AN ABSTRACT OF THE THESIS OF

TJ Robertson for the degree of Master of Science in Computer Science presented on 2/26/2004
Title: Effects of Interruption-Style on End-User Programmers

Abstract approved:

Redacted for Privacy

Margaret M. Burnett

This thesis presents the results of two studies that investigate the question of what interruption-
styles are most appropriate for end-user programmers who are debugging programs. In the
studies, end-user programmers are presented with surprises that encourage them to investigate,
use, and learn about debugging devices in their programming environment. We used various
interruption-styles to present these surprises to the end-user programmers, and we evaluated how
they affected the end-user programmers ability to learn about the debugging features, their
accuracy at debugging their programs, and how accurate they were at judging how well they had
dugged their programs. The three styles we compared were immediate-style interruptions
(which force the user to acknowledge them), low-intensity negotiated-style interruptions (which
do not force the user to acknowledge them, but rather use visual elements such as red circles
around cell values in order to notify users that there is something for them to do), and high-
intensity negotiated-style interruptions (which are the same as low-intensity negotiated-style
interruptions except that the visual elements are more intense, e.g. they are larger and they blink).
We found that low-intensity negotiated-style interruptions best supported end-user programmers
dugging, learning, and self-assessment. We also found that immediate-style and high-intensity
negotiated-style interruptions had very similar effects on the end-user programmers.
Effects of Interruption-Style on End-User Programmers

by

TJ Robertson

A THESIS
submitted to
Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented February 26, 2004
Commencement June 2004
Master of Science thesis of TJ Robertson
presented on February 26, 2004

APPROVED:

Redacted for Privacy

Major Professor, representing Computer Science

Redacted for Privacy

Head of the School of Electrical Engineering and Computer Science

Redacted for Privacy

Dean of the Graduate School

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

Redacted for Privacy

TJ Robertson, Author
ACKNOWLEDGEMENTS

I would like to thank Dr. Margaret Burnett for providing me with guidance in doing my research (ideas, experiments, and writing). I would also like to thank the Forms/3 team not only for Forms/3 but also for their help in facilitating the studies in this thesis. I would especially like to thank Shrinu Prabhakararao for his hard work on statistical analysis, creating screenshots, and content for the first paper. I would also like to thank Dr. Cook for his knowledge and work in analyzing the data. Thanks also to Dr. Derryberry in OSU's psychology department for his help in understanding the psychological details relevant to this research. Thanks also go to my family for raising me and providing an atmosphere conducive to academic success.
CONTRIBUTION OF AUTHORS

Shrinu Prabhakararao administered the first study, as well as performing the statistical analysis and much of the writing and figures for the first paper. He also helped administer the second study. Margaret Burnett helped brainstorm, write, and edit the papers.

Curtis Cook helped with brainstorming, statistical analysis, and writing part of the first paper.

Joseph R. Ruthruff helped administer the studies as well as writing part of the first paper.

Laura Beckwith helped administer the studies as well as writing part of the first paper.

Amit Phalgune helped administer the studies, implemented the immediate-style interruptions, and helped write part of the first paper.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 General Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Impact of Interruption Style on End-User Debugging</td>
<td>2</td>
</tr>
<tr>
<td>3 Impact of High-Intensity Negotiated-Style Interruptions on End-User</td>
<td>21</td>
</tr>
<tr>
<td>Debugging</td>
<td></td>
</tr>
<tr>
<td>4 General Conclusion</td>
<td>46</td>
</tr>
<tr>
<td>Combined Bibliography</td>
<td>47</td>
</tr>
<tr>
<td>Appendices</td>
<td>51</td>
</tr>
<tr>
<td>Appendix A – Bibliographic Notes</td>
<td>52</td>
</tr>
<tr>
<td>Appendix B: Notes on the Effects of Professional Spreadsheet</td>
<td>60</td>
</tr>
<tr>
<td>Experience on High-Intensity Participants</td>
<td></td>
</tr>
<tr>
<td>Appendix C: Tutorial Script</td>
<td>62</td>
</tr>
<tr>
<td>Appendix D: Spreadsheet Descriptions and Questionnaires</td>
<td>70</td>
</tr>
<tr>
<td>Appendix E: Experiment Spreadsheet Formulas</td>
<td>96</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Assertion examples.</td>
<td>8</td>
</tr>
<tr>
<td>1.2</td>
<td>Instances of immediate-style interruptions in the experiment.</td>
<td>9</td>
</tr>
<tr>
<td>1.3</td>
<td>Participants who answered each comprehension question correctly</td>
<td>12</td>
</tr>
<tr>
<td>2.1</td>
<td>Assertion examples.</td>
<td>28</td>
</tr>
<tr>
<td>2.2</td>
<td>What the high-intensity interruptions look like (top row) versus low-intensity interruptions (bottom row).</td>
<td>30</td>
</tr>
<tr>
<td>2.3</td>
<td>Participants' scores on assertion-related comprehension questions.</td>
<td>32</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Mean number of minutes:seconds before participants entered their first assertion in each task.</td>
<td>10</td>
</tr>
<tr>
<td>1.2</td>
<td>Categorizations of the comprehension questions.</td>
<td>12</td>
</tr>
<tr>
<td>1.3</td>
<td>The mean (standard deviation) productivity. Significant differences (p&lt;.05) are shaded.</td>
<td>14</td>
</tr>
<tr>
<td>1.4</td>
<td>Average number of each type of activity engaged in by the participants.</td>
<td>16</td>
</tr>
<tr>
<td>1.5</td>
<td>Regression analyses of actual bugs corrected vs. perceived bugs corrected.</td>
<td>16</td>
</tr>
<tr>
<td>2.1</td>
<td>Categorizations of the comprehension questions.</td>
<td>32</td>
</tr>
<tr>
<td>2.2</td>
<td>Average time until users entered their first assertion on both tasks, as well as after they started Help-Me-Test for the first time (on the first task)</td>
<td>33</td>
</tr>
<tr>
<td>2.3</td>
<td>The percentage of correct assertions that participants created and the percentage of cells with correct assertions</td>
<td>34</td>
</tr>
<tr>
<td>2.4</td>
<td>Productivity at fixing bugs for both tasks.</td>
<td>36</td>
</tr>
<tr>
<td>2.5</td>
<td>Average number of each type of activity engaged in by the participants.</td>
<td>37</td>
</tr>
<tr>
<td>2.6</td>
<td>Linear regression p-values for Number of bugs fixed vs belief in how successful they were at fixing all bugs.</td>
<td>38</td>
</tr>
</tbody>
</table>
GENERAL INTRODUCTION

Our research group has been working on helping end-user programmers (such as spreadsheet developers) find and fix errors in their programs. Towards this end, we have developed a method, termed Surprise-Explain-Reward, of encouraging users to explore and learn about the debugging features of their programming environment. The basis of Surprise-Explain-Reward is presenting surprises to end-user programmers (such as circles around possibly incorrect spreadsheet cells) in order to pique their curiosity and encourage them to explore the source of the surprise. In [27] this technique was shown to be successful at its goal of encouraging users to learn about and use debugging features.

This thesis is a “manuscript document” thesis. According to the OSU Graduate School’s Thesis Guide, this type of thesis is a collection of scholarly manuscripts or journal article addressing a common theme. In addition, a general introduction and general conclusion must be provided.

In the two papers that constitute this thesis, we explore the question of how best to surprise end-user programmers with respect to not disrupting them or harming their ability to learn and use the debugging features. To this end, we have more formally defined surprise communications as interruptions according to McFarlane’s [18] classification of interruptions.

We identify our standard surprise communications (used in [27]) as negotiated-style interruptions according to McFarlane’s classification. Negotiated-style interruptions inform users of a pending message without actually preventing them from working on what they are doing.

In the first paper of this thesis, we show how to map all of the negotiated-style interruptions of [27] to a set of immediate-style interruptions, which do interrupt what they are working on (we chose to use pop-up “ok” dialog boxes). We then compare how well the negotiated vs immediate-style participants performed at learning the debugging features, debugging the spreadsheets, and how well they were able to judge their accuracy at fixing bugs.

In the second paper of this thesis, we become more specific about negotiated-style interruptions and choose to divide them into low-intensity negotiated-style interruptions (our original [27] negotiated-style interruptions) and high-intensity negotiated style interruptions, which means that they are more intense (such as being larger, blinking, and/or accompanied by text). We then compare participants with high-intensity interruptions to the participants with low-intensity and immediate-style interruptions.

In comparing these three variations of interruptions with one another, our intention was to get a clear picture of how end-user programmers are affected by interruptions along the spectrum from low-intensity to high-intensity to immediate-style interruptions.
IMPACT OF INTERRUPTION STYLE ON END-USER DEBUGGING

T. J. Robertson, Shrinu Prabhakararao, Margaret Burnett,
Curtis Cook, Joseph R. Ruthruff, Laura Beckwith, and Amit Phalgune

ACM Conference on Human Factors in Computing Systems,
Vienna, Austria,
April 2004
Impact of Interruption Style on End-User Debugging

T. J. Robertson, Shrinu Prabhakararao, Margaret Burnett,
Curtis Cook, Joseph R. Ruthruff, Laura Beckwith, and Amit Phalgune
Oregon State University
Corvallis, OR 97331
{robertth, prabhash, burnett, cook, ruthruff, beckwith, phalgune}@cs.orst.edu

ABSTRACT

Although researchers have begun to explicitly support end-user programmers’ debugging by providing information to help them find bugs, there is little research addressing the proper mechanism to alert the user to this information. The choice of alerting mechanism can be important, because as previous research has shown, different interruption styles have different potential advantages and disadvantages. To explore impacts of interruptions in the end-user debugging domain, this paper describes an empirical comparison of two interruption styles that have been used to alert end-user programmers to debugging information. Our results show that negotiated-style interruptions were superior to immediate-style interruptions in several issues of importance to end-user debugging, and further suggest that a reason for this superiority may be that immediate-style interruptions encourage different debugging strategies.

Categories & Subject Descriptors: D.1.7 [Programming Techniques]: Visual Programming; D.2.4 [Software Engineering]: Software/Program Verification—Validation; D.2.6 [Software Engineering]: Programming Environments—Interactive environments; H.1.2 [Information Systems]: User/Machine Systems—Software psychology; H.4.1 [Information Systems Applications]: Office Automation—Spreadsheets; H.5.2 [Information Interfaces and Presentation]—User Interfaces (D.2.2, H.1.2, I.3.6)

General Terms: Human Factors, Languages

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
INTRODUCTION

Research on end-user programming has, in the past, concentrated primarily on supporting end users' creation of new programs. But recently, researchers have begun to consider assisting end users in debugging these programs. Research on how to support debugging by end users generally involves the system performing some kind of reasoning relevant to program bugs or program structure, followed by communication of the results to the user (e.g., [11, 15, 17, 19]). But, how should this communication be done?

Such communication, when initiated by the system, involves some form of interruption. Research has shown that interruptions can have detrimental effects on the user's concentration and productivity, but can be helpful in calling important facts to the user's attention. Since previous research has most often concentrated on interruptions in relatively simple tasks, it is not clear whether and how these findings apply to the complex domain of interest here: debugging, done by a population without much experience in debugging.

In our work on supporting debugging by end-user programmers, interruptions are a vehicle for attempting to surprise the user as part of our Surprise-Explain-Reward strategy [20]. The element of surprise is used to arouse users' curiosity about two types of things: (1) features in the environment that might help them debug, and (2) locations in the program where the system believes bugs are lurking. In previous empirical work [4, 20], Surprise-Explain-Reward, supporting the debugging device used in the experiment reported here, succeeded on both these counts.

Surprises in the Surprise-Explain-Reward strategy can be communicated via negotiated-style interruptions, which, following McFarlane's classification of interruptions [14], are interruptions that inform the user of a pending message but do not force them to acknowledge it immediately. This is the style that we have used in our prototype so far. An example of a negotiated-style interruption in word processing software is the red underline that can appear under misspelled words.

In contrast to negotiated-style interruptions, a style used in some software is immediate-style interruptions, which are interruptions that require user action. A widespread example is pop-up dialog boxes that the user must move or close in order to resume the interrupted task.

In this paper we consider the impacts of negotiated- and immediate-style interruptions on end users' debugging efforts. We focus specifically on end-user programmers. For that population,
we consider impacts in the dimensions of learning, productivity, and ability of end-user programmers to self-assess their debugging performance:

**RQ1:** Which interruption style is more effective in helping end users learn debugging devices?

**RQ2:** Which interruption style is more effective in helping end users fix bugs?

**RQ3:** With which interruption style can end users best predict when all the bugs are gone?

**RELATED WORK**

McFarlane identified four ways of interrupting users [14]. (In addition to the negotiated and immediate styles, he considered two others. *Mediated interruptions* present information when the system decides it is an appropriate time to interrupt the user. *Scheduled interruptions* present information at fixed time intervals.) McFarlane found that no one style was a clear winner, but rather that different styles were appropriate for different goals.

Based on the results of his study, McFarlane suggested design guidelines for when to use each style. The guidelines recommend negotiated-style interruptions when the goal is efficiency on either the primary task or the interruption’s task—i.e., the task to which the interruption is trying to bring attention. Negotiated-style interruptions are also recommended over immediate-style when accuracy on the primary task, accuracy on the interruption’s task, or judgment of accuracy is important. The guidelines recommend immediate-style interruptions when the goal is completeness or promptness on the interruption’s task.

McFarlane’s guidelines, however, were created based on the results of a study in which users were being interrupted during a speed-critical, but cognitively simple, video game task, in order to perform a completely irrelevant matching task. Thus, these guidelines may not apply to interruptions relevant to the complex task of debugging.

