Technical Report 04-60-08 — School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR, USA, 21 June 2004

The Impact of Two Orthogonal Factors in Interactive Fault Localization

Joseph R. Ruthruff, Margaret Burnett, and Gregg Rothermel
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR, 97331
[ruthruff, burnett, grother]@cs.orst.edu

ABSTRACT

End users develop more software than any other group of programmers, using software authoring devices such as e-mail filtering editors, by-demonstration macro builders, and spreadsheet environments. Despite this, there has been only a little research on finding ways to help these programmers with the dependability of the software they create. We have been working to address this problem in several ways, one of which includes supporting end-user debugging activities through fault localization techniques. Our preliminary studies have shown that some fault localization techniques can provide end users with effective feedback; however, these studies provide no guidance into the factors that impact technique effectiveness. This paper presents the results of an empirical study that examines the impact of two orthogonal factors on the effectiveness of fault localization techniques. Our results have implications for fault localization for end users, and highlight the importance of separately evaluating each factor in a fault localization technique.

Keywords

fault localization, debugging, end-user software engineering, end-user programming

1. INTRODUCTION

A quiet revolution is occurring in the world of software. Not too long ago, most software was developed primarily by professional programmers, a large portion of whom had reason to be interested in and understand software engineering theory and practice. Today, however, end users write far more software than professional programmers. It is estimated that, by next year, 55 million end users, as compared to only 2.75 million professional programmers [6], will be creating software applications such as multimedia simulations, dynamic web pages, e-mail filtering rules, and spreadsheets.

Is adequate support being provided to these end users? The evidence suggests that it is not. Boehm and Basili observe that 40-50% of the software created by end-user programmers contains non-trivial faults [7]. These faults can be serious, costing millions of dollars in some cases [13, 23].

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

Software engineering researchers have invested considerable effort into bringing fault localization strategies to professional programmers (e.g., [1, 9, 14, 16, 18, 21, 25, 27, 33]). However, significant differences exist between professional and end-user software development, and these differences have ramifications for fault localization strategies.

A first class of differences is that, unlike professional programmers, end users rarely have knowledge of software engineering theory and practice, and are unlikely to take the time to acquire it. This impacts fault localization strategies because, traditionally, such strategies often require at least partial knowledge of such theory to either properly employ the strategy or understand its feedback. For example, critical slicing [14] uses mutation-based testing, and end users would be unlikely to understand—and therefore to trust—the resulting feedback. (As [12] explains, understanding is critical to trust, which in turn is critical to users actually believing a system's output and acting upon it.)

A second class of differences pertains to the manner of interaction between the software developer and the programming environment. End-user programming environments are usually modeless and interactive: users incrementally experiment with their software and see how the results seem to be working out after every change, using devices such as the automatic recalculation feature of spreadsheets. Techniques that perform batch processing (e.g., [1, 18]) are therefore at best unsuited, and at worst incompatible, with these types of interactive environments.

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

A fourth class of differences pertains to a common assumption in software engineering devices created for professional programmers: that the accuracy of the information (such as testing information) provided to the devices is reliable. Evidence shows that end users often make mistakes when performing interactive testing and debugging [24]. (Professional programmers are not perfect either, of course, but there is

reason to hope that their own understanding of testing and their institution's testing processes render them less error-prone than end users.) Unfortunately, many fault localization techniques (e.g., [21]) cannot operate in the presence of such unreliable information.

We have been working to bring fault localization support to end users, in ways that accommodate the foregoing considerations, as part of our *end-user software engineering* research. Our previous work [24, 29] shows that our fault localization support can effectively pinpoint program points containing faults, and can help end users find faults by guiding them into more effective debugging strategies. This work suggested the possibility that there may be multiple, independent *factors* of fault localization techniques that impact technique effectiveness in the realm of end-user software development. An understanding of these factors will be necessary in order for future designers of end-user fault localization techniques to build upon principled design choices instead of ad hoc guesses.

We have therefore designed and conducted a controlled experiment analyzing two factors involved in a fault localization technique's ability to pinpoint the location of faults. The factors we consider, information base and mapping, pertain to the information maintained by a technique to provide fault localization feedback and how that technique maps the information into such feedback, respectively.

This work makes three primary contributions. First, we empirically investigate and provide data on the importance of the two factors of information base and mapping. Second, we provide data on three specific fault localization information bases and three mappings. Third, we provide insights into the way that fault localization effectiveness needs to be measured, so as to inform others' evaluation work in end-user fault localization techniques.

2. BACKGROUND: WYSIWYT

Our fault localization techniques are prototyped in the spreadsheet paradigm, in conjunction with our "What You See Is What You Test" (WYSIWYT) testing methodology [28], so we briefly describe that methodology here. Figure 1 presents an example of WYSIWYT in Forms/3 [10], a spreadsheet language that utilizes "free-floating" cells in addition to traditional spreadsheet grids.[1] The underlying assumption behind the WYSIWYT testing methodology is that, as a user incrementally develops a spreadsheet, he or she is also testing incrementally. Because the intended audience is end users, all communication about testing is performed through visual devices. In WYSIWYT, untested cells that have non-constant formulas are given a red border (light gray in this paper), indicating that the cell is untested. (Cells whose formulas are simply constants do not participate in WYSIWYT devices, since the assumption is that they do not need to be tested.) The borders of such cells remain red until they become more "tested".

