Rewarding "Good" Behavior: End-User Debugging and Rewards

Joseph R. Ruthruff, Amit Phalgune, Laura Beckwith, Margaret Burnett, and Curtis Cook
Oregon State University, Corvallis, Oregon, 97331, USA
{ruthruff, phalgune, beckwith, burnett, cook}@cs.orst.edu

Abstract

Emerging research has sought to bring effective debugging devices to end-user programmers. This research has largely focused on how well such devices bring genuine "functional" rewards to end users. However, emerging models of programming behavior indicate that another, often ignored, type of reward—perceivable rewards—can play an equally vital role in how well debugging devices serve end users. Using an empirically evaluated fault localization device, this paper investigates the impact such perceivable rewards can have on end-user debugging. Our results indicate that perceivable rewards alone can significantly improve the effectiveness and understanding of end users performing debugging tasks.

1. Introduction

Is it possible to bring some of the benefits of software engineering practices to end-user programmers? In pursuit of this goal, we have been developing a vision called *end-user software engineering*, whose aim is to bring some of the benefits of software engineering methods to end users—without requiring knowledge, or even interest, in software engineering itself. Some of the aspects of our end-user software engineering work include WYSIWYT, a visual testing methodology to help end users perform systematic testing [6, 22]; assertions to continually monitor values [8]; and visual fault localization [19, 23]. Fault localization, which aims to help end users debug, is the aspect of interest in this paper.

A problem with our fault localization device, which helps locate *faults* (erroneous source code) during the debugging process, has been that users do not choose to use it very often. In contrast to this, our other end-user software engineering devices have been very successful [9]. In trying to understand why our fault localization device has not shared in these successes, we focused our attention on the rewards offered by the device.

First, we empirically investigated rewards in the form of *functional effectiveness*. Empirical data reassured us on this point, showing that the device effectively pinpointed program points containing faults [23], and guided end users to effective debugging strategies [19]. Since the problem was not rooted in functional rewards, we began to suspect that the problem lay in *perceivable* rewards, and perhaps in their negatives as well, perceivable punishments. *Perceivable rewards* are those that a user *may perceive* to be a reason to use the device, yet do not directly reflect the device's functionality. (Similarly, *perceivable punishments* are those that a user *may perceive* to be a reason not to use the device.) The difference between perceivable and functional rewards is that perceivable rewards (1) are not tied to how effectively the device performs its purpose, and (2) potentially appeal to the user's emotions, such as by contributing to a user's sense of making progress toward their goal.

To investigate whether perceivable rewards play a significant role in end-user debugging, we conducted an experiment in which two experimental groups had environments with *exactly the same functionality*; the only difference being the *perceivable reward structure* of the fault localization device. Specifically, we set out to investigate the following research questions:

RQ1: *Effectiveness*: Do perceivable rewards impact end users' abilities to fix faults?
RQ2: *Usage*: Do perceivable rewards impact end users' usage of a debugging device?
RQ3: *Understanding*: Do perceivable rewards impact end users' understanding of a debugging device?

This paper is, to our knowledge, the first to differentiate functional rewards from perceivable rewards, and to isolate the impact of the latter.

2. Background and Related Work

2.1 Rewards in End-User Computing

The concept of rewarding users is an important component of a strategy we have been using called *Surprise-Explain-Reward* [26]. Our empirical results have shown that this strategy can be quite effective in promoting the use of end-user software engineering devices. Drawing from research on curiosity [16], "surprises" are used to arouse users' curiosity, enticing them to investigate items related to the surprise. For example, in our research prototype (Forms/3 [7], which is a member of the spreadsheet paradigm), as soon as a cell is given a formula, it is decorated with a checkbox containing a "?". If a curious

user investigates, the "explain" component, which is based on minimalist learning theory [10, 21], produces explanations via popup tooltips that communicate the meaning of the object, suggested action(s) if any, and the potential rewards for taking the action(s). For example, the explanation for a "?" is "Left click if cell's value is correct. Right click if it is wrong. These decisions help test and find errors." The "reward" component of the strategy, based on the Attention Investment model [5], consists of both interactive feedback and the potential for genuine improvements in the program. For example, a user's right click provides visual fault localization feedback (described in Section 3.2). If the feedback leads the user directly to an erroneous formula, he or she has received a clear reward for using the device.