Regarding complex tasks (such as tasks that require the user to hold many things in their short-term memory), interrupting the user during the task can harm their performance because they must re-orient themselves when returning to the primary task [1, 2]. Although choosing appropriate times to interrupt them [7, 10] can reduce the reorientation penalty, overall this body of research suggests that immediate-style interruptions will slow down users’ debugging.

Yet, it has been found that interruptions highly relevant to the task at hand are less disrupting than non-relevant interruptions [7, 18]. In fact, one project found that interruptions that provided users with hints on how to complete their task could be more helpful than harmful to the user [16].

Relevant interruptions often aim, at least in part, to help users learn to employ useful techniques. This aspect is particularly pertinent for end-user debugging, because many end users have never learned effective debugging. Immediate-style interruptions are a successful vehicle in
on-line learning systems (e.g., as in [6, 13]). When the interruption’s goal is to help users learn, the practices of such learning systems are consistent with McFarlane’s recommendation to use immediate-style interruptions for completeness and promptness on the interruption’s task. Since effective support for learning of new debugging features seems necessary to users’ debugging productivity, immediate-style interruptions’ successful track record with that aspect could be predicted to have a cascading advantage for debugging: first for learning, and as a result for productivity.

**EXPERIMENT**

To investigate the research questions enumerated in the introduction, we conducted a controlled laboratory experiment with two groups of end-user participants.

**Design, Procedures, and Tasks**

The experiment replicated the design reported in [20] except for the treatment of interruptions. Interruption style was manipulated for one debugging device: assertions (described below). For the negotiated-style group, assertions were supported by the Surprise-Explain-Reward strategy via negotiated-style interruptions only. For the immediate-style group, these negotiated-style interruptions were supplemented by immediate-style interruptions.

The participants were 38 business majors with spreadsheet experience. We used the data from the 16 participants of our earlier experiment [20] as the negotiated-style group, and recruited 22 additional participants for the immediate-style group. To ascertain whether the participants in the two groups had similar backgrounds, we administered a background questionnaire and analyzed the data. There was no significant difference between the background information of the two groups. Subsequent analysis combining each background item with treatment type confirmed that differences between negotiated-style versus immediate-style groups’ backgrounds did not affect results.

Replicating our previous experiment, after a tutorial, the participants were asked to debug two spreadsheets, Grades and Weekly Pay, with time limits of 35 and 22 minutes, respectively; see [4, 20] for details of these spreadsheets. (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 problem order so as to distribute learning effects evenly.

The problem descriptions given to the participants included details of what the spreadsheet was to accomplish. The participants were instructed to “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.”

Electronic transcripts recorded all on-line activity for later analysis. After each debugging problem, participants answered questionnaires in which they rated how well they thought they had debugged the spreadsheet. After the second problem, the participants answered questions testing their understanding of assertions, the debugging device for which we manipulated interruption style.

**The Environment for Interruptions**

One of the most widely used programming paradigms by end-user programmers is the spreadsheet paradigm. Thus, the prototype environment for Surprise-Explain-Reward is the research spreadsheet language Forms/3 [3]. One of the end-user debugging devices supported by Surprise-Explain-Reward is assertions on spreadsheet cells, which past empirical work has shown that end users can use effectively [4, 20]. For this experiment, assertions were the vehicle for investigating interruption style.

Assertions are represented as allowable ranges for a cell’s value. When the user creates an assertion (termed a *user-entered assertion*), it is propagated through the dataflow chain of the spreadsheet (creating *system-generated assertions*), so that cells have assertion ranges if the cells that they reference have assertion ranges. When an assertion range is violated, a red circle is drawn around the cell’s value; such a violation is termed a *value violation*. When a system-generated assertion conflicts with a user-entered assertion, a red circle is drawn around the two conflicting assertions; such a conflict is termed an *assertion conflict*.

For example, in Figure 1, the user has entered an assertion for cell input_temp, which propagated through output_temp’s formula to create a system-generated assertion. Since the values “200” and “33.3333” do not fulfill their cells’ assertions, they are circled. Finally, the user also entered an assertion “0 to 100” for output_temp; since it disagrees with the cell’s other assertion, they are both circled.

Besides assertions, participants had other debugging devices available. If they decided a cell’s value was correct, they could check it off in the corner of each cell (e.g., the checkbox in Figure 1’s output_temp cell). This was rewarded by incrementing “testedness” indicators in the environment, such as changing the cell’s border color toward blue along a red-blue continuum to indicate increased testedness. If they wanted help conjuring up more test inputs, participants could push a *Help-Me-Test* button to automatically generate more values [8]. Help-Me-Test’s role in our experiment was in its use as a springboard by the Surprise-Explain-Reward strategy for introducing users to assertions.
Here is how this springboard works: When a user invokes Help-Me-Test, the system not only generates values for input cells, but also creates a (usually incorrect) "guessed" assertion to place on these cells. These guessed assertions, termed HMT assertions (because they are generated by Help-Me-Test), are intended to surprise the user into becoming curious about assertions. They can satisfy their curiosity using tool tips, which will inform them of the benefits and syntax of assertions. If the user follows up by accepting an HMT assertion (either as guessed or after editing it), the resulting assertion will be propagated as in Figure 1. As a result, value violations or assertion conflicts may occur; if so, red circles will appear as in Figure 1, which are often another surprise. All of these attempted surprises are communicated via interruptions.

**Negotiated-Style Interruptions**

The communications as just described come in the form of negotiated-style interruptions. For example, the red circles around potentially incorrect cell values are negotiated-style interruptions. They are interruptions because they request attention from the user; they are negotiated-style because the user decides when and if they want to see the content of the message, which they can do via tool tips at the time of their choice.

**Mapping Negotiated-Style Interruptions to Immediate-Style Interruptions**

For the immediate-style group, we did a one-to-one mapping of each negotiated-style interruption to an immediate-style interruption. (In addition, to eliminate memorization as a factor, the tool tips remained available, and to eliminate directness differences as a factor, the negotiated-style output devices, such as red circles around offending values, also remained present.) The immediate-style interruptions took the form of a modal pop-up dialog box containing exactly the same message as the tool tips, as in Figure 2.
**Tutorial**

We began with a 25-minute hands-on tutorial on the environment just described. To ensure that no influences would arise from tutorial differences, we presented exactly the same tutorial (with negotiated-style interruptions only) to both groups.

The tutorial taught use of the checkbox for checking off cells and Help-Me-Test at the GUI level, but did not include any debugging or testing strategy content. Most importantly, we did not present assertions—in fact, they were never even mentioned. (This was to support our investigation of RQ1, the system’s ability to promote learning.) Instead, participants were simply introduced to the use of tool tips and given time to explore via a practice task.

**RESULTS AND DISCUSSION**

When McFarlane introduced the concept of negotiated- and immediate-style interruptions, he provided guidelines suggesting how these interruption styles would affect performance on the primary task, judgment of performance on the primary task, and performance on the task to which the interruptions are drawing the user. However, as stated in the related work section, McFarlane’s guidelines are based on a study in which users were being interrupted during a cognitively simple video game task, in order to perform a completely irrelevant matching task. Throughout our results section, we will compare our results to McFarlane’s predictions. In this way, we will test the applicability of his guidelines to relevant interruptions during the cognitively challenging task of debugging.

**RQ1: Learning Results**

Research in debugging for end-user programmers has focused on trying to guide end users when users are interrupted:

1. when users indicate interest in assertions
2. when there are value violations
3. when HMT generates assertion(s)
4. when there is an assertion conflict
5. when system-generated assertion(s) are created

![Figure 2: Instances of immediate-style interruptions in the experiment.](image)
users to new behaviors, supported by the system, that will result in productive debugging. For example, much of this work attempts to guide users in narrowing down the locations of bugs (e.g., emerging work from Ko and Myers [12], work by Wagner and Lieberman [19] and by our own group [17]). Such guidance invariably is accompanied by new features (colorings, diagrams, and new interaction devices), which users must master. Thus, the system must include devices to help the users achieve this mastery. In this section, we consider which interruption style best facilitates helping end users learn to use such debugging features.

Interest “Draw”

If the user has a choice about whether to attend to or ignore a new device, the system may, as its first task, need to draw the user’s attention to the device being introduced. Our statistical vehicle for considering which interruption style best draws users’ interest to learning assertions, the debugging device to which interruptions are trying to draw attention, is the following (null) hypothesis:

\[ H_1: \text{There will be no difference in the elapsed time until the negotiated- and immediate-style participants are enticed to enter assertions.} \]

We will denote whichever spreadsheet problem a participant worked first or second as “Task 1” or “Task 2,” respectively. As Table 1 shows, on the participants’ first task, the negotiated-style participants placed their first assertions somewhat later than did the immediate-style participants. (This trend would agree with McFarlane’s prediction that immediate-style interruptions will lead to promptness on the interruptions task.) By the second task, however, the differences disappeared; in fact, they were reversed. None of these differences were significant at the .05 level (Mann Whitney: Task 1 \( p = 0.1153 \), Task 2 \( p = 0.0952 \)), and furthermore they oppose each other; thus \( H_1 \) cannot be rejected.

<table>
<thead>
<tr>
<th>Interruption style</th>
<th>1st Task</th>
<th>2nd Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negotiated (n=16)</td>
<td>13:26</td>
<td>3:40</td>
</tr>
<tr>
<td>Immediate (n=22)</td>
<td>8:34</td>
<td>4:49</td>
</tr>
</tbody>
</table>

Table 1: Mean number of minutes:seconds before participants entered their first assertion in each task.

Comprehension Scores

Given that participants were enticed to enter assertions at somewhat comparable times, was there a difference in how well they ultimately comprehended them? We measured each participant’s comprehension of assertions using seven comprehension questions on the post-session questionnaire.
H2: There will be no difference in the negotiated- and immediate-style participants' comprehension of assertions.

We expected that the immediate-style interruptions would facilitate learning, just as tutoring mechanisms introduce educational information at the very moment that information can be used to help solve a problem. However, our expectation was wrong. Instead, participants with negotiated-style interruptions answered an average of 67% of the comprehension questions correctly, significantly outperforming participants with immediate-style interruptions, who averaged 46% correct (Mann-Whitney, p=0.0153). Therefore, we reject H2.

Digging deeper into this result, Figure 3 shows the percentages of participants who answered each question correctly, grouped by interruption style. As the figure and accompanying Table 2 show, the entry-level features, such as entering assertions and understanding value violations, were understood approximately the same by both groups, but the propagation features—which are key in automatically identifying a formula’s bugs—were not understood very well by the immediate-style participants.
Figure 3: Participants who answered each comprehension question correctly: negotiated-style participants (dark bars) and immediate-style participants (light bars).

<table>
<thead>
<tr>
<th>Question number</th>
<th>Question content</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1, q2</td>
<td>Ability to recognize user-entered assertions and values being outside these ranges (shown as red circles in the environment).</td>
</tr>
<tr>
<td>q5, q6</td>
<td>Comprehension of the computer-guessed HMT assertions.</td>
</tr>
<tr>
<td>q3, q4, q7</td>
<td>Comprehension of assertions’ propagation through the dataflow chains formed by formulas, including conflicts between user-entered and system-generated assertions that could arise.</td>
</tr>
</tbody>
</table>

Table 2: Categorizations of the comprehension questions.

The negotiated-style participants’ superior comprehension of assertions is surprising, because the immediate-style participants had all the opportunities for learning given to the negotiated group—the immediate-style interruptions (an average of 46 per participant during the experiment) were in addition to the negotiated-style interruptions and tool tips. In fact, both groups looked at approximately the same number of tool tips: the negotiated-style group averaged 149 per user and the immediate-style group averaged 154 per user during the experiment. Due to the increased exposure, it would have been reasonable to expect participants with immediate-style interruptions to have a better comprehension of assertions.

A possible explanation for the difference in comprehension can be found in minimalist learning theory, which states that learning is enhanced when self-initiated [5]. The negotiated-style
interruptions have this property: if the user does not understand the interruption notification and
wants information about it, they must actively seek an explanation through tool tips. The
immediate-style interruptions, however, deviate from this property by forcing explanations on the
user without their requesting them.

Conjuring Up Accurate Assertions

The aim of helping users learn any new debugging device is, obviously, to enable the
participants to use the new device accurately. A measure corresponding to this aim in our study is
a participant’s ability to conjure up accurate assertions:

\[ H3: \text{There will be no difference in the accuracy of assertions entered by the negotiated- and}
\text{immediate-style participants.} \]

Interestingly, despite the differences in comprehension demonstrated above, the two
groups were identical in their accuracy. Participants with negotiated-style interruptions created
correct assertions 95% of the time—exactly the same percentage as the immediate-style
participants. Clearly, we do not reject H3. Note that this result does not agree with McFarlane’s
prediction that using negotiated-style interruptions would be better for the accuracy of the
interruption’s task (here, assertions).

To summarize, both groups entered assertions with equal accuracy. However, assertion
accuracy may or may not be “good enough” to support productive debugging—the comprehension
difference may play a critical role. Hence, we now investigate productivity directly.

RQ2: Debugging Productivity

Most previous research has found that immediate-style interruptions have a negative
impact on performance [1, 2, 7, 14, 16, 18], although generally, relevant interruptions do less harm
than irrelevant ones. But prior work has not addressed how relevant interruptions affect
performance when attempting to support cognitively complex tasks such as debugging.

Bugs Fixed

To evaluate participants’ debugging performance, we measured productivity by counting
the average number of bugs fixed per minute during each task. (This measure is used so as to
normalize the number of bugs fixed, because different spreadsheets had different numbers of bugs
and time limits, and the order spreadsheets were encountered as participants’ first/second task was
varied).

\[ H4: \text{There will be no difference in the negotiated- and immediate-style participants’ debugging}
\text{productivity on the first task.} \]
<table>
<thead>
<tr>
<th>Interruption style</th>
<th>Total bugs fixed</th>
<th>1st Task (Bugs per minute)</th>
<th>2nd Task (Bugs per minute)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negotiated (n=16)</td>
<td>13 (3.24)</td>
<td>0.202 (0.079)</td>
<td>0.264 (0.061)</td>
</tr>
<tr>
<td>Immediate (n=22)</td>
<td>11.18 (3.56)</td>
<td>0.210 (0.066)</td>
<td>0.163 (0.069)</td>
</tr>
</tbody>
</table>

Table 3. The mean (standard deviation) productivity. Significant differences (p<.05) are shaded.