In order for cells to become more tested, tests must occur. Tests can occur at *any* time—intermingled with editing formulas, adding new formulas, and so on. The process is as follows. Whenever a user notices a correct value, he or she can place a checkmark ($\surd$) in the decision box at the corner of the cell he or she observes to be correct: this *testing decision* constitutes a successful "test". Such checkmarks increase the "testedness" of a cell, which is reflected by adding more blue to the cell's border (more black in this paper). 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. WYSIWYT "testedness" colors reflect the use of a dataflow test adequacy criterion that measures the interrelationships in the source code that have been covered by the users' tests. (Details of this criterion are given in [28].)

In addition to providing feedback at the cell level, WYSIWYT gives the user feedback about testedness at two other granularities. A percent testedness indicator provides testedness feedback at the program granularity. Testedness feedback is also available at a finer granularity through dataflow arrows. In addition to displaying dataflow relationships at the cell level (in Figure 1, the user has triggered dataflow arrows for the `Course_Avg` cell), arrows can be shown at the subexpression level (not shown in Figure 1). The system also provides testedness feedback through an intelligent explanation system [36], implemented via "on-demand" tooltips.

3. CHARACTERISTICS OF INTERACTIVE FAULT LOCALIZATION

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

In our prototype, which follows the spreadsheet paradigm (although our fault localization approach is generalizable to other paradigms, as discussed in [30]), WYSIWYT information serves as a springboard for fault localization: instead

[1] WYSIWYT has also been extended to the dataflow [19] and screen transition [8] paradigms.

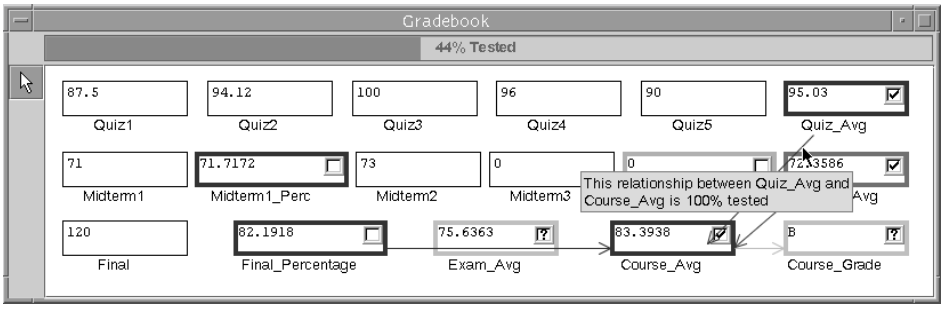

Figure 1: An example of WYSIWYT in the Forms/3 language.

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," as in Figure 2.

These X-marks trigger a *fault likelihood* calculation for each cell that might have contributed to the failure. Fault likelihood, updated for each appropriate cell after any testing decision or formula edit, is represented by visually highlighting the interior of suspect cells in shades of red (gray in this paper). This serves our first goal of reducing the user's search space. As the fault likelihood of a cell grows, the suspect cell is highlighted in increasingly darker shades of red (gray). The darkest cells are estimated to be the most likely to contain the fault, and are the best candidates for the user to consider in trying to debug; this serves our second goal of helping end users decide how to prioritize their search.

We have previously developed three techniques for realizing this support [29], which we briefly summarize here.

- *Test Count.* When a user places an X-mark, this technique calculates fault likelihood from the number of passed and failed tests for each cell, and highlights appropriate cells based on these calculations.
- *Blocking.* This technique also uses the number of passed and failed tests to calculate the fault likelihood of cells after an X-mark is placed. However, a test t_1 can "block" another test t_2 from affecting the fault likelihood of a cell c if all dataflow from c to the cell in which the t_2 testing decision was made goes through the cell in which the t_1 testing decision was made.
- *Nearest Consumers.* Rather than using passed and failed tests directly, when a user places an X-mark, this technique visually highlights a cell c based on (1) the average fault likelihood of the cells C' directly consuming c's value, and (2) the current testing decisions (X- and $\surd$ marks) on both c and C'.

3.1 Two Factors in Fault Localization

Our previous empirical work [24, 29] showed that these three fault localization techniques can provide effective feedback that is useful to end users. However, the work also suggested that more than one independent factor may be involved in determining technique effectiveness. This paper investigates this possibility.

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

- *Information Base:* The *information* used by a technique to calculate fault likelihood. To abstract away implementation details such as data structures or algorithmic details, we use this term to refer to only the type of information used and the circumstances under which it is maintained.
- *Mapping:* How a technique maps the information base into fault localization feedback.

