicing Techniques," *J. Programming Languages*, vol. 3, no. 3, pp. 121-189, 1995.
[63] J.M. Voas, "Software Testability Measurement for Assertion Placement and Fault Localization," *Proc. Int'l Workshop Automated and Algorithmic Debugging*, pp. 133-144, 1995.
[64] E. Wagner and H. Lieberman, "An End-User Tool for e-Commerce Debugging," *Proc. Int'l Conf. Intelligent User Interfaces*, pp. 331, Jan. 2003.
[65] E.J. Wagner and H. Lieberman, "Supporting User Hypotheses in Problem Diagnosis on the Web and Elsewhere," *Proc. Int'l Conf. Intelligent User Interfaces*, pp. 30-37, Jan. 2004.
[66] M. Weiser, "Program Slicing," *IEEE Trans. Software Eng.*, vol. 10, no. 4, pp. 352-357, July 1984.
[67] 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," *Proc. ACM Conf. Human Factors in Computing Systems*, pp. 305-312, Apr. 2003.
[68] C. Wohlin, P. Runeson, M. Host, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering.* Boston: Kluwer Academic, 2000.

Joseph R. Ruthruff is currently a PhD student in computer science at the University of Nebraska-Lincoln. In 2004, he received the MS degree in computer science from Oregon State University. In 2002, he received the honors BS degree in computer science, magna cum laude, with a minor in mathematics from Oregon State University. His research interests lie in the analysis and verification of software systems and in the human-computer interaction issues arising in those systems. He is a student member of the ACM, the ACM SIGSOFT, the IEEE, and the IEEE Computer Society. He is a member of the Upsilon Pi Epsilon Honors Society.

Margaret Burnett received the BA degree in mathematics from Miami University, Oxford, Ohio, and the MS and PhD degrees in computer science from the University of Kansas, Lawrence. She worked for several years for large and small companies before beginning her academic career as an assistant professor at Michigan Technological University in 1991. She is currently a professor of computer science at Oregon State University. She is also project director of the EUSES Consortium (End Users Shaping Effective Software), a national research consortium investigating ways to support end-user software engineering. She has been a recipient of the US National Science Foundation's Young Investigator Award, of Oregon State University's Elizabeth P. Ritchie Distinguished Professor Award, and of Oregon State University's Research Collaboration Award (College of Engineering). She is currently on the steering committees for the IEEE Symposium on Visual Languages and Human Centric Computing and for the ACM Symposium on Software Visualization. She is a senior member of the IEEE and a member of the ACM and the IEEE Computer Society.

Gregg Rothermel received the PhD degree in computer science from Clemson University, the MS degree in computer science from the State University of New York at Albany, and the BA degree in philosophy from Reed College. He is currently a professor and the Jensen Chair of Software Engineering in the Department of Computer Science and Engineering at the University of Nebraska-Lincoln. His research interests include software engineering and program analysis, with emphases on the application of program analysis techniques to problems in software maintenance and testing and on empirical studies. He is program cochair for ICSE 2007 and has previously served as associate editor-in-chief for the *IEEE Transactions on Software Engineering*, program chair for ISSTA 2004, and the chair of the steering committee for the International Conference on Software Maintenance. He is a member of the editorial boards for the *Empirical Software Engineering Journal* and the *Software Quality Journal*. He has served as a member of numerous program committees. He is a member of the IEEE and the ACM.