H5: There will be no difference in the negotiated- and immediate-style participants' debugging productivity on the second task.

Table 3 presents the productivity of the negotiated- and immediate-style interruption groups. For Task 1, in which the learning curve was still a major factor, there was no significant difference (Mann Whitney, p=0.5059). We therefore cannot reject H4.

However, by the second task, the participants with negotiated-style interruptions were significantly more productive than the participants with immediate-style interruptions (Mann Whitney, p<0.0001). Therefore, H5 is rejected.

A critical goal of the surprise aspect to which the interruptions are tied is to entice users to explore portions of the program likely to contain bugs. McFarlane’s guidelines suggest that negotiated-style interruptions should result in the greatest efficiency and accuracy on the primary task (here, debugging). Our results agree. This can be clearly seen in Task 2 (i.e., after much of the initial learning curve has been overcome). Thus, the message for developers of end-user programming environments is that negotiated-style interruptions will result in the greatest efficiency on debugging.

How the Participants Spent Their Time

As previously mentioned, other researchers have established that there is a reorientation period after an immediate-style interruption [1, 2], and this should hold true in the case of debugging as well. But, is that the only reason for the productivity difference? A detailed look at the transcripts of participant activity showed some revealing behavior differences.

In particular, we examined the frequency with which participants performed the following four actions (which are the ones associated with debugging progress in our environment): editing formulas (to improve the “source code” or to manually enter different test values), entering assertions, using Help-Me-Test to automatically generate new test values, and checking off correct cell values.
Table 4 lists the average number of each type of action, as well as the number of total actions done by participants, on the first and second tasks. The first surprise is that the total number of actions done by the two groups was not significantly different. This is in contradiction with what we expected based on prior interruptions research [1, 2], from which we predicted that the cumulative effect of the reorientation penalties should have led to a decrease in the total number of debugging-related actions.

There were, however, significant differences in the participants’ choices of activities. By the second task, the negotiated-style participants were editing significantly more formulas than the immediate style participants were (Mann Whitney: p=0.0597 for Task 1 and p=0.0231 for Task 2). Also, although both groups performed testing activities using Help-Me-Test with approximately the same frequency on Task 1, by Task 2, the immediate-style participants used it significantly more (Mann Whitney, p=0.0406). These results suggest fundamental differences in participants’ strategies, a point we will pursue in the section discussing debugging strategies.

**RQ3: Debugging Self-assessment**

“Am I done debugging yet?” In the practice of software development, it is often this question that is used to decide whether a spreadsheet or other type of program is ready to use. Helping users make reasonable judgments to answer this question can be important in preventing software from going into use prematurely.

Using post-problem questionnaires, we asked participants to rate on a 1 (“not confident”) to 5 (“very confident”) scale, for each spreadsheet, how confident they were that they had corrected all the bugs. The issue relevant to debugging is to what extent these self-ratings were related to correctness of the spreadsheets. To investigate this, we compared self-ratings to actual performance.

*H6: There will be no difference between the negotiated- and immediate-style participants’ self-ratings as predictors of correctness (number of bugs in the spreadsheets at the end of the task).*

The regression analyses of the participants’ ability to predict how well they corrected the bugs are shown in Table 5. The regression coefficient is the slope of the least squares fitting of the ratings against the bugs that were corrected.
Table 4: Average number of each type of activity engaged in by the participants. Significant differences (p<.05) are shaded.

As Table 5 shows, the negotiated-style participants’ self-ratings were statistically significant predictors of actual performance in fixing bugs for both problems (regression analysis, p<0.05). The immediate-style participants’ self-ratings, on the other hand, were ineffective as predictors, and their regression coefficients were not significantly different from zero.

This result supports McFarlane’s prediction that immediate-style interruptions would harm users’ assessment of accuracy on the primary task. It also implies that immediate-style interruptions in an end-user programming environment will interfere with users’ judgment of when they have found all the bugs.

**Discussion – Impacts on Debugging Strategies**

Recall from Table 4 that there were significant differences in the ways the participants spent their time. These differences strongly suggest different debugging strategies.

<table>
<thead>
<tr>
<th>Interruption style</th>
<th>Regression coefficient</th>
<th>T-value</th>
<th>Signif.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Grades:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negotiated (n=16)</td>
<td>1.214</td>
<td>2.251</td>
<td>0.0410</td>
</tr>
<tr>
<td>Immediate (n=22)</td>
<td>0.240</td>
<td>0.590</td>
<td>0.9534</td>
</tr>
<tr>
<td><strong>Weekly Pay:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negotiated (n=16)</td>
<td>0.683</td>
<td>2.650</td>
<td>0.0190</td>
</tr>
<tr>
<td>Immediate (n=22)</td>
<td>0.301</td>
<td>1.420</td>
<td>0.1711</td>
</tr>
</tbody>
</table>

Table 5: Regression analyses of actual bugs corrected vs. perceived bugs corrected.
Indeed, there is precedent for interruption style impacting users' strategies. Both Hess and Detweiler [9] and McFarlane [14] found that users with different interruption styles developed different strategies for engaging in their primary task. But, in what directions might different interruption styles steer debugging strategies?

Immediate-style interruptions have been found to have a disruptive effect on users' short-term memory [1, 2], which could impact users' strategy choices. There is research establishing the importance of short-term memory to debugging. For example, in Ko and Myers's recent empirical work, 30% of their participants' debugging breakdowns were tied to attentional problems such as loss of situational awareness or working memory strain [11].

In light of this background, it appears likely that the immediate-style participants were subjected to frequent losses of the short-term memory contents they had built up. Our theory is that, because of these losses, the participants avoided debugging strategies that had high short-term memory requirements.

Consider the data. What immediate-style participants did significantly less of was editing formulas. Editing a non-constant formula is generally a sign that a user believes they have found a bug. Although finding a bug can occasionally be done by looking at just one cell, often users must consider multiple related cells in the dataflow chain. Editing a constant formula is the way users set up a test. To make "testedness" progress, this requires considering dataflow relationships in subexpressions to figure out a value that will help increase a partially tested cell's testedness. Thus, the users' considerations for both non-constant and constant formula edits can require extensive use of short-term memory.

Compared to participants with negotiated-style interruptions, participants with immediate-style interruptions made significantly more use of Help-Me-Test, ultimately checked off more values (but not to a significant extent), and made equal use of assertions. All of these operations are highly local. To invoke Help-Me-Test, one simply pushes a button and waits for new test values; to check off a value, one considers that value and the original inputs and makes a decision; placing an assertion involves reasoning about only one cell. Thus, these three devices do not require users to keep much in their short-term memory.

If our theory is correct, then the implications for using immediate-style interruptions in end-user debugging are profound: namely, they will promote over-reliance on local, shallow, problem-solving strategies.

**CONCLUSION**

In investigating the effects of interruptions on helping end-user programmers debug, we expected to find advantages from each style of interruption. For example, we expected to see
better productivity at bug finding with negotiated-style interruptions but better learning with immediate-style interruptions (because of its common ground with on-line tutoring). Instead, we found advantages for only the negotiated style.

In particular, the following results were unexpected:

- Immediate-style interruptions did not promote learning as well as the negotiated style, as seen by the participants’ comprehension scores. This was despite the fact that immediate-style participants received more explanations, which were timed to arrive at a pertinent moment.
- Examination of users' activities suggests that immediate-style interruptions may have promoted over-reliance on shallow, local strategies that have low short-term memory loads.
- The results that agreed with those of interruptions research in simpler domains boil down to these two points:
  - The negotiated-style participants were significantly more productive at debugging after the initial learning curve climb (i.e., by the second task).
  - The negotiated-style participants were reasonably effective at assessing how well they had succeeded at fixing all the bugs, whereas the immediate-style participants were not.

What do these results say to designers of end-user programming environments? Negotiated-style interruptions were more effective than immediate-style interruptions regardless of whether the aim was to alert the user to the presence of program errors or to introduce the user to new debugging features. The participants were more effective even given that the debugging strategy in the experiment was based on using the element of surprise to attract the user's attention! Such strong results send a clear message to designers of end-user programming environments: resist the temptation to use immediate-style interruptions to "help" users find bugs. We found no reasons to use immediate-style interruptions, and several reasons not to.

ACKNOWLEDGMENTS

We thank Douglas Derryberry for helpful insights into this work. This work was supported in part by NSF under ITR-0082265 and in part by the EUSES Consortium via NSF's ITR-0325273.

REFERENCES


IMPACT OF HIGH-INTENSITY NEGOTIATED-STYLE INTERRUPTIONS ON END-USER DEBUGGING

TJ Robertson

Journal of Visual Languages and Computing
Block A2 Westbrook Centre Milton Road
Cambridge, CB4 1YG, UK
Submitted 2/18/2004
Impact of High-Intensity Negotiated-Style Interruptions on End-User Debugging

TJ Robertson
Oregon State University
tjuggle@yahoo.com

Correspondence To above email address
ABSTRACT

Extending our previous work in [17] we delve deeper into the question of which interruption style best supports end-user debugging. Previously, we found no advantages of immediate-style interruptions (which force the user to divert attention to the interruption at hand) over negotiated-style interruptions (which notify them without actually preventing them from working) in supporting end-user debugging. In this study, we altered our negotiated-style interruptions from [21] (which were shown to help end-user debuggers learn about and use debugging features of our programming language) such that they were more intense (larger, blinking, and/or accompanied by text).
INTRODUCTION

Unlike most research on end-user programming, which focuses on the creation of new programs, our research team has been focusing on supporting aspects of software engineering for end-user programmers. Recently, our group and other researchers have begun to focus on helping end-user programmers debug their programs. Most systems that aim to help end users debug programs have to notify the users of information relevant to debugging (e.g. [9,14,15,18]). The mechanism for communicating such information, can have a significant impact on the systems effectiveness. We have been investigating the appropriateness of various methods of communicating this information to users.

In [17], we presented our findings from a study comparing the effects on end users of two styles of interruptions. We studied the effects on the end users in terms of their ability to learn the debugging features of the software, their ability to fix bugs, and their ability to accurately rate how successful they were at fixing bugs. The two debugging styles compared were negotiated-style interruptions and immediate-style interruptions, according to McFarlane’s classification of interruptions [12]. Negotiated-style interruptions are interruptions that inform the user of a pending message, but do not force them to acknowledge the message immediately. (An example of one used in our study is a red circle appearing around possibly incorrect values in spreadsheet cells, thereby letting the user know that they should check that cell for errors.) Immediate-style interruptions are interruptions that must be acknowledged by the user. (In our study, we used modal pop-up “ok” boxes that displayed messages.)

Our study in [17] concluded that negotiated-style interruptions were superior to immediate-style interruptions in all aspects studied (participants’ learning of debugging features, fixing bugs, and predicting their performance). Additionally, analysis of the actions in which the different participants engaged suggests that the immediate-style interruptions may have encouraged over-reliance on shallow debugging strategies that had low short-term memory requirements.

This paper extends [17] in that we delve deeper into the question of how to interrupt end-user programmers in order to help them better debug their programs. We have already shown that negotiated-style interruptions were superior to immediate-style interruptions in every metric we used. Now we wish to explore how the intensity of the negotiated-style interruptions affect end-user programmers. In this paper, we present the
results of a study in which we compare the previous participants with a new group of participants. The new participants had much more intense negotiated-style interruptions, i.e., the interruptions had more intense characteristics such as being larger, blinking, and being accompanied by text. We will refer to these more intense forms of negotiated-style interruptions as high-intensity negotiated-style interruptions. In contrast, our original negotiated-style interruptions (such as the red circle around cell values) will be referred to as low-intensity negotiated-style interruptions.

Because negotiated-style interruptions can range from barely noticeable to extremely hard to ignore, we felt that it was not sufficient to simply endorse negotiated-style interruptions (as in [17]). This study will therefore compare the higher intensity negotiated-style interruptions with both the lower intensity negotiated-style interruptions and the immediate-style interruptions. In doing so, we hope to paint a clearer picture of the possible effects on users of negotiated-style interruptions.

For this study, our research question is as follows.

What are the effects on end-user programmers (in terms of learning debugging features, debugging productivity, and ability to predict debugging performance) of high-intensity negotiated-style interruptions as compared to both low-intensity negotiated-style interruptions and immediate-style interruptions?

RELATED WORK

Researchers have recently begun to study aspects of end-user programming beyond just the programming stage; examples include [9, 10, 15, 20]. McFarlane created a classification for interruptions in which he described four styles of interruptions, each with associated strengths and weaknesses [12]. Negotiated interruptions alert the user of pending notifications, but do not force them to acknowledge the notification immediately. Immediate interruptions prevent the user from continuing with their work until the interruption is dealt with. Mediated interruptions present information when the system decides it is an appropriate time to interrupt the user. Scheduled interruptions present interruptions at fixed time intervals. In [17] our group found negotiated-style interruptions to be better for assisting end-user debugging than immediate-style interruptions in all metrics we measured.

It has been shown, however, that the properties of negotiated-style interruptions can have significant effects on users. In a study of peripheral scrolling text displays [13], it was found that displays that scrolled continuously were more distracting than displays
that only scrolled in new information when it was available. In addition, it was found that the information remembered by users was the same in either case.