For example, Tarantula [18], a fault localization technique for traditional programming languages, uses a set of passed and failed tests as its information base. As its mapping, two mathematical formulas calculate (1) a color representing each statement's participation in testing, and (2) the technique's confidence in the correctness of each color.

3.1.1 Information Bases

Each of our three techniques has a unique base of information used to achieve the behavior described earlier.

- *Test Count* (`I-TC`). The information base of the Test Count technique tracks, for each cell c, the set of passed and failed tests that dynamically execute c. This information base is similar to that used in Tarantula [18]. The size of this information base grows with respect to both program 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 passing and failed tests for each cell. However, to achieve the previously mentioned "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 cell. (See [29] for these specific circumstances.) This information base is similar to that used in dicing [21]. Because of the necessary overhead to track this information, `I-BL` is more computationally expensive than `I-TC`.
- *Nearest Consumers* (`I-NC`). This base tracks only (1) the current fault likelihood of every cell in the program, and (2) the current testing decision for each cell *affected by the current test case* so as to adjust for

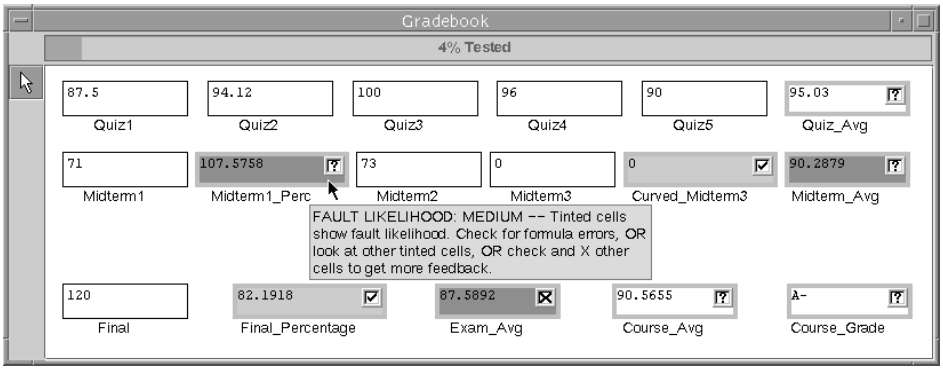

Figure 2: The user notices an incorrect value in `Exam_Avg`—the value is obviously too high—and places an X-mark in the cell's decision box. All cells that could have dynamically contributed to this incorrect value have been highlighted in shades of red (gray in this paper), with darker shades corresponding to increased fault likelihood. (This example uses the Nearest Consumers technique.)

trends in testing decisions. Since each of these components requires only constant space, the information base grows with respect to program size only. This "discount" information base is more modest than most other fault localization techniques in the literature.

Note that, because the context is *interactive* fault localization, each of these information bases is immediately updated whenever any action is taken by a user that affects the contents of the base, potentially interfering with the environment's interactivity. One reason to compare these information bases then is to learn whether it is possible for a modest version such as `I-NC` to compete in effectiveness with the other two more extensive information bases.

3.1.2 Mappings

In light of the unreliable nature of end-user testing, each of our mappings incorporate a "robustness" feature (labeled "Property 1" in [29]) 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 was deemed to be integral to the design of any fault localization technique for end users to ensure that, if there had been even one correctly placed X-mark involving a cell, ensuing incorrect testing decisions could not completely remove the cell from a user's search space.

Many mappings are possible, but it would not be feasible to compare them all. Further, doing so is not warranted until we learn whether mapping is important to a technique's effectiveness. Thus, we use the three particular mappings from our past empirical work as a vehicle for investigating the importance of mapping as an independent factor.

- *Test Count* (`M-TC`). The Test Count technique's mapping ensures that the fault likelihood of a cell c is directly proportional to the number of c's failed tests, and inversely proportional to the number of c's passed tests. It has the characteristic of mapping information bases to four fault likelihood values, and begins by assigning c the lowest fault likelihood if it contributes to a single failure (X-mark), thereby allowing fault likelihood to build with further failures.
- *Blocking* (`M-BL`). This mapping is similar to `M-TC`, except that it supports the blocking features of `I-BL`. It also 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 based on trends in current testing decisions (readers are referred to [29] for details). It also supports five values, and begins by assigning c the third fault likelihood value so as to allow viability of increasing or decreasing fault likelihood values as the cell's neighbors' fault likelihoods increase and decrease.

Although these mappings come from different techniques, all have two characteristics: (1) some number of fault likelihood values, and (2) an "initial" value. Later in this paper, when we refer to applying a mapping to an information base, we refer to applying these two characteristics. Note that we do not attempt to tease apart the influences these characteristics might have, but simply consider them an atomic unit to learn whether mapping in general can have an impact.