The "reward" component of Surprise-Explain-Reward is grounded in Blackwell's Attention Investment model [5], an economic model to predict user behaviors in the realm of programming-like activities. According to this model, users take into account perceived benefits, perceived payoffs, perceived costs, and perceived risks when making cost-benefit decisions about whether to pursue an activity requiring an investment of attention (roughly the same as time). For example, a user deciding whether to use fault localization will consider the savings expected if the device identifies the fault(s) quickly (*perceived benefit*), the cost of learning to use the device and then invoking it (*perceived cost*), the expected future savings from using the device (*perceived payoff*), and possible losses or penalties such as being led to wrong cells by the device (*perceived risk*). (We will refer to these unexpected "after-the-fact" penalties as *punishments*, even though Blackwell's model does not explicitly label them as such.) This model suggests that perceivable rewards can impact users' abilities to benefit from devices such as our fault localization device.

A conceptual neighbor of the Attention Investment model is the model of Information Foraging [18]. This is a model to predict human activities in the domain of information access technologies. Although Information Foraging is based on a biological model rather than an economic one, it parallels Blackwell's model regarding costs and rewards. The costs of obtaining information are analogous to Blackwell's costs, and the information for which users "forage" can be viewed as a reward.

2.2 End-User Debugging

Work aimed specifically at aiding end users with debugging is beginning to emerge. Woodstein [25] is a software agent that assists e-commerce debugging. Ko and Myers present the Whyline [14], an "interrogative debugging" device for the event-based programming environment Alice. 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 [15], which classifies errors and their causes. Other strategies are statistical outlier finding [17] and anomaly detection [20], which use statistical analysis and interactive techniques to direct end-user programmers' attention to potentially problematic areas during automation tasks.

There has been a particularly large variety of work supporting program comprehension and debugging by end users in the spreadsheet paradigm. Igarashi et al. present devices to aid spreadsheet users in dataflow visualization and editing tasks [12]. S2 [24] provides a visual auditing feature in Excel 7.0: similar groups of cells are recognized and shaded based upon formula similarity, and are then connected with arrows to show dataflow. This technique builds upon the Arrow Tool, a dataflow visualization device proposed by Davis [11]. Ayalew and Mittermeir present a method of fault tracing based on "interval testing" and slicing [4] that has some similarities to our own work on assertions to help users automatically guard against faults [8]. There is also recent work to automatically detect certain kinds of errors, such as errors in spreadsheet units [1] and types [2].

Our work focusing on end-user testing and debugging support includes a visual testing methodology [6, 22], assertion support [8], and visual fault localization [19, 23]. Our previous empirical studies of these devices have focused on each device's functional effectiveness. This paper is the first to investigate the impact of perceivable rewards, and uses fault localization as a vehicle to accomplish this.

3. Rewards in Fault Localization

To investigate perceivable rewards in the realm of end-user debugging, we identified the perceivable rewards and punishments in our end-user programming environment. This section discusses those rewards as they pertain to our fault localization device. We then build upon this discussion to present two implementations with varying quantities of perceivable rewards and punishments.

3.1 WYSIWYT's Rewards

Our approach to fault localization is integrated into the "What You See Is What You Test" (WYSIWYT) visual testing methodology [22]. We have prototyped WYSIWYT in the spreadsheet paradigm, and it has also been extended to the screen transition paradigm [6] and to the dataflow paradigm [13]. In this section, we consider how the user interacts with WYSIWYT in terms of its rewards.

The underlying assumption behind WYSIWYT is that as a user incrementally develops a program, he or she can also be testing incrementally. Figure 1 presents an example of WYSIWYT in Forms/3 [7]. In WYSIWYT,

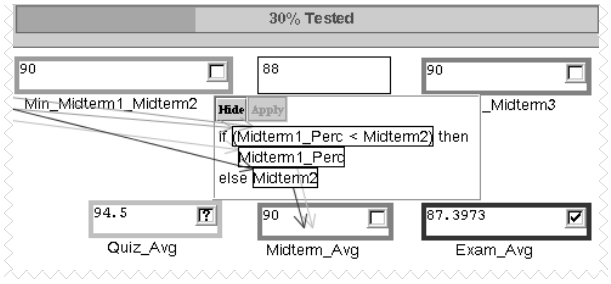

Figure 1. An example of WYSIWYT in Forms/3.

untested cells have red borders (light gray in this paper). Whenever users notice a correct value, they can place a checkmark (√) in the decision box at the corner of the cell they observe to be correct: this *testing decision* constitutes a successful test. Such checkmarks increase the "testedness" of a cell according to a test adequacy criterion detailed in [22], and this is reflected by adding more blue to the cell's border (more black in this paper).