Berlyne explains that the intensity variables (traits such as size and color) will affect the amount of arousal (a psychological term which essentially refers to the amount of mental stimulation a person is experiencing) [1]. Thus our high-intensity negotiated-style interruptions are likely to evoke high arousal in our participants. Furthermore, arousal research [5] shows that people have an optimal level of arousal, and they will adjust their activities to reach this optimal level. Exceeding this level can result in the user becoming desperate to get rid of the arousal evoking stimuli. Thus, though high-intensity interruptions are negotiated-style interruptions, it may be so hard to ignore them that they pull the user from their work just as an immediate-style interruption would. If this is the case, then we should expect their performance to suffer, just as the performance of the participants with immediate-style interruptions suffered in [17].

It has been shown that waiting until the user has completed certain tasks is less disruptive than interrupting them in the middle of the tasks [4]. Indeed, it was found that in a data entry task, an immediate-style interruption on screen was more distracting than a phone call or walk-in visitor (both of which can be considered negotiated-style interruptions, because the user can continue working for a brief period of time until they get to an appropriate stopping point) [19]. A potential problem with high-intensity negotiated style interruptions is that they may disrupt the user (cause them to reach overly high arousal levels) at inopportune times.

On the other hand, new work [8] shows that software that receives data from a relatively simple set of sensors can determine the best times to interrupt users. So intelligent timing of such high-intensity interruptions may mitigate such a problem.

It is interesting to note that even low-intensity negotiated-style interruptions work on the principle of raising arousal in order to evoke action from the user. This study is intended to provide some data showing what the effects are at high intensity levels.

**EXPERIMENT**

We conducted a controlled laboratory experiment with three groups of end-user participants. For brevity, throughout this paper we will refer to our different groups of participants as high-intensity, low-intensity, and immediate-style participants.
Design, Procedures, and Tasks

We replicated the experiment done in [21] and [17], except for the way in which interruptions were presented. In this study, we modified the environment from [21] so that the negotiated-style interruptions were more intense.

We used the 16 participants from [21] as our low-intensity group and the 22 participants from [17] as our immediate-style group, and added in a new group of 15 participants for our high-intensity group. As in earlier studies, participants responded to an invitation given to business majors who were at least sophomores and who had prior spreadsheet experience. This study, however, occurred during Fall term, rather than during summer term as the other two studies did.

To test for differences in our participants that might bias our results, we administered and analyzed a background questionnaire. No difference was found in the participants' confidence in their ability to create a spreadsheet. There was also no difference found in the participants' GPAs. Additionally, these participants were all business majors, as were most participants in previous studies. However, we uncovered two differences that could threaten the validity of this study. First, the high-intensity participants had less professional spreadsheet experience than the low-intensity participants (Mann-Whitney p=0.0418) and less total spreadsheet experience than the immediate-style participants (Mann-Whitnery p=0.0370). Second, the high-intensity participants tended to be of junior standing, whereas the other participants tended to have senior standing (Mann-Whitney p=0.0023 for high-intensity versus low-intensity participants and Mann-Whitney p=0.0295 for high-intensity versus immediate-style participants).

We gave the participants a 25-minute tutorial introducing them to the spreadsheet language Forms/3. After the tutorial, they were asked to debug two spreadsheets, Grades and Weekly Pay, with time limits of 35 and 22 minutes respectively; see [2, 21] for details of these spreadsheets. The time limits were to ensure that all participants worked on both spreadsheets, to avoid possible peer influences of some participants leaving early, and to ensure consistent treatment of all participants. Roughly half of the participants did Grades first and the other half did Weekly Pay first. This was done so as to evenly distribute learning effects over the problems.

The participants were given problem descriptions that described what the spreadsheet was to accomplish, and what individual cells were for. The problem description instructed participants to “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."

All user actions were recorded in electronic transcripts for later review. After each problem, participants were given a questionnaire in which they rated how well they thought they had debugged the spreadsheet. At the end of the study, every participant answered questions that tested their understanding of the debugging feature that pertained to our high-intensity interruptions.

**Forms/3 and Surprise-Explain-Reward**

This study was conducted using the research spreadsheet language Forms/3 [3]. One of the features of Forms/3 is assertions on spreadsheet cells, which past empirical work has shown that end users can use effectively [2, 6]. These assertions are the debugging device to which our high-intensity negotiated-style interruptions pertain.

Assertions are represented as allowable ranges for a cells' value. When the user creates an assertion (termed a *user-entered assertion*), it is propagated through the dataflow chain of the spreadsheet (creating *system-generated assertions*), so that cells have assertion ranges if the cells that they reference have assertion ranges. When an assertion range is violated, a red circle is drawn around the cell's value; such a violation is termed a *value violation*. When a system-generated assertion conflicts with a user entered assertion, a red circle is drawn around the two conflicting assertions; such a conflict is termed an *assertion conflict*.

For example, in Figure 1, the user has entered an assertion for `input_temp`, which propagated through `output_temp`'s formula to create a system-generated assertion. Since the values “200” and “33.3333” do not fulfill their cells' assertions, they are...
circled. Finally, the user also entered an assertion “0 to 100” for output_temp; since it disagrees with the cell’s other assertion, they are both circled.

Besides assertions, participants had other debugging devices available. If they decided a cell’s value was correct, they could check it off in the corner of each cell (e.g., the checkbox in Figure 1’s output_temp cell). This resulted in an incremented “testedness” indicator in the environment, and also changed the tested cells’ border colors toward blue along a red-blue continuum to indicate increased testedness. If they wanted help conjuring up more test inputs, participants could push a Help-Me-Test button to automatically generate more values [6].

*Surprise-Explain-Reward* is a strategy we use to introduce users to new features that may benefit them, such as assertions. Help-Me-Test’s role in our experiment was in its use as a springboard by the Surprise-Explain-Reward strategy for introducing users to assertions. Here is how this springboard works: When a user invokes Help-Me-Test, the system not only generates values for input cells, but also creates a (usually incorrect) “guessed” assertion to place on these cells. These guessed assertions, termed *HMT assertions* (because they are generated by Help-Me-Test), are intended to surprise the user into becoming curious about assertions. They can satisfy their curiosity using tool tips, which will inform them of the benefits and syntax of assertions. If the user follows up by accepting an HMT assertion (either as guessed or after editing it), the resulting assertion will be propagated as in Figure 1. As a result, value violations or assertion conflicts may occur; if so, red circles will appear as in Figure 1, which are often another surprise. All of these attempted surprises are communicated via interruptions.

**High-Intensity Negotiated-Style Interruptions**

In [21], the mechanisms for communicating surprises to the user came in the form of low-intensity interruptions. Surprise-Explain-Reward with these low-intensity interruptions was found to be beneficial to end-user programmers [21]. In [17], the low-intensity interruptions were accompanied with immediate-style interruptions in the form of pop-up “ok” dialog boxes that contained the same text as the tool-tip.

In this study, we substituted high-intensity interruptions for the low-intensity interruptions and removed the experimental immediate-style interruptions. Recall that by intensity, we refer to properties of the interruption. Figure 2 shows each property that was manipulated.
Figure 2: What the high-intensity interruptions look like (top row) versus low-intensity interruptions (bottom row). Explanations below:

<table>
<thead>
<tr>
<th>Event Description</th>
<th>Low-intensity</th>
<th>High-intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 When the user shows interest in assertions (by clicking the button to make an</td>
<td>Stick figure next to guard</td>
<td>Large stick figure and explanation</td>
</tr>
<tr>
<td>assertion visible, or deleting an HMT assertion)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 When there are value violations</td>
<td>Red circle around value</td>
<td>Blinking red circle and cell background</td>
</tr>
<tr>
<td>3 When Help-Me-Test generates assertion(s)</td>
<td>Help icon with assertion range</td>
<td>Large icon with blinking background on assertion</td>
</tr>
<tr>
<td>4 When there is an assertion conflict</td>
<td>Red circle around assertion icons</td>
<td>Blinking red circle around enlarged icons</td>
</tr>
<tr>
<td>5 When system-generated assertion(s) are created</td>
<td>Computer icon with assertion range</td>
<td>Larger icon with temporarily blinking background on assertion range</td>
</tr>
</tbody>
</table>
RESULTS

Learning Results

In this section, we will examine the effects of high-intensity interruptions on users' ability to learn the debugging features of the software. We examine this because much of the research in end-user programming focuses on guiding users to find bugs [18,15]. This necessarily involves teaching them about the debugging features of the software.

We analyzed the participants' post-session questionnaire scores for questions that test participants' understanding of assertions. In [17] it was found that immediate-style participants answered fewer questions correctly, and furthermore, the questions that posed a problem for them were ones that tested their understanding of advanced assertion features (specifically propagation of assertions through dataflow chains). The deficit in understanding advanced features is important, because those features are the ones that will aid users by automatically identifying software errors.

To test for differences in comprehension between the high-intensity versus low-intensity and immediate-style participants, we formulated the following null hypotheses.

Hypothesis H1: There will be no difference in the high-intensity and low-intensity participants' comprehension of assertions.

Hypothesis H2: There will be no difference in the high-intensity and immediate-style participants' comprehension of assertions.

As discussed in the related work section, the high-intensity interruptions may affect users similarly to the immediate-style participants. If so, they should result in a lower comprehension of assertion features (just as they did the immediate-style participants).

Indeed, the high-intensity participants did perform very similarly to the immediate-style participants. The high-intensity participants correctly answered 43% of assertion related questions, compared to 67% for the low-intensity participants and 46% for the immediate-style participants. Thus, we cannot reject Hypothesis H2 (Mann-Whitney p=0.6933). Just as with the immediate-style participants, the high-intensity participants were found to have significantly lower comprehension than the low-intensity participants (Mann-Whitney p=0.0080). Therefore we reject Hypothesis H1.
Observe from Figure 3 and Table 1 that the high-intensity participants’ performance in answering assertion-related questions mirrored that of the immediate-style participants. As in [17], they had trouble understanding the advanced features of assertions, including propagation of assertions and HMT assertions. The lack of knowledge about HMT assertions is interesting, because in this study, the high-intensity participants had HMT assertions that blinked (presumably increasing their awareness of them). As we will address in the discussion section, the blinking may have caused them to care less about learning the reason behind assertions existence, and more about getting them to go away.

First Assertion Time

Surprise-Explicit-Reward’s goal of teaching users how to use the debugging features of software hinges on its ability to draw the users’ attention to those features. To evaluate the ability of high-intensity interruptions to draw users to use assertions, we will examine the average time that it took for participants to enter assertions under three
different conditions. The three conditions were: first task, second task, and after they started Help-Me-Test for the first time (on the first task).

Times for the first and second tasks were calculated by taking the difference between the time they entered their first assertion and the time they made their first action. As Table 2 shows, the immediate-style participants had the fastest time to enter an assertion on the first task. The immediate-style participants were significantly faster at creating their first assertion than the low-intensity participants (Mann-Whitney p=0.0288) \(^1\). The high-intensity participants appeared to perform just slightly slower than the immediate-style participants, though they were not quite significantly faster than the low-intensity participants at entering their first assertion on the first task (Mann-Whitney p=0.0580). On the second task, the time differences were not very pronounced, and there were no significant differences in any participants’ times.

On the first task, we checked for differences between the first time participants used Help-Me-Test and the first time they entered an assertion. This is because Help-Me-Test is Surprise-Explain-Reward's gateway to the interruptions that we manipulated in this study. The delays between starting Help-Me-Test and entering their first assertion were not significantly different for any of the participants (all Mann-Whitney p>0.60).

<table>
<thead>
<tr>
<th>Average time to first assertion</th>
<th>1st Task</th>
<th>2nd Task</th>
<th>After Help-Me-Test (1st Task)</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-intensity</td>
<td>7:33</td>
<td>6:25</td>
<td>4:42</td>
</tr>
<tr>
<td>Low-intensity</td>
<td>13:26</td>
<td>5:54</td>
<td>3:44</td>
</tr>
<tr>
<td>Immediate</td>
<td>7:10</td>
<td>5:01</td>
<td>4:04</td>
</tr>
</tbody>
</table>

Table 2: Average time until users entered their first assertion on both tasks, as well as after they started Help-Me-Test for the first time (on the first task). Times after Help-Me-Test were calculated by considering only participants who used Help-Me-Test and then assertions.

\(^1\) There is another way of scoring the immediate-style participants’ first assertion time, and that is because they had to hit an “experiment start” button, which could be considered to be the first action they took. If that scoring method is used, 8:34 seconds is their start time (as was reported in [17]). The alternative scoring method for the immediate-style subjects, however, is inconsistent with the way first assertion times were graded for the rest of the participants.
Conjuring Up Accurate Assertions

As part of assessing how well participants learned how to use assertions, we will examine how accurate they were at creating correct assertions. This is an important metric, as the goal of Surprise-Explain-Reward is to teach users how to properly use debugging features. Additionally, creating correct assertions is crucial because when they propagate (regardless of whether users understand how they propagate) through the dataflow chain, the red circles and the propagated assertions will only be correct if the original was correct.

_Hypothesis H3: There will be no difference between the percentage of assertions created correctly by high-intensity participants versus the low-intensity and immediate-style participants._

As can be seen in Table 3, the high-intensity participants appeared to be slightly less accurate than the low-intensity participants and immediate-style participants (which have approximately equivalent accuracy at creating assertions). However, none of the participants were significantly different from each other in terms of assertion accuracy. Thus we cannot reject hypothesis H3.

<table>
<thead>
<tr>
<th>Percent of correct assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-intensity</td>
</tr>
<tr>
<td>Low-intensity</td>
</tr>
<tr>
<td>Immediate</td>
</tr>
</tbody>
</table>

Table 3: The percentage of correct assertions that participants created and the percentage of cells with correct assertions

Debugging Productivity

In [17] we found that the participants with low-intensity negotiated-style interruptions were significantly more effective at fixing bugs than the immediate-style participants. The significant difference was found to be on the second task (after the learning phase of the first task in which there no significant difference).