3.2 Evaluation of Interactive Techniques

Many traditional fault localization techniques report feedback only at the end of a batch processing of information. This point of maximal system reasoning potential—when the system has its best (and only) chance of producing correct feedback—is therefore the appropriate point to measure these techniques.

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

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

- *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 more information than the first X-mark, so measuring at this point helps to gauge effectiveness trends over time. For the same reason, we measured at the third X-marks, fourth X-marks, and so on, but the participants in our experiment kept their debugging very incremental, which caused almost all debugging sessions to consist of two or fewer X-marks. Thus, this paper does no analysis beyond the second X-mark.
- *Last Test.* When users find the cause of the 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 note that the need to evaluate at such points is not specific to our particular experiment. Rather, *any* interactive fault localization technique must be evaluated on the basis of multiple feedback points because without doing so, the experiment would be omitting most of the data reported by the technique.

4. EXPERIMENT

How important is each factor of a fault localization technique to the effectiveness of that technique? Given the evidence of mistakes in interactive end-user testing, how well do different information bases handle unreliable information?

To shed some insight into these questions, we conducted an experiment investigating the following research questions:

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

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

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

One reason to investigate RQ1 is that the three information bases are markedly different in cost. If there is no difference in their effectiveness, we can safely choose the least expensive; and if there is a difference in their effectiveness, that information tells us which one to pursue.

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

Our third research question was inspired by the unreliability in interactive end-user testing, which we have seen in previous empirical work [24]. We focus this question on information bases because it is specifically the information base that is corrupted by such mistakes.

4.1 Design

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

A second possible methodology was to follow a classic test suite generation approach: generate hypothetical test suites according to some criterion, and select (randomly or according to other criteria) from these test suites to simulate end users' testing actions. We could then run each selected test suite under each technique, and compare effectiveness. This methodology features the tight controls we sought, but the test suites could not be tied to our ultimate users, and may not be representative of real end-user testing actions.

We chose instead to adopt a third methodology that draws upon advantages from both of the above, while avoiding their drawbacks. We obtained actual testing actions from real end users, and then uniformly applied these actions across all mapping and information base combinations. The test suites, as defined by the testing actions that the end users performed, were the subjects of our experiment. These test suites were sampled according to the methods outlined in Section 3.2.

4.2 Participants

To obtain the necessary test suites as subjects, we recruited 20 business major participants. We chose business majors because spreadsheets are commonly developed in business environments. We required participants to be experienced with spreadsheets because we did not want the learning of basic spreadsheet functionality to be a variable.

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 the earlier empirical work [24, 29], we chose to incorporate the `I-TC` information base with the `M-BL` mapping into the environment described in Sections 2 and 3. We then applied the testing actions collected using this technique across all information base and mapping combinations.

4.3 Materials

The experiment utilized two spreadsheet programs, `Gradebook` and `Payroll` (shown in Figures 2 and 3, respectively). To make our programs representative of real end-user spreadsheets, `Gradebook` was derived from an Excel spreadsheet of an (end-user) instructor, which we ported into an equivalent Forms/3 spreadsheet. `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 real end users. To obtain these faults, we provided three separate end users with the following: (1) a "template" spreadsheet for each program with cells and cell names, but no cell formulas; and (2) a description of how each spreadsheet 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 according to Allwood's classification system [3]. Under Allwood's system, mechanical faults include simple typographical errors or wrong cell ref-

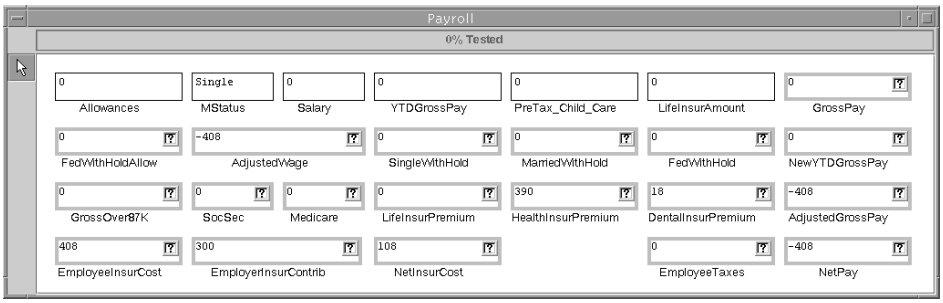

Figure 3: The `Payroll` task.

erences. Logical faults are mistakes in reasoning and are more difficult to correct than mechanical faults. An omission fault is information that has never been entered into a cell formula, and is the most difficult to correct [3]. 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. `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.

4.4 Procedure

After completing a background questionnaire, participants were given a brief tutorial to familiarize them with the environment. 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.

After the tutorial, participants were given the `Gradebook` and `Payroll` programs (tasks) with instructions to test and correct any errors found in the programs. The 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.