The systematic colorings of these cell borders may be perceived as rewards by the user (at the cost of making a testing decision) because they show progress, and that the system is helping the user keep track of what still needs to be tested. A genuine functional reward may also ensue: as a result of attempting to turn every cell blue, the user's testing may reveal an incorrect value—in software engineering terminology, this value is a *failure*.

In addition to the cell granularity, testedness is depicted at two other granularities. First, at the subexpression granularity, dataflow arrows depict not only dataflow but also testedness. In Figure 1, the user has triggered arrows for the `Min_Midterm1_Midterm2` cell, showing flow from/to the subexpressions of `Min_Midterm1_Midterm2`'s formula. Each arrow's color depicts the testedness of its relationship, which helps explain to users why a cell's border is not completely blue. Second, at the spreadsheet granularity is an "overall testedness" progress bar, residing at the very top of the spreadsheet. For example, Figure 1's spreadsheet has an overall testedness of 30%. In past experiments, users have commented upon their efforts to reach 100% according to this bar: reaching 100% appears to be perceived as a reward worth attaining.

3.2 Fault Localization's Rewards

Instead of noticing that a cell's value is correct, the user might notice that the value is incorrect. In this case, instead of checking off the value, the user can put an X-mark in the cell's decision box. X-marks trigger *fault likelihood* calculations for each cell that might have contributed to the incorrect value [23]. Cells suspected of containing faults are colored in shades according to a yellow-orange continuum (shades of gray in this paper), with darker orange shades given to cells with increased fault likelihood. (Figure 2 presents an example of this behavior in the fault localization implementation of one experimental group.) The intended functional effectiveness reward is that the user might be led directly to the faulty cell (colored the darkest orange). Our empirical work has shown promising results in this regard [19].

3.3 Reward/Punishment Issues

When we carefully considered fault localization's perceivable reward structure, several issues were revealed. For each issue, there were trade-offs and/or side effects involved in deciding which of two apparently reasonable solutions to choose. We implemented both solutions in each case, putting one of the two into a

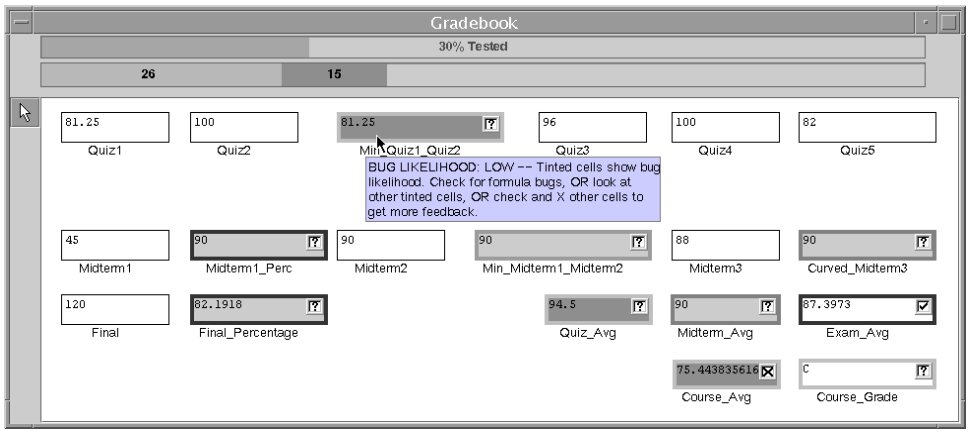

Figure 2. The user notices an incorrect value in `Course_Avg`—the value is obviously too low—and places an X-mark in the cell's decision box. All cells that could have dynamically contributed to this incorrect value have been colored in shades of yellow and orange (gray in this paper), with darker shades corresponding to increased fault likelihood (known as bug likelihood to users). In this example, three cells have an estimated fault likelihood of "Low" and five cells have a fault likelihood of "Very Low".