Again, we propose that the high-intensity interruptions will affect participants similarly to the way that immediate-style interruptions will affect them. Thus we expect
them to perform roughly the same as immediate-style participants of [17]. To investigate, we use the following null hypotheses.

_Hypothesis H4:_ There will be no difference in the high-intensity and low-intensity participants’ debugging productivity on the first task.

_Hypothesis H5:_ There will be no difference in the high-intensity and immediate-style participants’ debugging productivity on the first task.

_Hypothesis H6:_ There will be no difference in the high-intensity and low-intensity participants’ debugging productivity on the second task.

_Hypothesis H6:_ There will be no difference in the high-intensity and immediate-style participants’ debugging productivity on the second task.

Table 4 shows the debugging productivity of all three groups of participants and the Mann-Whitney p-values comparing the high-intensity participants with the other two groups of participants. As can be seen from the table, the only statistically significant result was on the second task. On the second task, the high-intensity participants performed significantly worse than the low-intensity participants (Mann-Whitney p=0.007) at fixing bugs. As a result of these data, we reject only Hypothesis H6. In conclusion, it appears that the high-intensity participants had roughly the same debugging performance as the immediate-style participants.
How The Participants Spent Their Time

In [17] we attempted to understand immediate-style participants’ lower scores at fixing bugs by examining the types of actions they chose to do. We found significant differences in the actions they engaged in, and those differences suggested fundamental differences in their debugging strategies.

There was a mistake in [17], in that the 2nd task debugging performance for immediate-style participants was reported as 0.163. This was incorrect; the actual number is 0.263, but it does not change the statistical significance reported in that paper.
<table>
<thead>
<tr>
<th>Interruption style</th>
<th>1st Task</th>
<th>2nd Task</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Edit formula</td>
<td>Edit assertion</td>
</tr>
<tr>
<td>High-intensity</td>
<td>12.07</td>
<td>11.27</td>
</tr>
<tr>
<td>Low-intensity</td>
<td>17.81</td>
<td>11.19</td>
</tr>
<tr>
<td>Immediate-style</td>
<td>12.73</td>
<td>12.68</td>
</tr>
<tr>
<td>High-intensity vs</td>
<td>p=0.0546</td>
<td></td>
</tr>
<tr>
<td>Low-intensity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Mann-Whitney)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Average number of each type of activity engaged in by the participants. Significant differences (p<.05) between the low-intensity and high-intensity participants are shaded.

As can be seen from Table 5, there were significant differences in activities engaged in by high-intensity participants compared to low-intensity participants. On the second task, the high-intensity participants edited significantly more assertions and used Help-Me-Test significantly more than the low-intensity participants. This differs from the immediate-style participants who did approximately the same number of assertion edits, but did significantly less formula edits than the low-intensity participants.

The more frequent use of debugging features during the second task (after the majority of learning effects have occurred) tells us that the high-intensity interruptions encouraged participants to make use of the debugging features. This is interesting because we have shown that they were less knowledgeable of the features and less effective at debugging compared to the low-intensity participants. We will address the more frequent use of assertions in the discussion section. As for the more frequent use of Help-Me-Test, it is possible that an effect similar to what we believe occurred to the immediate-style participants of [17] occurred. Namely, the high arousal state which the
high-intensity interruptions induced in the participants encouraged them to use tools such as Help-Me-Test which do not require as much mental effort, or short-term memory load. Indeed, there is psychological research showing that under high arousal, people’s short-term memory can suffer [5].

**Debugging Self-Assessment**

With end-user programs being used for important calculations, it is important that these programs not go into use prematurely. In practice, end-user programmers are notorious for overestimating their ability to create correct software [16]. For this reason, we will consider the participants’ abilities to evaluate their accuracy at debugging spreadsheets.

After participants completed each problem, we administered a questionnaire that asked them to rate how confident they were that they had corrected all the bugs in the spreadsheet. Their choices ranged from 1 (“not confident”) to 5 (“very confident”). In [17] it was seen that the low-intensity participants’ self-ratings were significant predictors of actual performance at fixing bugs whereas the high-intensity participants’ scores were not. We now investigate how well the high-intensity participants were able to judge their debugging performance.

Hypothesis H7: The high-intensity participants’ self-ratings will not be significantly correlated with their ability to fix bugs on a per problem basis.

Table 6 shows the regression analysis of the participants’ ability to rate how well they corrected bugs for both problems. As the table shows the high intensity participants’ self-ratings were not significant predictors of their performance at fixing bugs (unlike the low-intensity participants). Thus, we cannot reject Hypothesis H7.

<table>
<thead>
<tr>
<th>Grades</th>
<th>Linear Regression p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-Intensity</td>
<td>0.0410</td>
</tr>
<tr>
<td>High-intensity</td>
<td>0.966392</td>
</tr>
<tr>
<td>Immediate-style</td>
<td>0.1194</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weekly Pay</th>
<th>Linear Regression p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-intensity</td>
<td>0.0190</td>
</tr>
<tr>
<td>Immediate-style</td>
<td>0.1711</td>
</tr>
<tr>
<td>High-intensity</td>
<td>0.3383</td>
</tr>
</tbody>
</table>

Table 6: Linear regression p-values for Number of bugs fixed vs belief in how successful they were at fixing all bugs.
DISCUSSION – AGITATED USERS

Theories on arousal suggest that if a users' arousal exceeds an optimum amount, the user will resort to avoidance strategies, in which they attempt to make the stimuli go away [5]. This is different from a more constructive state of curiosity, which might compel them to explore the surprise and learn how it works [11]. Rather, in such an overly aroused state, a user may be satisfied with any action that causes the arousing stimuli to go away. It seems likely that some of our high-intensity participants were in just such a state.

As noted earlier, the high-intensity participants edited significantly more assertions in the second task than other participants. Indeed, the assertion edits on the second task reveals some interesting information about the high-intensity participants.

On the second task, all of the high-intensity participants except for two edited at least 13 assertions with a maximum of 25. By contrast, the low-intensity and immediate-style participants had an average of slightly less than 11 assertion edits each. What's remarkable is that the two high-intensity participants who did not perform double-digit numbers of assertion edits performed zero assertion edits (though they did use Help-Me-Test about as frequently as the other participants, so they were exposed to blinking HMT assertions). That is, the high-intensity participants appeared to be polarized in that they performed either many assertion edits, or zero assertion edits.

It is trivial to see how participants performing large numbers of assertion edits are compatible with the theory that they were experiencing an excess of arousal (and thus avoidance behavior). They found that editing assertions was a way to make the blinking HMT assertions go away. But what about the participants with zero assertion edits? It was noticed during pilot tests and during the administration of the experiment that some participants found a way to hide all of the blinking HMT assertions by pressing a “Close HMT” button that appears whenever Help-Me-Test is invoked. It is possible that some participants learned to press Help-Me-Test to get new test values and then press the “Close HMT” button (which would leave the new test values in place) to make all the blinking HMT assertions go away. Unfortunately, pressing this button was not recorded in our electronic transcripts, so we have no way of knowing whether the two participants with zero assertions used this strategy.

Another bit of data that supports the theory that participants may have been avoiding the high-intensity interruptions is the performance of one of our high-intensity participants. This participant edited 21 assertions, but every one of them was blank. This means that 21 times the user highlighted an HMT assertion, deleted the text (e.g. “3 to
14"), and hit the "Apply" button. This participant clearly had little concept of the purpose of the assertions, though they were very vigilant about editing them. Indeed, this user only got the simplest two questions (out of seven) correct on the post session questionnaire. Thus, the most obvious explanation is that this user was trying to get the HMT assertions to stop blinking (and was not aware of the "Close HMT" trick), so they individually deleted 21 HMT assertions (out of a total 22 HMT assertions with which they were presented).

If these speculations are correct then it could mean that high-intensity interruptions have the serious drawback of causing users to deal with them by any means possible (and possibly not in a way which is productive).

**THREATS TO VALIDITY**

Here we will address potential threats to the validity of our results.

Threats to internal validity are unintended factors that may have influenced our results. To provide a uniform treatment for participants, we used a tutorial script (though for the immediate-style participants, the script was read by a non-native English speaker) and written problem descriptions which were the same for all participants. We did have to rearrange the spreadsheet layouts slightly for the high-intensity participants so that the high-intensity interruption mechanisms (such as the larger icons) would fit onto the screen without overlapping with other spreadsheet cells. While rearranging them, however, we made every attempt to hold the relative locations of cells constant. On the Grades spreadsheet, unfortunately, high-intensity participants had to scroll down to see all the cells whereas the other participants did not. To ensure that our participants were uniform, we sent out an email to business students asking for a specific set of characteristics that were the same as for the earlier participants. Additionally, we asked the participants for background information so that we could analyze the data for differences. A possible source of bias was that the high-intensity study was done during fall term, whereas the other studies were done during the summer. The difference between fall term and summer school students may not be negligible (for example, the summer school students may have been older than their fall term counterparts). Indeed, as mentioned earlier, we did find significant differences in seniority and participants’ prior experience creating spreadsheets.

Construct validity refers to the question of whether the results are based on appropriate information. We used electronic transcripts to record user actions, so that there would be no question of what participants did (though as mentioned in the
Discussion, the transcripts did not record when participants pressed the “Close HMT” button. For most of our statistical data, we wrote tools to automatically harvest data from the transcripts for use in statistical comparisons. Additionally, we tended to use multiple choice questions rather than free response questions in order to avoid ambiguity in participants’ responses. Sometimes, however, participants would write in a free response answer because they did not realize that the questions were multiple choice. This may have lead to some bias, because we had to choose the closest multiple choice answer to what they wrote.

External validity relates to how well the results can be generalized. About 50% of the participants in each group had at least some professional spreadsheet experience, and we discovered that significant differences in our data were primarily due to these participants. This means that our results should be applicable to spreadsheet developers in the business world. However, because our study only lasted about 1.5 hours, the results may not hold for long-term usage. Indeed, it has been found [7] that when people train on a system with immediate-style interruptions, they are better equipped to handle working with those immediate-style interruptions. Thus it is possible that the effects of high-intensity/immediate-style interruptions may become less negative with time.

CONCLUSION

To further investigate the question of how interruption styles affect end-user programmers’ abilities to effectively learn and use the debugging features of a programming language, we have extended [17] to examine user performance when high-intensity negotiated-style interruptions are used. As expected, the high-intensity interruptions seemed to affect the users the same way as did immediate-style interruptions. However, we caution that these results should only be considered in light of the threats to validity discussed above.

Specifically, when compared to users with low-intensity interruptions:

- High-intensity interruptions impaired the users’ ability to learn the debugging features of the language.
- Participants with high-intensity interruptions were less effective at debugging.
- Participants with high-intensity interruptions were not effective at judging their ability to debug.
Additionally, psychological theories, as well as user actions indicate that high-intensity interruptions may push the user into an uncomfortable state of mental arousal. This highly aroused mental state may have driven users to engage in any activity that would make the interruptions go away regardless of whether those actions were productive in terms of learning the debugging devices or improving the software.

What do these results tell the designers of end-user programming environments? Not only is it important to use negotiated-style interruptions when attempting to assist end-user programmers in learning and using debugging features, but the nature of the negotiated-style interruptions must also be considered. In particular, negotiated-style interruptions that are very intense should be avoided, as they can have similar effects on end users as immediate-style interruptions.

ACKNOWLEDGMENTS

Dr. Douglas Derryberry in Oregon State University’s psychological department provided help in finding relevant psychological research, Dr. Margaret Burnett in Oregon State University’s Computer Science was the faculty advisor for this study and as assisted in editing this paper.
REFERENCES


GENERAL CONCLUSION

With our two studies, we compared three variations of interruptions within the context of Surprise-Explain-Reward in terms of their effects on end-user programmers' ability to learn about debugging features, ability to successfully debug spreadsheets, and their ability to assess how accurate they were at debugging the spreadsheets. The interruption variations we compared were high-intensity negotiated-style interruptions, low-intensity negotiated-style interruptions, and immediate-style interruptions.

We found that low-intensity negotiated-style interruptions were more effective than the other two styles of interruption according to all metrics we examined. Additionally, by showing that high-intensity negotiated-style interruptions have the same drawbacks as immediate-style interruptions, we have shown that not only are immediate-style interruptions harmful to end-user programmers performance, but negotiated-style interruptions can be just as problematic if they are of a very high intensity.

The message to designers of end-user programming languages is that when they wish to help users learn and effectively use debugging features of software, they should avoid distracting end-user programmers with immediate-style or high-intensity negotiated-style interruptions. Instead, they should rely on less distracting, low-intensity, negotiated-style interruptions.

The message to researchers is that negotiated-style interruptions are not silver bullets. Rather, care must be taken in choosing how to present the negotiated-style interruptions to user.


APPENDICES
APPENDIX A – BIBLIOGRAPHIC NOTES

In preparing for this thesis, I read a number of papers related to arousal and curiosity, as well as interruptions in the workplace (with an emphasis on computers). This section contains notes from my readings (some of the notes, particularly ones related to interruptions, have been cut because they are covered in the thesis papers).

Additionally, based on the notes, a hypothesis on how negotiated-style interruptions can be classified is proposed that has potential to become the basis for other studies. Particularly, this hypothesis predicts that negotiated-style interruptions can fall into four categories (based on intensity levels). Each intensity level is expected to have different effects on users (based on how it affected their arousal levels). Exploration of this topic could lead to enough knowledge to make suggestions as to what types of interruptions (out of the 4 intensity levels) are appropriate for various goals in end-user programming environments.