4.5 Measures For Evaluation

For this experiment, 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. We measured effectiveness as 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 faulty cells from the average fault likelihood of the colored correct cells. (Subtraction is used instead of calculating a ratio because the color choices form an ordinal, not a ratio, scale.) Positive effectiveness is preferable, and a greater effectiveness means a better distinction between faulty and non-faulty cells. Our metric was designed to consider colored cells only, because those are the cells that are intended to draw the user's attention. This effectiveness metric is the dependent variable of our experiment, and is employed at every feedback point outlined in Section 3.2.

4.6 Threats to Validity

Any controlled experiment is subject to threats to validity, and these must be considered in order to assess the meaning and impact of results. (See [37] for a general discussion of validity evaluation and a threats classification.)

4.6.1 Threats to Internal Validity

The specific types of faults seeded in a program can affect fault localization results. To reduce this threat, as described in Section 4.3, we selected faults according to Allwood's classification scheme [3] to ensure that different types of faults were included.

As mentioned in Section 4.2, in order to apply the same test suites uniformly across all techniques, we had to obtain suites using a single information base and mapping, and we chose the `I-TC` information base and `M-BL` mapping. It is possible that the specific actions taken by participants, in response to fault localization feedback, would have varied had a different information base or mapping been chosen. This tradeoff was necessary in order to obtain uniform test suites, as we have already explained.

4.6.2 Threats to Construct Validity

The effectiveness metric considers only the cells *colored* by a technique. This ignores search space size (i.e., the number of colored cells) because cells that are not visually highlighted are not considered by the effectiveness metric; including such cells would reward techniques that do not highlight colored cells. In this experiment, however, search space size was not an issue, since all combinations of information bases and mappings create an identical search space for any testing situation—only the fault likelihood values of specific cells differ across information bases and mappings.

Still, it is possible that other metrics could better measure how well techniques provide fault localization feedback. To reduce this threat, we cross-checked our results with the effectiveness metric used in [29] (which does consider the number of non-highlighted cells) and found the same trends and results as we report in Section 5. (We do not report these results due to space constraints.)

4.6.3 Threats to External Validity

Program representativeness is an issue facing our experiment. If the programs used in our experiment did not mimic those that real end users create, our results may not generalize. To reduce this threat, we selected "real-world" spreadsheets from a real end-user instructor and a real payroll description. 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 4.3. Finally, our experiment was conducted in the Forms/3 research language [10]. However, end users may debug differently in a different language. All of these external validity concerns can be addressed only through repeated studies, using different programs, faults, and languages.

5. RESULTS

5.1 RQ1: Information Base

To investigate the different information bases' impact on technique effectiveness in isolation from the mapping factor, we compared the information bases' effectiveness three times, once under each mapping. The comparisons were done at each feedback point described in Section 3.2.

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.

Tables 1(a)–1(c) show the results. We used the Friedman test to statistically analyze the data. This test is an alternative to the repeated measures ANOVA when the assumption of normality or equality is not met. (We did not run Friedman tests on the Second X-mark data due to the small sample sizes.) Table 1(a) shows no significant differences in information base effectiveness at the 0.05 level under use of the `M-TC` mapping, so H1 cannot be rejected. However, Table 1(b) shows marginal significance (0.10) at the First X-mark for the `Payroll` task and 0.01 significance by the Last Test. Differences were even more pronounced in Table 1(c). Therefore, we reject H2 and H3.

As the table shows, the `I-NC` information base, which is the basis of the inexpensive Nearest Consumers technique [29], showed the highest average effectiveness at almost every point measured. Implications of this will be discussed in Section 6.

5.2 RQ2: Mapping

How important is mapping alone to technique effectiveness? The table in Section 5.1 above is 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.

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

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

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

As Tables 2(a)–2(c) show, there were significant differences in technique effectiveness among the different mappings. The 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.

5.3 RQ3: Robustness

As our first step, to investigate the pervasiveness of mistakes, we counted the number of incorrect testing decisions made in each subject (end-user test suite). In the context of our environment, this is either a WYSIWYT checkmark ($\surd$), signifying a correct value, placed in a cell that really has an incorrect value, or an X-mark, signifying an incorrect value (a failure), placed in a cell that really has a correct value. In the `Gradebook` task, 8.99% of the checkmarks and 5.95% of the X-marks were incorrect. This trend continued in `Payroll`, where 20.62% of the checkmarks and 3.33% of the X-marks were incorrect.

Given that such mistakes corrupt information bases, how did these mistakes impact an information base's effect on technique effectiveness? To investigate this, we measured effectiveness at each First X-mark, Second X-mark, and Last Test *that was in the context of at least one incorrect testing decision.* We isolated information bases using the same procedure as Section 5.1.

H7: There is no difference in the effectiveness of the three information bases when feedback is provided in the context of mistakes.

Although we ran comparisons under all three mappings, due to space constraints, we show only those measurements under the `M-NC` mapping, since `M-NC` was the superior mapping in Section 5.2. (The information bases under the other mappings show the same general trend.)