"Low-Reward" implementation and the other into a "High-Reward" implementation. We emphasize that there are rational reasons for *both* implementations, and the categorization was done strictly based on *quantity* of potentially perceivable rewards. There were five differences between the Low-Reward and High-Reward implementations. These differences are described next.

3.3.1 Mixed Message vs. "Loss" of Testedness

The first issue arose from previous users sometimes describing cells with blue borders as "correct" instead of tested. (Since end users are not likely to have software engineering background, this misunderstanding is not surprising.) If a tested cell has fault likelihood, this belief can lead to confusion due to a "mixed message" from seemingly conflicting testing and fault localization feedback, which indicates that a tested cell may still be "wrong" (i.e., have a fault in its formula).

We devised two solutions. One solution was to eliminate the mixed messages by removing the testing feedback—thereby removing the conflict with the fault localization feedback, but reducing the quantity of perceivable rewards—and the other was to allow testing feedback to remain. The first solution went into the "Low-Reward" implementation, and the second solution went into "High-Reward" for the following reasons.

In our first solution, changing the testing borders (and associated arrows) of colored cells back to red removes any indication that the cell is "tested"—this constitutes the first two differences in the reward structure. For consistency with the cell borders, which may have gone from "tested" to "untested", the overall testedness progress bar must also decrease—this is the third difference. (The underlying testing information in the system remains unchanged, and so removing the X-mark causes the colors and progress bar to revert to their previous state.) Because these differences might be perceived as a sign that past successful tests involving the colored cells have been discarded—in fact, a past empirical subject remarked that she had just lost all her work when this happened—this solution was assigned to the Low-Reward implementation due to the potential perception of punishment.

Our second solution (allowing the mixed messages to remain) is not necessarily "better". However, the solution does not contain the perceivable punishment of a loss of testedness information, and therefore has a greater quantity of rewards (less punishment) for using fault localization. For these reasons, the solution was put in the High-Reward implementation.

3.3.2 Explaining the Solutions

The removal of some testing feedback in the Low-Reward solution raised another issue—how to explain the red cell borders and arrows. For example, if a cell was 100% tested before it became colored, since the system hasn't discarded tests, the (now red-bordered) cell would still be 100% tested. Because there was no simple message to explain this, we chose not to display explanation tooltips at all for colored cells' (red) borders or their associated arrows, relying instead on the explanation tooltips describing colored interiors. This keeps with the common practice of conceptually "eliding away" information deemed too complex for end users.

Removal of the explanations for borders and arrows of colored cells could be perceived as a punishment (loss of information), which necessarily results in a smaller quantity of rewards—thus this solution was assigned to the Low-Reward implementation. This was the fourth reward/punishment difference in our study.

3.3.3 Competing with WYSIWYT

WYSIWYT had a greater quantity of perceivable rewards than the fault localization device. In addition to the cell-by-cell colorings, it also had the testedness progress bar, which in the past has seemed quite motivating to our users. Since fault localization did not have a progress bar of its own, this reward imbalance may have encouraged users more toward "positive" tests (checking off valid values) than toward "negative" tests (X'ing out invalid values).

For the High-Reward implementation, we instituted an additional reward of a "bug likelihood" progress bar (top of Figure 2). This bar summarizes the percentage of cells in the spreadsheet with each fault likelihood color intensity. In striving for the "reward" aspect, we included all the color intensities that are represented on the screen in the bar; this often has the effect of indicating overall "progress" in localizing one or more faults, as a smaller subset of cells grow darker (with higher "bug likelihood") due to increased testing information. For example, in Figure 2, the number of cells with higher likelihood is at 15%. This was the fifth and final reward/punishment difference in our study.

4. Experiment

We emphasize that the differences we have just described are not contrived differences. Rather, both groups' implementations had solid reasons. However, the High-Reward implementation always contained an implementation choice that was quantitatively greater in terms of perceivable reward, even if it had disadvantages from other perspectives.

To investigate the impact of the reward differences via the research questions enumerated in Section 1, we conducted a controlled laboratory experiment. The design of the experiment was such that both groups had environments with *exactly the same functionality*; the

only differences were those described in Section 3.3, which could affect users' perceptions of rewards and punishments associated with fault localization.