Surprise Explain Reward’s Place In Other Work

It has been suggested [Carroll 1987] that users are motivated to produce a product rather than to learn a new software system. Even when they do undertake the task of learning software, they do better with hands on exercises rather than reading a linear manual. Thus it has been argued that computer programs should support learning by an active user who desires to produce a product rather than to train. Surprise Explain Reward teaches and notifies the user while they are using the system. Rather than requiring the user to read from a manual to discover the software engineering features of the programming language, the software attempts to intrinsically motivate the user to learn about features through their own curiosity. Consequently, it conforms to the principle of supporting active users.

Mixed Initiative User Interfaces [Horvitz 1999] are an attempt to bridge the two popular goals of modern HCI research. One of the goals is automated services, which are services which automatically perform some task for a user. The other goal is giving the user better access to the software features. Mixed Initiative Interfaces attempt to bridge these two by providing automated services which interact with the user as need be. An example of such an interface would be the Microsoft Agent (Clippy the paper clip) in Microsoft Office. Surprise Explain Reward systems are Mixed Initiative User Interfaces because the system runs automated services which engage the user as need be.
Intrinsically motivating software is software which has characteristics which motivate a user. It has been found [David 2001] that intrinsically motivating software leads to better performance. If Surprise Explain Reward contributes to making the software more intrinsically motivating, then this is a positive side effect. Malone and Lepper (reference included in [David 2001]) have identified four software features that evoke intrinsic motivation, control, fantasy, challenge, and curiosity. Control is the feeling that the user is in control of the software system. Fantasy takes the forms of analogy's and metaphors and provides the user with a way to relate their existing knowledge to the software. Control and fantasy can be found in direct manipulation and metaphor interfaces (such as 'desktops'). Surprise Explain Reward runs the risk of affecting control if surprises are aggressive enough to interrupt the user. Challenge for intrinsic motivation occurs when the software's skill requirement meets the user's skill level. Curiosity is either attention attracting aspects of the software such as light or sound, or a difference between what the user expects, and what they perceive. Curiosity is the element that Surprise Explain Reward can tap into. When surprises are presented, they create curiosity by tapping into some sense, such as drawing a red circle or playing a sound. After the user has learned about and chosen to use the Surprise Explain Reward features of the software, the presence of surprises can evoke curiosity by virtue of the fact that it is pointing to a possible problem in the software which the user may not have been aware of before. Even in the absence of a surprise, the uncertainty of when a surprise can occur, or why one is not present may contribute to the curiosity factor. Intrinsic motivation will be further discussed later in these bibliographic notes.

Contributors To Curiosity and Fear

Surprise Explain Reward relies on stimuli evoking enough curiosity in the user to inspire them to explore the source of the stimuli. Psychologists have extensively studied what causes curiosity and what motivates people to explore.

One important thing to keep in mind is that most psychological research on curiosity is done with subjects in simple situations with curiosity evoking stimuli. End-user programmers, on the other hand, are in very deep thought situations. This state of deep thought may have a significant impact on how they respond to stimuli.

Deci [Deci 1975] identifies three major theories to explain intrinsic motivation.

- Behavioralist psychologists have offered theories of optimal arousal which postulate that humans seek an certain optimal amount of arousal (a physiological
phenomena involving activity in the brain). Humans will be motivated to increase or decrease their arousal until they reach this point.

- Similarly, theories of optimal incongruence state that humans seek an optimal state of incongruence (a psychological parallel to the physiological concept of arousal, which can be described as a sense of mystery, or lack of understanding).
- A third way of explaining intrinsic motivation is with competence and self-determination theories, which postulate that humans seek to have mastery over things and thus seek challenges. Competence theories also have a parallel with optimum levels in that people will seek a challenge which seems appropriately matched to their needs.

When presenting surprises to a user, the goal is to surprise them enough to evoke curiosity, but not enough to evoke a negative reaction. It is difficult, and beyond the scope of this work, to quantify the effects of a stimuli on a user in terms of arousal or incongruence. Even if one could, it is also difficult to predict exactly how a user will respond to the new level of arousal/incongruence. The most I hope to do is to identify the variables and relationships such that they can be used to derive hypotheses.

There are at least two orthogonal factors to consider in determining how much surprise is enough. One factor is the optimal level of arousal/incongruence for the user. Another factor is the user's reaction to a new level of arousal/incongruence.

The optimal level of challenge for a person (in competence theories) is very dependent on the person's personality (Ts/Taf ratio). [Deci 1975]. The optimal state of arousal or incongruence for a person is subject to change over time [Deci 1975 p. 49].

A stimulus evokes incongruence in a user when it is different from what the user expects. Curiosity evoked is not directly proportional to incongruence. There is an inverse U shaped relationship between incongruence and curiosity [Day 1981]. Curiosity is proportional to incongruence to a certain point, past this point, evoked curiosity decreases as incongruence increases. One factor involved in how much curiosity is evoked by a stimulus is the perceived size of the information gap. If a user feels that they know a great deal about a topic, and a stimulus conveys information about the topic that does not line up with their expectations, it will evoke more curiosity than if the user does not feel that they know a great deal about the topic [Lowenstein 1994].

If the amount of incongruence is too low, it will not evoke enough curiosity to encourage a user to explore the source of the surprise. If the amount of incongruence is too great, then they will not react with curiosity, but rather will react with fear and aversion [Lowenstein 1994, Berlyne 1960]. This is an important point to note when
designing Surprise Explain Reward systems, because if the surprise pushes them above the incongruence threshold, they may be motivated to undo whatever action led to the surprise and/or never do that action again.

Behavioral psychologists have found also found an inverse U shaped curve for play (willingness to explore) versus arousal. [Day 1981] When users have low arousal levels, they engage in *diversive exploration*, which is a state of searching for stimuli to increase arousal levels. Once they have achieved a certain level of arousal, they look for ways to reduce their arousal level. One behavior the user may engage in is avoiding the stimuli; this they will do this if arousal is too high. Another way in which users can seek to reduce arousal is through *specific exploration* in which they focus their attention on something in order to learn and reduce arousal. Specific exploration occurs in a certain arousal window when arousal is too great for diversive exploration, but not enough to cause avoidance. When specific exploration occurs, curiosity sharply increases for the object they became curious about [Lowenstein 1994]. Surprises in Surprise Explain Reward are used to capture users’ attention, focus it on a topic, and bring them to specific exploration. So for Surprise Explain Reward to succeed, it must bring the user to the arousal window in which specific exploration lies.

There are conflicting views on how the length of time affects the reaction to a stimulus. Arousal has been shown to be inversely proportional to the length of time in which a stimulus is presented to a person [Day 1981]. However, some theories of curiosity postulate that it is a drive, and therefore should increase with time.

Evoked arousal is directly proportional to stimuli intensity [Day 1981 p.289].

During development, children engage in intellectual play, which is when a child achieves a skill and then uses it to feel competent; eventually they tire of it and will look for a new skill. If this carries on into adulthood, we could expect that users will tire of using the same tools over and over again, and seek additional tools.

**Hypothesis Based on Bibliography Notes**

What useful behavior can we elicit from users? How does what our surprises are like affect what we elicit? Knowing the answers to these questions will provide an empirical basis for making some guidelines in how to design the surprises in systems that use Surprise Explain Reward.

**What Useful Behavior Can We Elicit From Users?**
We can consider a user of the system during an empirical study. We confine the discussion to users in empirical studies as opposed to real world work environments because that is the situation that we will be testing under.

The user is asked to find and eliminate bugs from the code, but is given no training in the debugging features of the software that Surprise Explain Reward is designed to teach. They are not aware of how many or what kinds of bugs are in the software, they are only given a requirement specification for what the software should compute. The user has a fixed amount of time to perform the task and must use all of the time. Notice that this differs from a probable real life work situation because in a working environment, they can stop when they feel they are done or are not making enough progress to be worth their time.

When an end user programmer is making progress in debugging software on a language with Surprise Explain Reward features, they will have enough arousal that they will be in a specific exploration mode. If they continue to make progress, then the continuous influx of new stimuli (a changing, more thoroughly tested, or less buggy program) should keep their arousal levels high enough for them to remain in specific exploration.

Several things can happen which could cause them to shift their focus, either by leaving specific exploration and switching to diversive, or switching the focus of their specific exploration.

If they get stuck (a situation in which they can’t make further progress) they will try different techniques to get themselves back on track. For awhile, they will probably still be in specific exploration, in which case they will try things that seem most likely to get them back on track. But over time, if they cannot progress they will eventually drop to a lower state of arousal and will engage in diversive exploration. In diversive exploration they may be more willing to try things that have less to do with their original goal. Whether they are trying things in specific exploration, or trying things in diversive exploration, the effect is the same. A stuck user will begin to explore the software, and this is a good opportunity for a surprise to cause them to learn a new feature or investigate a possible software error.

The user may become bored if making progress is not enough to keep them at a high arousal state, or if they feel that they have completely removed all bugs from the software but are forced to use the remaining time. If they become bored they may drop below the arousal level necessary for specific exploration and move to diversive exploration. In this case, they may spread out their attention and investigate surprising aspects of the software.
Another way in which a user may switch the focus of what they are doing is if a surprise is strong enough to compel them to switch. In this case they don't leave specific exploration. There are many dangers in designing a system that forces this change of focus on a user. For one thing, the user has stopped doing their productive activity and instead turned to investigating the source of the surprise. Unless there is some reason why knowing the information immediately is of more use than them making progress on the software task, this may not be a good trade off. This change of focus scenario is similar to the studies of immediate style interruptions mentioned in the papers referenced earlier in this thesis in that users stop what they are doing in order to attend to the new task upon receiving an external stimulus. However, because it is still a negotiated interruption, it may not suffer as much from reorientation and harmful emotional reactions.

From the above discussion, two hypothesis about the kinds of behavior which can be elicited from users are proposed:

- Users will investigate some surprises when they are bored or stuck.
- Users will investigate some surprises even if they are already engaged (and not bored) with a primary task.

**How Does What Our Surprises Are Like Affect What We Elicit?**

We divide surprise communications into 4 categories by rating their intensity:

- minimal, medium, high, and shocking.

- A surprise communication of minimal intensity is not likely to evoke any more arousal than static features of the software, such as buttons or tool bars.
- A surprise communication of medium intensity will evoke more arousal than the static features of the software, thus when a person is trying different things to make progress, or increase their arousal, they are likely to explore these surprises.
- A surprise communication of high intensity is arousing enough that an engaged user is likely to stop what they are doing and switch focus to the source of the surprises.
- A surprise communication of shocking intensity is one that is strong enough to evoke a negative reaction from the user, in which case they will feel discomfort from its presence and will seek to find a way to get the surprise to go away. (Such as undoing their last action, not doing that action in the future, or
searching for any method to make that surprise go away, regardless of whether they understand why it went away.)

The category that a surprise falls into is subject to change from user to user and moment to moment (primarily depending on what they know about the surprise and what they are doing).

For negotiated interruption notifications, we propose that the longer they are on the screen, the lower the perceived intensity will be.

**Thoughts**

I think that receiving a surprise communication in response to an action may have a tendency to dissuade novice users from making the action, because arousal is always so high. A way to test this is to make red circles and shaded cells in response to X’s in WYSIWYT check boxes. It may be seen that users react to red circles with avoidance (Taking the X away and not placing more X’s in the future. This was casually observed during a think aloud study). If so, this would indicate that surprising users immediately in response to an action they have made is not a good idea. Rather surprise communications should come later (though it may not be good to completely hide the connection).

I believe that if a study were conducted, testing users’ reactions to the high-intensity surprise communications of all four intensity levels (though determining which intensity level a surprise communication is in is not a trivial task), that the following conclusions may be derived from the results. For the majority of notifications, which are not of crucial importance, a medium intensity surprise communication is appropriate. For a few notifications that are of crucial importance, a high intensity surprise communication is appropriate, but there are associated penalties and care must be taken not to accidentally present a surprise communication of shocking intensity. Minimal-intensity surprise communications are no more effective at attracting attention than static software elements such as toolbar buttons. Surprise communications of shocking-intensity should never be used, as they do not encourage learning, but rather encourage only avoidance.

A non-interrupting notification that remains on screen will decrease in its perceived intensity over time. It may, for example, drop to a lower category of intensity.

**Threats To Validity**

The hypothesis could be tested for users in experimental situations. The work environment may yield different results, particularly with respect to users that are 'bored'
or 'stuck', because such users won't be expected to work for a specific amount of time in a real world setting.

References For Bibliographic Notes


APPENDIX B: NOTES ON THE EFFECTS OF PROFESSIONAL SPREADSHEET EXPERIENCE ON HIGH-INTENSITY PARTICIPANTS

Like Dr. Cook's findings (from the immediate-style study), the high-intensity study also shows that the significant differences come from participants with professional spreadsheet experience. When comparing high-intensity to low-intensity, professional experience accounts for the differences in comprehension, bugs fixed on the second task, and editing of assertions. It did not, however, account for the higher usage of HMT on the second task by high-intensity participants (p=0.1206).

One oddity I noticed is that the high-intensity participants with professional experience were significantly able to assess the number of bugs they found on the grades spreadsheet (and not the weekly pay). Disregarding professional experience, the group, on the other hand, was not able to significantly assess debugging performance (p=0.1194). The people with professional experience were equally split between having grades as their first and second task, so that doesn’t account for anything. There just appears to be something about that spreadsheet that affects people with professional spreadsheet experience (and high-intensity interruptions).

Some of the raw Mann-Whitney p-values comparing participants with professional spreadsheet experience are given below.