As Table 3 shows, 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 `H7`. More

First X-mark	I-TC	I-BL	I-NC
Gradebook ($p = 0.8948$) ($n = 18$)	**0.39 (0.50)**	0.26 (0.62)	**0.39 (0.50)**
Payroll ($p = 0.1211$) ($n = 13$)	0.00 (0.00)	-0.17 (0.40)	**0.04 (0.08)**

Second X-mark	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.00 (1.00)	**0.16 (0.76)**	0.00 (1.00)
Payroll ($p = n/a$) ($n = 5$)	0.15 (0.46)	0.01 (0.12)	**0.18 (0.49)**

Last Test	I-TC	I-BL	I-NC
Gradebook ($p = 0.4389$) ($n = 18$)	-0.06 (0.54)	**0.00 (0.49)**	-0.04 (0.51)
Payroll ($p = 0.0608$) ($n = 13$)	0.13 (0.28)	-0.12 (0.48)	**0.21 (0.50)**

(a) Information bases with `M-TC`

First X-mark	I-TC	I-BL	I-NC
Gradebook ($p = 0.7165$) ($n = 18$)	0.83 (0.84)	0.86 (1.00)	**0.94 (0.92)**
Payroll ($p = 0.1000$) ($n = 13$)	0.29 (0.42)	0.35 (0.33)	**0.49 (0.40)**

Second X-mark	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.33 (2.08)	**0.50 (1.80)**	**0.50 (1.80)**
Payroll ($p = n/a$) ($n = 5$)	0.37 (0.95)	0.36 (0.60)	**0.56 (0.83)**

Last Test	I-TC	I-BL	I-NC
Gradebook ($p = 0.4464$) ($n = 18$)	0.26 (1.12)	0.33 (1.06)	**0.38 (1.03)**
Payroll ($p = 0.0128$) ($n = 13$)	0.30 (0.60)	0.60 (0.53)	**0.77 (0.66)**

(b) Information bases with `M-BL`

First X-mark	I-TC	I-BL	I-NC
Gradebook ($p = 0.0923$) ($n = 18$)	1.14 (1.32)	1.39 (1.37)	**1.50 (1.39)**
Payroll ($p = 0.0695$) ($n = 13$)	0.69 (0.78)	0.54 (0.65)	**0.93 (0.80)**

Second X-mark	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.50 (2.78)	0.83 (2.57)	**1.00 (2.65)**
Payroll ($p = n/a$) ($n = 5$)	0.69 (1.17)	0.54 (1.05)	**0.94 (1.22)**

Last Test	I-TC	I-BL	I-NC
Gradebook ($p = 0.0022$) ($n = 18$)	0.27 (1.52)	0.76 (1.48)	**0.79 (1.61)**
Payroll ($p = 0.0199$) ($n = 13$)	0.65 (0.86)	0.83 (0.88)	**1.27 (0.95)**

(c) Information bases with `M-NC`

Table 1: Isolating the information base factor. The mean (standard deviation) effectiveness comparing the three information bases with the same mapping are shown. The information base with the greatest average effectiveness is shown in bold. The "p" denotes p-values of the Friedman tests, and "n" denotes the number of subjects measured at each point.

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0031$) ($n = 18$)	0.39 (0.50)	0.83 (0.84)	**1.15 (1.32)**
Payroll ($p = 0.0060$) ($n = 13$)	0.00 (0.00)	0.29 (0.42)	**0.69 (0.78)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.00 (1.00)	0.33 (2.08)	**0.50 (2.78)**
Payroll ($p = n/a$) ($n = 5$)	0.15 (0.46)	0.37 (0.95)	**0.69 (1.17)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.1180$) ($n = 18$)	-0.06 (0.54)	0.26 (1.12)	**0.27 (1.52)**
Payroll ($p = 0.1220$) ($n = 13$)	0.13 (0.28)	0.30 (0.60)	**0.65 (0.86)**

(a) Mappings with I-TC

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0004$) ($n = 18$)	0.26 (0.62)	0.86 (1.00)	**1.39 (1.37)**
Payroll ($p = 0.0016$) ($n = 13$)	-0.17 (0.40)	0.35 (0.33)	**0.54 (0.65)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.16 (0.76)	0.50 (1.80)	**0.83 (2.57)**
Payroll ($p = n/a$) ($n = 5$)	0.01 (0.12)	0.36 (0.60)	**0.54 (1.05)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0424$) ($n = 18$)	0.00 (0.49)	0.33 (1.06)	**0.76 (1.48)**
Payroll ($p = 0.0001$) ($n = 13$)	-0.12 (0.48)	0.60 (0.53)	**0.83 (0.88)**

(b) Mappings with I-BL

First X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0001$) ($n = 18$)	0.39 (0.50)	0.94 (0.92)	**1.50 (1.39)**
Payroll ($p = 0.0005$) ($n = 13$)	0.04 (0.08)	0.49 (0.40)	**0.93 (0.80)**