4.1 Procedures

Participants from the same experimental group were seated one per computer during six separate sessions. The participants (mostly business students) were randomly divided into two groups: a group of 24 participants with the Low-Reward implementation from Section 3.3 and a group of 30 participants with the High-Reward version. (The differences in group size are due to some participants not showing up for sessions.) Statistical tests on the background of participants—obtained from a background questionnaire—showed no significant differences between the groups in terms of grade point average, spreadsheet experience, or programming experience.

After completing the background questionnaire, we administered a 35-minute "hands-on" tutorial to familiarize participants with the environment. The participants were then given two tasks. We captured their actions in electronic transcripts, as well as their final spreadsheets.

At the conclusion of each task, we administered post-task questionnaires in which participants self-rated their performance on the task. The last (post-session) questionnaire also included questions assessing participants' comprehension of fault localization and their attitudes toward the features they had used.

Prior to the experiment, we conducted a four-participant pilot study to test our experimental procedures and materials.

4.2 Tutorial

In the tutorial, participants performed actions on their own machines with guidance at each step. The tutorial taught the use of WYSIWYT (checkmarks and associated feedback), but did not include any debugging or testing strategy content. Most importantly, we did not teach use of fault localization. Instead, participants were simply introduced to the mechanics of placing X-marks and given time to figure out any aspects of the feedback that they found interesting. To ensure that no influences would arise from tutorial differences, we presented the same tutorial to both groups.

4.3 Tasks

The experiment consisted of two tasks termed `Gradebook` and `Payroll` (shown in Figures 2 and 3, respectively). To make our tasks representative of real end-user spreadsheets, `Gradebook` was derived from an Excel spreadsheet of an (end-user) instructor, which we ported into an equivalent Forms/3 spreadsheet. (To accommodate Forms/3 features, a minor change was made to two minimization operators.) `Payroll` was a spreadsheet designed by two Forms/3 researchers using a payroll description from a real company.

These spreadsheets were seeded with five faults created by real end users. To obtain these faults, we provided three separate end users with the following: (1) a "template" spreadsheet for each task 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 and 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 references. Logical faults are mistakes in reasoning and are more difficult than mechanical faults. An omission fault is information that has never been entered into a cell formula, and is the most difficult [3]. We seeded `Gradebook` with three of the users' mechanical faults, one logical fault, and one omission fault, and `Payroll` with two mechanical faults, two logical faults, and one omission fault. `Payroll` was deemed the more difficult task due to its larger size, greater level of dataflow and intertwined dataflow relationships, and more difficult faults.

The participants were provided these `Gradebook`

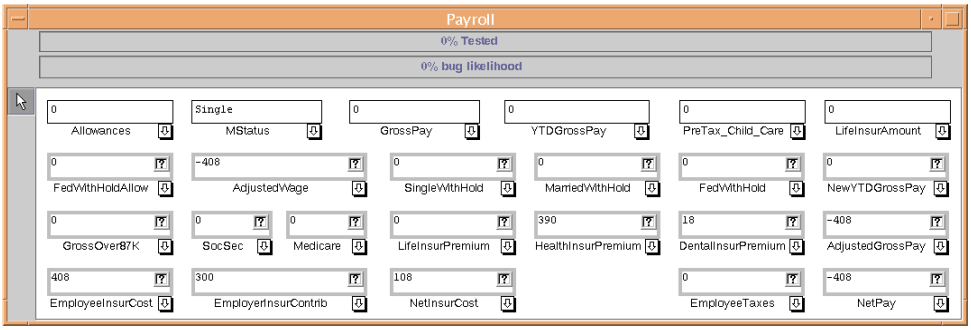

Figure 3. The `Payroll` task.

and Payroll spreadsheets and descriptions, with time limits of 22 and 35 minutes, respectively. (The debugging tasks necessarily involved time limits to ensure participants worked on both spreadsheets, and to remove possible peer influence of some participants leaving early.) The experiment was counterbalanced with respect to task order so as to distribute learning effects evenly. The participants were instructed, "Test the spreadsheet thoroughly to ensure that it does not contain errors and works according to the spreadsheet description. Also, if you encounter any errors in the spreadsheet, fix them."