<table>
<thead>
<tr>
<th></th>
<th>Professional Low-intensity vs High-intensity</th>
<th>Non-professional Low-intensity vs High-intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comprehension</td>
<td>0.0401</td>
<td>0.1142</td>
</tr>
<tr>
<td>Assertion Correctness</td>
<td>0.53</td>
<td>0.62</td>
</tr>
<tr>
<td>Bugs fixed first task</td>
<td>0.23</td>
<td>0.68</td>
</tr>
<tr>
<td>Assertion editing second task</td>
<td>0.0401</td>
<td>0.0712</td>
</tr>
<tr>
<td>HMT usage second task</td>
<td>0.1206</td>
<td>0.1416</td>
</tr>
</tbody>
</table>
Self assessment:
High-intensity participants with professional experience significantly accurate at judging their debugging performance on Grades (Mann-Whitney $0.0497$). Non-professionals are not significantly able to judge their accuracy (Mann-Whitney $p=0.4499$)
Hi, My name is TJ. I will be leading you through today's study.

Some of the other people involved are: Dr. Burnett, Dr. Cook grad and undergrad students that are helping me out today.

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

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

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

But first, I am required by Oregon State University to read the following statement:

Participation in this study is completely voluntary.

There are no risks associated with this study. We're testing techniques and not you. The benefits of completing the study are $20 for your time and the satisfaction of knowing that you have made a positive contribution to the field of computer science. There will be no record tying you to your work performed during the study. Your data will be assigned a random number, and no record will tie you to that number after the data has been collected.

If you choose not to participate, you may withdraw at any time.

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

Questions?

Contact:
- Dr. Margaret Burnett: burnett@cs.orst.edu
- Dr. Curtis Cook: cook@cs.orst.edu
- Dr. Gregg Rothermel: grother@cs.orst.edu
- Ledah Casburn: lcasburn@cs.orst.edu
Background questionnaire (Hand it out)
(please do NOT turn to any other pages until you are asked to do so)

Tutorial

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

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

As we go through this tutorial, I want you to actually DO the steps I'm describing. When I say "click", I'll always mean click the left mouse button once unless I specify otherwise. Pay attention to your computer screen while you do the steps.

If you have any questions, please don't hesitate to ask me to explain.

For each spreadsheet that we will working, you will have a sheet of paper describing what the spreadsheet is supposed to do.

Read the first page of the description of the DonutsStorageDecision spreadsheet now.
(PAUSE)

Now open the DonutStorageDecision spreadsheet by clicking on the icon labeled DonutStorageDecision at the bottom of the screen.
There are a few ways that Forms/3 spreadsheets look different than spreadsheets you are familiar with.

Forms/3 spreadsheets don't have cells in a grid layout. We can put cells anywhere.

We can give the cells useful names like Avg_Donuts_Sold. (Point to the cell on the spreadsheet)

You can also see that some cells have colored borders. I'll explain their meaning shortly.

However, just like with any other spreadsheet you can see a value associated with each cell.

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

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

So, the red border around the cells is just telling us that the computer does not know if the cell's value is correct. It's up to us to make a decision about the correctness of the cells based on how we know the spreadsheet should work. In our case we have the spreadsheet description that tells us how it should work.

Observe that Donuts_Sold_Day1 & Donuts_Sold_Day2 cells don't have any special border color. Such cells without colored borders are called input cells. Cells with colored borders are called output cells.

Now, Let's test the Avg_Donuts_Sold cell by making a decision whether or not the value is correct for the inputs. Read the description given on 2nd page of the DonutStorageDecision handout for the Avg_Donuts_Sold cell. (Pause).

The description says that Avg_Donuts_Sold cell gives us the average number of donuts
sold per day. Look at the values in Donuts_Sold_Day1 and Donuts_Sold_Day2, and compute the average of them. Is 268 the daily average? (pause) No. We should check to see if the formula for Avg_Donuts_Sold is correct.

Now, move your mouse and hold it over the box with a picture of an arrow on it (point to it) until a message appears. What does the message tell us? (pause) It says that by clicking on that tab, we are able to view the cell's formula. Go ahead and do that. You see the formula uses the value in Total_Sum and divides it by 3. This is wrong. It should divide by 2. Go ahead and change the 3 to a 2 since we only have 2 days worth of data. When you are done, hit "Apply". Clicking on "Hide" will close the formula box. Go ahead and hide the formula box.

Now lets make a decision about the correctness of the Avg_Donuts_Sold cell. Move your mouse to the box with the questionmark in it and hold it there until a message pops up. What does it say? The message tells us that if the cell's value is correct to go ahead and click there. Go ahead and click the box.

Notice what happened. Three things changed. A checkmark replaced the questionmark in the decision box. The borders for the cells Total_Sum and Avg_Donuts_Sold turned blue, and the % testedness indicator changed to 33% (point to it). Forms/3 lets us know what percent of the spreadsheet is tested through the % testedness indicator. It is telling us that we have tested 33% of this spreadsheet.

We know now what the red border color means, now find out what the blue border indicates by holding the mouse over the cell's border in the same way as before. What does the message say? It tells us that the cell is fully tested.

If you accidentally checked off the decision box, the value in the cell was really wrong, or you haven't seen the changes that occurred, you can "uncheck" the decision about Avg_Donuts_Sold with another click in the same decision box. Try it. (Pause) Everything went back to how it was. The cells' borders turned back to red, the %testedness indicator dropped back to 0% and a questionmark reappeared in the decision box.

Since we've already decided the value in the Avg_Donuts_Sold cell is correct, we want to retell Forms/3 that this value is correct for the inputs. So click in the decision box.
You may have noticed the blank decision box in the Total_Sum cell. Position your mouse on top of the box to find out why it is blank. A message pops up that says we have already made a decision about this cell, but I don't remember us making any decisions about Total_Sum and look, the cell's border is blue too.

How did that happen? Let's find out. Position your mouse to the Avg_Donuts_Sold cell and click the middle mouse button (it's the one that looks like a wheel). Notice that two colored arrows appear, one blue and one red.

Click the middle mouse button again on one of the arrows -- it disappears. Now, click the middle mouse button again on Avg_Donuts_Sold cell -- all the other arrows disappear. Now bring the arrows back again by re-clicking the middle mouse button on Avg_Donuts_Sold.

Move your mouse over to the blue arrow and hold it there until a message appears. It explains the arrow is showing a relationship exists between Total_Sum and Avg_Donuts_Sold. This means that the answer for Total_Sum goes into or contributes to the answer for Avg_Donuts_Sold.

Oh, ok, so this explains why the arrow is pointed in the direction of Avg_Donuts_Sold? Yes it does, and it also explains why both of the cell borders turned blue. Again, if you mark one cell as being correct and there were other cells contributing to it, then those cells will also be marked correct.

Look at the value for the cell Storage_Decision, ok now refer back to the description for the cell on the 2nd page of the spreadsheet description. (Pause). According to spreadsheet description this cell displays -1 to indicate a decrease in storage capacity if the Avg_Donuts_Sold per day is less than 40% of the storage capacity (ie 200 donuts). But, if the Avg_Donuts_Sold is greater than 80% of the storage capacity (ie 400 donuts) then this cell displays "1" to indicate an increase in storage capacity is recommended. If Avg_Donuts_Sold is between 40 and 80% (200 to 400 donuts) then the Storage_Decision cell displays a "0" to indicate that no changes should be made to the storage capacity.

The current set of values covers a situation when Avg_Donuts_Sold is greater than 400 donuts. The Storage_Decision Cell displays "1." Is "1" the correct answer for this
situation? (Look at the cell description and then make a decision—Pause). "1" is in fact the correct answer, go ahead and click the decision box for Storage_Decision.

Notice the Storage_Decision cell did not turn blue it's now a shade of purple and the % testedness indicator is only at 66%. Position your mouse over the purple border to find out what the color means. A message appears telling us that this cell is only partially tested. This means there are more situations for this cell that we haven't tried to test. Let's try to find out what those situations could be. Open the formula for Storage_Decision.

woah, the formula looks pretty complicated. How do we figure out which inputs are going to give us a new situation we haven't tested? Say the situation when Avg_Donuts_Sold is between 200 and 400 donuts. We could figure this out manually by going back to the spreadsheet description and changing the input cells, but Forms/3 can also help. Do you see the help button at the top of the screen? Go ahead and click it. Now if you're looking you may or may not see values in the cells changing, but when Help stops you'll notice a couple of other things.

Wait a second, what are those little boxes for that are popping up on my cells, they weren't there before? Ignore them for now, we'll come back to explore them in a second. But first, notice that the borders thicken up in some of the cells, and now we have a questionmark in the Storage_Decision cell where we didn't have one before.

The thicker borders just let us know which cells the Help feature modified to give us a new situation.

I want us to make a decision about the value in Storage_Decision. (since this is the whole reason we pushed the help button in the first place). We have a questionmark in the decision box and a new value in the cell. (If you're having trouble remembering what any of the features are, Like the questionmark, just hover your mouse over the object until a message pops up reminding you what it is) Now, take a look at the Avg_Donuts_Sold cell and refer back to the cell description for Storage_Decision. If Avg_Donuts_Sold is 399.5, what value should we expect for the Storage_Decision? (Pause) We should expect to see a "0" in this cell. Is that what we see on our spreadsheet? yes. Good. Let's click on the decision box to tell Forms/3 this is a correct value.
You should notice that the Storage_Decision cell turns slightly more bluish but is still a shade of purple. This is telling us that our cell is not yet fully tested. You may have also noticed that one of the arrows pointing to the Storage_Decision cell is now fully blue and our %testedness indicator is up to 88%.

Can you tell by looking at the arrows which situation for Storage_Decision it is that we haven't covered in our testing? Yes, it's the purple one -- the situation where the Avg_Donuts_Sold is less than 40% or 200 donuts.

To test this situation we can use the same Help feature. Before, we just pressed Help without asking it to help us test a specific situation. This time, we are going to select a specific arrow to get Help with testing. (You can also select a specific cell to test.)

Go ahead and select the purple arrow by clicking on it. Instead of clicking on the Help button again; look in the lower right hand corner of the spreadsheet. You should see a button that says "Start Testing." Go ahead and push that button.

You'll notice that lots of cells just changed. I want you to explore these cells by resting your mouse over the various pieces of them. You'll have 30 seconds to do just that before we continue.

Can you see anything that looks wrong or out of place? Remember that the ultimate goal is to make sure the spreadsheet works like it says in the description. Refer back to it as many times as you need it. Take two minutes now to explore the spreadsheet. See if you can find any more bugs and if you do, fix them. If you need help or can't remember how something works, use the mouse over feature to get more information about it.

(pause 2 minutes)

Exploratory Task #2:

Ok, time is up. To continue developing the skills you'll need in a few minutes, I'm going to ask you first minimize the spreadsheet you were working with. Then, open the Donut_Muffins_Sales spreadsheet and read the spreadsheet description for it. (read it with them--so we now how long to pause)
Now turn to the third page of the Donut_Muffin_Sales description and read the tutorial task. (pause) Some spreadsheet developer has already created the spreadsheet that has 3 days of data for donut and muffin sales.

Now, you'll have about 5 minutes to explore the spreadsheet. If you encounter any errors, fix them. Remember that the ultimate goal is to make sure the spreadsheet works like it says in the description. Refer back to it as many times as you need it. If you need help or can't remember how something works, use the mouse over feature to get more information about it.

(pause 5 minutes)

(End of tutorial)

You'll have X minutes to complete the next task, if you finish before the time is up. Please remain seated and at your computer. Do not check your email or open up anything unless you are instructed to do so.

Allow 35 minutes for subjects to complete Grades task.
Allow 22 minutes for subjects to complete Weekly_Pay task.
Remember to switch around task order for actual study:

When time is called for the first task, have them turn to the first post session questionnaire and take a minute to fill it out.
APPENDIX D: SPREADSHEET

DESCRIPTIONS AND QUESTIONNAIRES

Donut_Muffin_Sales

This spreadsheet gives the details of donut and muffin sales at the "Simply Fresh" Bakery.
Currently the Bakery has an on-shelf storage capacity for 400 donuts and 100 muffins.
Daily, the bakery makes 400 donuts and 100 muffins. Any donuts or muffins that are not sold over the day are thrown away.

Management wants to make a decision on increasing the on-shelf storage space. They decide to observe the average sales over 3 days. If the average number of donuts and muffins sold over three days is more than 80% of the current on-shelf storage capacity they will accept the proposal for increasing the storage capacity.
<table>
<thead>
<tr>
<th><strong>Input Cells</strong></th>
<th><strong>Functionality of the cells</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Donuts_Sold_Day1</td>
<td>Donuts sold on day 1. This can't be less than zero and more than</td>
</tr>
<tr>
<td></td>
<td>the current storage capacity (i.e. 400).</td>
</tr>
<tr>
<td>Donuts_Sold_Day2</td>
<td>Donuts sold on day 2. This can't be less than zero and more than</td>
</tr>
<tr>
<td></td>
<td>400.</td>
</tr>
<tr>
<td>Donuts_Sold_Day3</td>
<td>Donuts sold on day 3. This can't be less than zero and more than</td>
</tr>
<tr>
<td></td>
<td>400.</td>
</tr>
<tr>
<td>Muffins_Sold_Day1</td>
<td>Muffins sold on day 1. This can't be less than zero and more than</td>
</tr>
<tr>
<td></td>
<td>the current storage capacity (i.e. 100).</td>
</tr>
<tr>
<td>Muffins_Sold_Day2</td>
<td>Muffins sold on day 2. This can't be less than zero and more than</td>
</tr>
<tr>
<td></td>
<td>100.</td>
</tr>
</tbody>
</table>
Muffins Sold Day3
Muffins sold on day 3. This can't be less than zero and more than 100.

Output Cells

Total_Sum_Donuts
Total donuts sold over the given number of days.

Total_Sum_Muffins
Total muffins sold over the given number of days.

Avg_Donuts_Muffins
Average number of donuts plus muffins sold per day.

Extra_Storage_Decision
If Avg_Donuts_Muffins_Sold is more than 80% of the current on-shelf storage capacity (i.e. 400) then this cell displays "1" This means increase in storage capacity is recommended. Otherwise, it displays "0" meaning there is no need to increase storage capacity.