Second X-mark			
	M-TC	M-BL	M-NC
Gradebook ($p = n/a$) ($n = 3$)	0.00 (1.00)	0.50 (1.80)	**1.00 (2.65)**
Payroll ($p = n/a$) ($n = 5$)	0.18 (0.49)	0.56 (0.83)	**0.94 (1.22)**

Last Test			
	M-TC	M-BL	M-NC
Gradebook ($p = 0.0045$) ($n = 18$)	-0.04 (0.51)	0.38 (1.03)	**0.79 (1.61)**
Payroll ($p = 0.0036$) ($n = 13$)	0.21 (0.50)	0.77 (0.66)	**1.27 (0.95)**

(c) Mappings with I-NC

Table 2: The mean (standard deviation) effectiveness comparing the mappings under each information base.

important though, perhaps due to the help of our robustness feature, all three information bases were able to provide effective feedback (indicated by positive values in the table) in most cases, even in the presence of user mistakes. (As one would expect, the mistakes did appear 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.)

The I-NC information base usually showed the highest average effectiveness in the context of these testing mistakes. We discuss possible reasons for this superior robustness in Section 6.

First X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.1251$) ($n = 13$)	1.10 (1.27)	1.42 (1.48)	**1.56 (1.38)**
Payroll ($p = 0.1095$) ($n = 10$)	0.68 (0.79)	0.45 (0.55)	**0.95 (0.80)**

Second X-mark			
	I-TC	I-BL	I-NC
Gradebook ($p = n/a$) ($n = 3$)	0.50 (2.78)	0.83 (2.57)	**1.00 (2.65)**
Payroll ($p = n/a$) ($n = 3$)	**1.28 (1.00)**	0.85 (1.20)	1.24 (1.52)

Last Test			
	I-TC	I-BL	I-NC
Gradebook ($p = 0.0024$) ($n = 13$)	-0.11 (1.36)	0.58 (1.57)	**0.59 (1.65)**
Payroll ($p = 0.0665$) ($n = 10$)	0.70 (0.84)	0.67 (0.71)	**1.16 (0.94)**

Table 3: The effectiveness of the information bases with mapping M-NC for feedback points that were in the context of at least one incorrect testing decision.

6. DISCUSSION

The results regarding RQ1 showed that the information base factor can make a significant difference in technique effectiveness. This result is in keeping with tradition. We also found that the information bases' differences in effectiveness were most pronounced at the end of debugging sessions, most likely due to the increased testing information available at the end of a session, allowing the techniques a greater opportunity to differentiate themselves from each other. However, note that effectiveness did not always get better as debugging sessions progressed—in the case of Gradebook, all nine information bases and mappings consistently got worse. We believe this may relate to the mistakes the users made in their testing, a point we will return to shortly.

Another surprise was the superior effectiveness of the I-NC information base. The first surprising element in this result is the fact that this information base is the *least computationally expensive* of the three we compared. The second surprising element in this result is that the I-NC information base is the information base least like those of many traditional fault localization techniques, which tend to use counts of passed and failed tests (as does I-TC) or dicing-like approaches (as does I-BL) to generate feedback.

Turning to RQ2, the role of mapping in the fault localization techniques' performance was quite pronounced. The significant differences occurred despite only small distinctions among the three mappings (described in Section 3.1.2).

This result has two implications. First, regarding the design of fault localization techniques, our results suggest that because mapping plays such a critical role, great care should be exercised in selecting what mapping to include in a fault localization technique. The second concerns the evaluation of fault localization techniques. Since the information base and mapping factors had significant, *independent* roles in the techniques' effectiveness, our results suggest that evaluating each factor separately is necessary in order to obtain accurate information as to the effectiveness of a fault localization technique.

One result of RQ3 was that the end-user test suites contained quite a few mistakes, which corroborates previous findings [24]. The effects of these mistakes were far-reaching too—although they numbered just under 25% of the tests, they affected 74% of the debugging sessions, underlining the seriousness with which this issue should be regarded. We hypothesize that these mistakes caused the drop in the effectiveness of fault localization feedback in `Gradebook` as debugging sessions progressed, while the larger size of `Payroll` may have diluted the impact of these mistakes.

Given its "modest" nature, we were surprised that the `I-NC` information base continued to generally provide the most effective fault localization feedback even in the presence of mistakes. The reason for this may be tied to the non-traditional nature of the `I-NC` information base. Unlike `I-TC` and `I-BL`, `I-NC` does not contain sets of passed and failed tests, which our results indicate are often corrupted by end-user mistakes. Instead, `I-NC` reasons using only (1) testing decisions made about the current test case, combined with (2) previous fault likelihood calculations, including trends observed in the relationships of X-marks to checkmarks. This emphasis on recent testing decisions allows the effects of past mistakes to recede more quickly than is possible in the other two information bases.

7. RELATED WORK

Most fault localization research has been based on program slicing [35] and dicing [21] techniques; see [32] for a survey of this work. Our fault localization techniques draw from information gleaned via program slicing, and make use of that information using heuristics inspired by dicing.