4.4 Fault Localization Strategy

To calculate fault likelihood values, this experiment's fault localization strategy was the "Test Count Technique" in [23]. (We modified the strategy to support five, rather than four, color intensities.) We chose this strategy because it was the most robust strategy in the presence of user mistakes [23], which our previous research has shown to be an important issue [19].

5. Results

5.1 RQ1: Effectiveness

To evaluate participants' debugging performance, we measured the number of faults fixed by each group. As a statistical vehicle for investigating the number of faults fixed by the Low-Reward and High-Reward participants, we state the following null hypothesis:

H1: There will be no difference in the number of faults fixed by Low-Reward and High-Reward participants.

Figure 4 summarizes the number of faults fixed for each task by each group. Although there was no significant difference in the number of faults fixed in the Gradebook task (two-sided t-test: t=0.3936, df=53, p=0.696), the High-Reward group participants fixed significantly more faults in the Payroll task (t=-2.31, df=53, p=0.025). (A two-sided t-test, rather than repeated measures ANOVA, was selected because the two experimental tasks were distinctly different—for example, in types of faults and task times—and because the experiment was counterbalanced with respect to task order.) Therefore, we reject H1.

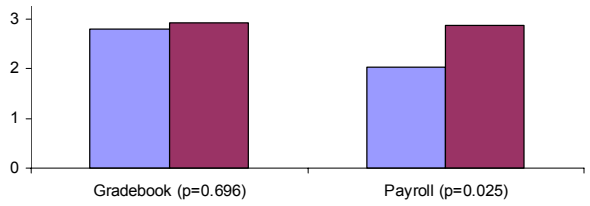

Figure 4. The average number of faults fixed by each participant. The High-Reward (dark bars) group fixed significantly more faults than the Low-Reward group (light bars) on the more difficult Payroll task.

Recall that the functionality of the Low-Reward and High-Reward environments was *exactly the same*: the same cells were colored as likely to contain faults, using the same coloring scheme, same calculations, etc. Thus the same functional effectiveness reward potentially existed in both environments. The only changes were in aspects that might affect users' perception of rewards. The significant difference in debugging effectiveness suggests that these aspects may have a powerful impact on users' effectiveness.

5.2 RQ2: Usage

One possible explanation for the effectiveness difference could be that the High-Reward participants used X-marks more. We consider this using two metrics: (1) "Persistent" X-marks that, once placed, persist until changes in the program's source code (spreadsheet formulas) cause those X-marks to become obsolete (at which point the system removes them), and (2) "Total" X-marks, which include all X-marks placed, even transitory ones that are placed only for a moment and then removed (such as if a user is experimenting with X-marks to see what they do, or changes his/her mind about whether a value is really incorrect). The latter metric accounts for even brief feedback that the user saw before removing the X-mark, as even the brief presence of such feedback could prove constructive to a debugging effort.

H2: There will be no difference in the usage of fault localization by Low-Reward and High-Reward participants.

The number of "Persistent" and "Total" X-marks per spreadsheet task by each group can be found in Table 1. A two-sided t-test on these results showed no significant differences between the groups for either "Persistent" X-marks (Gradebook: t=0.3436, df=53, p=0.7325; Pay-

	Metric 1: "Persistent"	Metric 2: "Total"
Gradebook:		
Low-Reward (n=24)	1.58 (2.98)	4.67 (5.77)
High-Reward (n=30)	1.52 (2.26)	4.13 (5.74)
Payroll:		
Low-Reward (n=24)	1.96 (3.07)	8.67 (13.88)
High-Reward (n=30)	2.00 (2.68)	5.98 (7.34)

Table 1. Mean (standard deviation) number of X-marks placed. The number of "Total" X-marks that participants placed ranged from 0 to 60 for each task, with large variations in the data. None of the differences are significant.

`roll`: t=0.9292, df=53, p=0.357) or "Total" X-marks (`Gradebook`: t=0.1629, df=53, p=0.8712; `Payroll`: t=–0.0078, df =53, p=0.9938). Therefore, we cannot reject H2.

We were surprised by these results because, when combined with those of Section 5.1, they tell us that High-Reward participants fixed more faults *despite no difference in the amount of usage of the fault localization device*!

The lack of difference in usage rate was corroborated by one of the post-session questions, which solicited the users' opinions on the helpfulness of various aspects of the system. Specifically, we asked for functionality "helpfulness ratings" (regarding finding and fixing faults) on the WYSIWYT, explanation, and fault localization devices. They rated the helpfulness of each item on a 5-point Likert scale. The mean ratings scored WYSIWYT the highest, then explanations, and finally fault localization. However, none of the differences were significant. The lack of difference between the two groups' ratings reaffirms the usage data: fault localization was not deemed any more functionally helpful by the High-Reward group than by the Low-Reward group, leading to similar usage patterns.

5.3 RQ3: Understanding

From the previous section, it is clear that the greater effectiveness of the High-Reward group cannot be explained by mere usage of the fault localization device. In this section, we consider an alternative explanation—that the High-Reward group had more understanding of the feedback's implications than the Low-Reward group did.

H3: There will be no difference in the ability to understand fault localization feedback by Low-Reward and High-Reward participants.

We consider two types of comprehension: the shallow level of being able to interpret feedback received, and the deeper level of being able to predict feedback under various circumstances. Two questions on the post-session questionnaire were used to evaluate interpretation (e.g., "What does it mean when the color in the interior of one cell is darker than the others?"). Six questions measured prediction, such as asking participants to determine what would happen to a particular cell (whether it would turn darker, lighter, or remain the same color) if an X-mark was placed in a specified cell.

High-Reward participants provided more correct responses than did the Low-Reward participants *on all but one question* (although not always significantly). The difference in correct responses between the two groups was significant at the 0.05 level (two-sided t-test: t= -2.01, df=53, p=0.0496). Most of this difference was accounted for by the prediction scores (t=-2.05, df=53, p=0.0454). Figure 5 shows the mean scores question-by-question. Clearly, H3 is rejected.

It is actually quite remarkable that the High-Reward participants understood the feedback better, since it is this group who experienced the "mixed messages" described in Section 3.3.1. To investigate whether these "mixed messages" were really mixed, we had included a four-part question in the post-session questionnaire for the High-Reward group (since only they had the mixed message). Given a cell with a blue border and an orange interior, the first three parts of the question asked about the correctness of the value, the cell's testedness, and the cell's fault likelihood; and the fourth part asked whether the combination of a blue border and an orange interior made sense. The results were mixed. The percentages answered correctly by the participants on the first three parts varied, ranging from 50% to 87%. More to the point, only about half (54%) said the mixed message made sense. The other 46% either said it did not make sense, they did not know, or did not answer the question. Clearly, the mixed message introduced confusion in many of the High-Reward participants.

It would be reasonable to expect that the confusion they reported would seriously hamper understanding of the feedback. Despite this confusion, however, the perceivable rewards still drew significantly better overall understanding.

5.4 Interpreting the Results

We designed this experiment to increase the perceivable rewards of using fault localization, without changing the device's functional rewards, in order to investigate the research questions outlined in Section 1. However, there are other interpretations that could explain our results. For example, the curiosity that may have resulted from seemingly inconsistent feedback (e.g., mixed messages) could have caused users to better debug the more difficult `Payroll` task, and to better under-

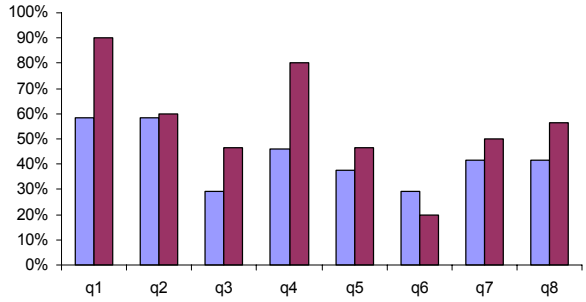

Figure 5. Participants who answered each question correctly: Low-Reward participants (light bars) and High-Reward participants (dark bars). Questions 1-6 are prediction questions, and 7-8 are interpretation questions.

stand the device's feedback. Alternatively, providing additional feedback, even if it conflicts with preexisting feedback, may have in fact bettered users' debugging efforts. Further investigation is needed to tease apart the impact directly due to perceivable rewards from other factors, such as curiosity, that may be involved.