There has been a great deal of work on fault localization strategies for professional programmers (e.g., [1, 9, 14, 16, 18, 21, 25, 27, 33]). For example, Agrawal et al. [1] present a technique, implemented as a tool called χslice, for locating faults in traditional programming languages using execution traces from tests. This technique is based on displaying dices of the program relative to one failing test and a set of passing tests. Jones et al. [18] 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 [16] 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 [27] 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.

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 [5] present a method of "fault tracing" for spreadsheets 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 [34] is a software agent that assists e-commerce debugging. Ko and Myers [20] 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. 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.

These is other work that can help end users find faults. S2 [31] provides a visual auditing feature in Excel 7.0: similar groups of cells are recognized and shaded based upon formula similarity, and are then connected with arrows to show dataflow. Igarashi et al. [17] 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 [4, 15] and types [2].

There has also been work to help end users detect failures. Statistical outlier finding [22] and anomaly detection [26] use statistical analysis and interactive techniques to direct end-user programmers' attention to potentially problematic areas during automation tasks. Finally, the assertions approach in Forms/3 automatically detects failures in spreadsheet cells, and has been shown empirically to help end-user programmers correct errors [11, 36].

8. CONCLUSIONS

Despite the growing number of end-user software developers, to date, relatively little research has sought to address the dependability issues that arise for end-user software. We are working on ways to bring software engineering methodologies to bear upon this problem, focusing in this paper on fault localization for end-user programmers.

This paper separately considers two factors that make up fault localization techniques—information base and mapping—and investigates their impact on technique effectiveness. The new contributions resulting from this work are:

- We show why traditional measures for fault localization effectiveness, which are based on the points of maximal system reasoning potential, are unsuitable in evaluating effectiveness in interactive environments, and present a set of measures that is instead based on the location of user feedback points.
- Of the three information bases compared, `I-NC`—which is the one least like those of many traditional fault localization techniques and also is the one least expensive in time and space—was the most effective of our three information bases, even in the presence of corrupt testing information.
- The results clearly indicated the need to evaluate the factor of information base separately from the factor of mapping. As our results show, even (seemingly) trivial changes in the mapping can result in significant differences in a technique's effectiveness, but these differences would improperly be attributed to the technique as a whole if mapping had not been evaluated separately from information base.
- The empirical results show that robustness is central in end-user fault localization, with 74% of the debugging sessions done in the context of at least one incorrect testing decision. The results further show that, given robustness features, it is possible for a technique to maintain at least some positive effectiveness.

All four of the above findings run counter to prior research

in fault localization aimed at professional programmers. As these results show, a number of traditional assumptions from this prior work are not appropriate for fault localization in end-user software environments.

9. ACKNOWLEDGMENTS

This work was supported in part by the EUSES Consortium via NSF grant ITR-0325273.

10. REFERENCES

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

[2] Y. Ahmad, T. Antoniu, and S. Goldwater. A type system for statically detecting spreadsheet errors. In *Proceedings of the IEEE Conference on Automated Software Engineering*, October 2003.

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

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

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

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

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

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

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

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

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

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

[13] D. Cullen. Excel snafu costs firm $24m. *The Register*, June 19, 2003. http://www.the-register.co.uk/content/67/31298.html.

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

[15] M. Erwig and M. Burnett. Adding apples and oranges. In *Proceedings of the International Symposium on Practical Aspects of Declarative Languages*, January 2002.

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

[17] T. Igarashi, J. Mackinlay, B.-W. Chang, and P. Zellweger. Fluid visualization of spreadsheet structures. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 118–125, 1998.

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

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

[20] A. Ko and B. Myers. Designing the Whyline: A debugging interface for asking questions about program failures. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, page (to appear), Vienna, Austria, April 24–29, 2004.

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

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

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

[24] S. Prabhakararao, C. Cook, J. Ruthruff, E. Creswick, M. Main, M. Durham, and M. Burnett. Strategies and behaviors of end-user programmers with interactive fault localization. In *Proceedings of the IEEE Symposium on Human-Centric Computing Languages and Environments*, pages 15–22, Auckland, New Zealand, October 28–31, 2003.

[25] B. Pytlik, M. Renieris, S. Krishnamurthi, and S. Reiss. Automated fault localization using potential invariants. In *Proceedings of the International Workshop on Automated and Algorithmic Debugging*, September 2003.

[26] O. Raz, P. Koopman, and M. Shaw. Semantic anomaly detection on online data sources. In *Proceedings of the International Conference on Software Engineering*, pages 302–312, Orlando, FL, May 2002.

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

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

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

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

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

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

[33] J. Voas. Software testability measurement for assertion placement and fault localization. In *Proceedings of the* 2^{nd} *International Workshop on Automated and Algorithmic Debugging*, pages 133–144, St. Malo, France, 1995.

[34] E. Wagner and H. Lieberman. Supporting user hypotheses in problem diagnosis on the web and elsewhere. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 30–37, 2004.

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

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

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