6. Conclusions

Research often focuses on improving programming environments and devices without taking perceivable rewards and punishments into consideration. Our results indicate that perceivable rewards alone may significantly improve the effectiveness and understanding of end users performing debugging tasks. This finding is important for a number of reasons. First, it indicates that the traditional approach of addressing functional rewards, while valuable, may not be the only means of improving end-user debugging. Second, it suggests the need for further work in this area. Finally, our results add to the emerging body of evidence validating the importance of rewards in end-user programming. This study is especially unique in that it is the first to differentiate the importance of functional rewards from perceivable rewards in end-user debugging.

Acknowledgments

This work was supported in part by NSF under ITR-0082265 and the EUSES Consortium via NSF grant ITR-0325273.

References

[1] R. Abraham and M. Erwig, "Header and Unit Inference for Spreadsheets Through Spatial Analyses", *Proc. IEEE Symp. Visual Langs. and Human-Centric Computing*, 2004 (to appear).

[2] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi, "A Type System for Statically Detecting Spreadsheet Errors", *Proc. IEEE Conf. Auto. Soft. Eng.*, 2003.

[3] C. Allwood, "Error Detection Processes in Statistical Problem Solving", *Cognitive Science* 8, 4, 1984, 413-437.

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

[5] A. Blackwell, "First Steps in Programming: A Rationale for Attention Investment Models", *Proc. IEEE Symp. Human-Centric Computing Langs. and Envs.*, 2002, 2-10.

[6] 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 Langs. and Envs.*, 2003, 203-210.

[7] M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang, "Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm", *J. Func. Prog.* 11, 2, 2001, 155-206.

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

[9] M. Burnett, C. Cook, and G. Rothermel, "End-User Software Engineering", *Comm. ACM*, Sept. 2004 (to appear).

[10] J. Carroll, *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*, MIT Press, Cambridge, MA, 1990.

[11] J.S. Davis, "Tools for Spreadsheet Auditing", *Int. J. Human-Computer Studies* 45, 1996, 429-442.

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

[13] M. Karam and T. Smedley, "A Testing Methodology for a Dataflow Based Visual Programming Language", *Proc. IEEE Symp. Human-Centric Computing Langs. and Envs.,* 2001, 280-287.

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

[15] A.J. Ko and B.A. Myers, "Development and Evaluation of a Model of Programming Errors", *Proc. IEEE Symp. Human-Centric Computing Langs. and Envs.*, 2003, 7-14.

[16] G. Lowenstein, "The Psychology of Curiosity", *Psychological Bulletin* 116, 1, 1994, 75-98.

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

[18] P. Pirolli and S. Card, "Information Foraging in Information Access Environments", *Proc. ACM Conf. Human Factors Computing Systems*, 1995, 51-58.

[19] S. Prabhakararao, C. Cook, J. Ruthruff, E. Creswick, M. Main, M. Durham, and M. Burnett, "Strategies and Behaviors of End-User Programmers with Interactive Fault Localization", *Proc. IEEE Symp. Human-Centric Computing Langs. and Envs.*, 2003, 15-22.

[20] O. Raz, P. Koopman, and M. Shaw, "Semantic Anomaly Detection in Online Data Sources", *Proc. Int. Conf. Soft. Eng.*, 2002, 302-312.

[21] M. Rosson and C. Seals, "Teachers as Simulation Programmers: Minimalist Learning and Reuse", *Proc. ACM Conf. Human Factors Computing Systems*, 2001, 237-244.

[22] G. Rothermel, M. Burnett, L. Li, C. Dupuis, and A. Sheretov, "A Methodology for Testing Spreadsheets", *ACM Trans. Soft. Eng. Meth.* 10, 1, 2001, 110-147.

[23] 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. Soft. Visualization*, 2003, 123-132.

[24] J. Sajanieme, "Modeling Spreadsheet Audit: A Rigorous Approach to Automatic Visualization", *J. Visual Langs. Computing* 11, 1, 2000, 49-82.

[25] E.J. Wagner and H. Lieberman, "Supporting User Hypotheses in Problem Diagnosis on the Web and Elsewhere", *Proc. Int. Conf. Intelligent User Interfaces*, 2004, 30-37.

[26] 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 Computing Systems*, 2003, 305-312.