Tutorial Task
You are given a spreadsheet created by a spreadsheet developer.

Your tasks are:

Test the spreadsheet thoroughly to ensure that it does not contain errors and works according to the spreadsheet description.

If you encounter any errors in the spreadsheet fix them.

As you work through the above tasks, explore with the mouse to find features that may help you test and fix the spreadsheet.
Donut Storage Decision

This spreadsheet gives the details of donut sales at the "Simply Fresh" Bakery store. Currently the Bakery store has an on-shelf storage capacity for 500 donuts. The bakery store daily makes 500 donuts and any donuts that are not sold over the day are thrown away.

The management wants to make a decision about whether to increase or decrease the on-shelf storage space for donuts. It decides to observe the average sales over 2 days.

If the number of donuts sold over the two days are more than 80% of current on-shelf storage capacity then they will accept the proposal for increasing the storage capacity. But if the number of donuts sold over the two days are less than 40% of current on-shelf storage capacity then they will decrease the storage capacity. Otherwise, the management will assume everything is running just fine and keep the on-shelf storage capacity the same.
**Input Cells**  
**Functionality of the cells**

- **Donuts_Sold_Day1**  
  Number of donuts sold on day 1. This can't be less than zero and more than the current storage capacity (500 donuts)

- **Donuts_Sold_Day2**  
  Number of donuts sold on day 2. This can't be less than zero and more than the current storage capacity (500 donuts)

**Output Cells**

- **Total_Sum**  
  Total number of donuts sold over the given number of days

- **Avg_Donuts_Sold**  
  Average number of donuts sold per day

- **Storage_Decision**  
  If `Avg_Donut_Sold` is less than 40% of the current on-shelf storage capacity (ie. 200 donuts) then this cell displays 
  "-1" to indicate a recommendation to decrease storage capacity.  
  
  If `Avg_Donut_Sold` is greater than or equal to 40% of the current on-shelf storage capacity (ie. 200 donuts) but less than or equal to 80% (ie. 400 donuts) then this cell displays "0" to indicate there is no need to increase storage capacity.  
  
  Otherwise, if `Avg_Donut_Sold` is greater than 80% then this cell displays "1" to indicate a recommendation to increase storage capacity.
Tutorial Task

You are given a spreadsheet created by a spreadsheet developer.

Your tasks are:
Test the spreadsheet thoroughly to ensure that it does not contain errors and works according to the spreadsheet description.
If you encounter any errors in the spreadsheet fix them.
As you work through the above tasks, explore with the mouse to find features that may help you test and fix the spreadsheet.
Grades

The basic spreadsheet for determining the grade for a 411/511(Undergrad/Grad) class is based on three homeworks (one optional for Grads), two midterms and a final. The grad students can earn extra credit points by attending an advanced lecture series.

The first two homeworks carry 50 points each while the third homework carries 100 points. Both the midterms and the final also carry 100 points each.

The letter grade for both grads and undergrads is calculated from total score in the spreadsheet as follows:

<table>
<thead>
<tr>
<th>Total Score</th>
<th>Grade Awarded</th>
</tr>
</thead>
<tbody>
<tr>
<td>90 - 100</td>
<td>A</td>
</tr>
<tr>
<td>80 - 89</td>
<td>B</td>
</tr>
<tr>
<td>70 - 79</td>
<td>C</td>
</tr>
<tr>
<td>60 - 69</td>
<td>D</td>
</tr>
<tr>
<td>0 - 59</td>
<td>F</td>
</tr>
</tbody>
</table>

Note: In the spreadsheet, the cell names starting with 'U_' refer to the data for an undergraduate student and the cell names starting with 'G_' refer to the data for a graduate student.

If Grading criteria is different for Graduate and Undergraduates then the cell description is given separately for those cases.

Input Cells

HW1 A student’s score for homework1. It cannot be less than 0 or more than 50.

HW2 A student’s score for homework2. It cannot be less than 0 or more than 50.

Sum_1st2nd Sum of homework1 and homework2 score. It cannot be less than 0 or more than 100.
HW3  A student's score for homework3. It cannot be less than 0 or more than 100.

Midterm1  A student's score on the midterm1. It cannot be less than 0 or more than 100.

Midterm2  A student's score on the midterm2. It cannot be less than 0 or more than 100.

Final  A student's score on the final examination. It cannot be less than 0 or more than 100.

G_LecAttended  The number of lectures attended by a grad student. It cannot be less than 0 or more than 10.

Output Cells

U_EffHW  The average of both the Sum of first two homeworks and the third homework. It cannot be less than 0 or more than 100.

G_EffHW  If a student scores more than 90 points on the Sum of HW1 and HW2 then the G_EffHW score is this Sum. The third homework is an optional practice homework and is not included in the calculation of G_EffHW score.

If a student's score for the sum of HW1 and HW2 is 90 points or below then the G_EffHW score is taken as the average of all three homework scores (remember that the first two homeworks are only worth 50 points each).

In any case G_EffHW score cannot be less than 0 and more than 100 points.
EffMidterm An average of midterm1 and midterm2. EffMidterm score cannot be less than 0 or more than 100.

G_ECAward Students can earn extra-credit points by attending a series of 10 lectures.

1.5 extra credit points are awarded if the student attends 8 or more lectures.
2.0 points otherwise.

U_TotalScore Total score for the undergrad students is the sum of 30% of the effective homework score, 30% of effective midterm score, 40% of the final. It cannot be less than 0 or more than 100.

G_TotalScore Total score for the grad students is the sum of 30% of the effective homework score, 40% of effective midterm score, 30% of the final and ECAward.
It cannot be less than 0 or more than 105.

LetterGrade Displays the letter grade awarded to the student based on total score.

**Experiment Task:**

You are given a spreadsheet that determines the letter grade for two students, the first one is Undergraduate (cells on the left half of the spreadsheet) and the second one is a Graduate student (cells on the right half of the spreadsheet).

Professor MakeItPerfect' wants to be sure that the spreadsheet is working correctly and contain no errors.
Your task is to 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.
The Weekly Pay spreadsheet gives the weekly pay and income tax withholding for Sales Persons. It is based on the hours worked each day of the week (including weekends), pay rates and the sales for the week.

The Weekly Pay and the tax withheld is calculated in the spreadsheet as follows:

- The hourly payrate for a sales person for working on weekdays is $40.
- The salesperson can work at the most 8 hours a day (including weekends).
- The Week Days Pay for a salesperson is the product of the total number of hours worked on weekdays times payrate for weekdays.
- The company wants to encourage salespersons to work on weekends and it pays at higher rate “1.25 times the weekday payrate” for work on weekends.
- The Week End Pay for a sales person is the product of the total number of hours worked on weekends times payrate for weekend.
- The Weekly Base Pay is the Sum of Weekdays pay and Weekend pay.
- The 'maximum possible weekly sales' for salespersons is $40,000 because of limits on the production capacity.
- A salesperson earns a 1st Bonus (a % of weekly base pay) if his/her weekly sales exceed a first level minimum value for sales. A 2nd bonus (a % of weekly sales) can be earned if weekly sales exceeds a second level minimum value.

The table below gives the first and second level minimum values for sales in order to receive a bonus. The table also describes 'payrates' for weekdays and weekends for the salespersons.

<table>
<thead>
<tr>
<th>WeekDay PayRate</th>
<th>WeekEnd PayRate</th>
<th>Sales to get 1st bonus</th>
<th>1st Bonus</th>
<th>Sales to get 2nd bonus</th>
<th>2nd Bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>$40.00</td>
<td>$50.00</td>
<td>Greater than $15,000.00</td>
<td>10% weekly base pay</td>
<td>Greater than $25,000.00</td>
<td>2% of Sales</td>
</tr>
</tbody>
</table>

For income tax withholding, the rate is 15% for Weekly Total Pay less than $1500 and 20% for weekly total pay $1500 or higher.
### Input Cells

<table>
<thead>
<tr>
<th>Day</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monday</td>
<td>Gives the hours worked by a sales person on each day of the week. Number of hours listed cannot be less than 0 and more than 8.</td>
</tr>
<tr>
<td>Tuesday</td>
<td></td>
</tr>
<tr>
<td>Wednesday</td>
<td></td>
</tr>
<tr>
<td>Thursday</td>
<td></td>
</tr>
<tr>
<td>Friday</td>
<td></td>
</tr>
<tr>
<td>Saturday</td>
<td></td>
</tr>
<tr>
<td>Sunday</td>
<td></td>
</tr>
</tbody>
</table>

### Output Cells

<table>
<thead>
<tr>
<th>Cell</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDay_PayRate</td>
<td>Gives the hourly pay rate for the week days.</td>
</tr>
<tr>
<td>Weekly_Sales</td>
<td>Gives the sales value for a specific salesperson.</td>
</tr>
<tr>
<td>WEnd_PayRate</td>
<td>Gives the hourly pay rate for the week ends.</td>
</tr>
<tr>
<td>WDay_Hrs</td>
<td>Gives the total number of hours for a sales person on weekdays. It cannot be less than 0 and more than 40.</td>
</tr>
<tr>
<td>WEnd_Hrs</td>
<td>Gives the total number of hours for a sales person on weekends. It cannot be less than 0 and more than 16.</td>
</tr>
<tr>
<td>WDays_Pay</td>
<td>Gives the total base pay earned on weekdays by a salesperson. It cannot be less than $0 and more than $1600.</td>
</tr>
<tr>
<td>WEnd_Pay</td>
<td>Gives the pay earned on week ends by a salesperson. It cannot be less than $0 and more than $800.</td>
</tr>
<tr>
<td>Weekly_BasePay</td>
<td>Gives the total weekly base pay. It cannot be less than $0 and more than $2400.</td>
</tr>
<tr>
<td>1st_Bonus</td>
<td>Gives the bonus achieved by a sales person when the first level minimum sales value is exceeded.</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>2nd_Bonus</td>
<td>Gives the bonus achieved by a sales person when the second level minimum sales value is exceeded.</td>
</tr>
<tr>
<td>Weekly_TotalPay</td>
<td>Gives the total pay for the week.</td>
</tr>
<tr>
<td>Tax_Withheld</td>
<td>Gives the tax withheld for sales person depending on his Weekly Total Pay.</td>
</tr>
</tbody>
</table>

**Experiment Task:**

You are given a spreadsheet that determines weekly pay and tax withheld for a salesperson.

- Your task is to 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.
Background

Major 

Year 

Overall GPA 

Gender (circle): Male  Female 

Do you have any programming experience?

High school course(s). How many? 

College course(s). How many? 

Professional. How long? 

Have you ever created a spreadsheet for ? (Check all that apply)

A high school course 

A college course 

Professional use 

Personal use 

Have you participated in any previous Forms/3 experiments? Yes / No
Pre-session Questionnaire

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

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

1 = Not Confident
2 = Somewhat Confident
3 = Moderately Confident
4 = Quite Confident
5 = Totally Confident

1 ... if there was no one around to tell me what to do as I go.
2 ... if I had never used a spreadsheet like it before.
3 ... if I had only the software manuals for references.
4 ... if I had seen someone else using it before trying it myself.
5 ... if I could call someone for help if I got stuck.
6 ... if someone else had helped me get started.
7. ... if I had a lot of time to complete the task.

8. ... if I had just the built-in help facility for assistance.

9. ... if someone showed me how to do it first.

10. ... if I had used similar spreadsheets before this one to do this same task.
Post Session Questionnaire #1

1. Use this scale to answer the following question.
   
   1 = Not Confident  
   2 = Somewhat Confident  
   3 = Confident  
   4 = Quite Confident  
   5 = Very Confident

   How confident are you that you found all the bugs in the Grades 3 spread sheet?  
   1 2 3 4 5

   How confident are you that you fixed all the bugs in the Grades spreadsheet?  
   1 2 3 4 5

2. How much additional time would you need to complete this task?
   
   ____ None. It only took me ____ minutes.
   ____ None. I took about the whole time.
   ____ I would need about ____ more minutes.
   ____ I'm not sure.

---

3 For participants who had weekly pay first, the questions would ask about weekly pay.
Post Session Questionnaire #2

1. Use this scale to answer the following questions.

   1 = Not Confident
   2 = Somewhat Confident
   3 = Confident
   4 = Quite Confident
   5 = Very Confident

   How confident are you that you found all the bugs in the Weekly_Pay spreadsheet? 1 2 3 4 5
   How confident are you that you fixed all the bugs in the Weekly_Pay spreadsheet? 1 2 3 4 5

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

   _____ None. It only took me _____ minutes.
   _____ None. I took about the whole time.
   _____ I would need about _____ more minutes.
   _____ I'm not sure.

3. Use this scale to answer questions regarding use of Guards in finding and fixing errors:

   1 = Not Helpful
   2 = Somewhat Helpful
   3 = Helpful
   4 = Quite Helpful
   5 = Very Helpful
   6 = No Opinion

   Help Guards (Help figure) 1 2 3 4 5 6
   User Guards (stick figure) 1 2 3 4 5 6
   Computer Guards (computer figure) 1 2 3 4 5 6

---

4 If grades was the participants second spreadsheet, then the questions would pertain to grades.
| Value Circled (Value in cell out of range) | 1 | 2 | 3 | 4 | 5 | 6 |
| Guards Circled (Conflict between user & computer guard) | 1 | 2 | 3 | 4 | 5 | 6 |
| Arrows were | 1 | 2 | 3 | 4 | 5 | 6 |
| Cell border colors were | 1 | 2 | 3 | 4 | 5 | 6 |
| Percent tested indicator was | 1 | 2 | 3 | 4 | 5 | 6 |

Questions 4-5 Use this scale to answer the following question.

1 = Not at all
2 = Small Distraction
3 = Large Distraction

4. Did the flashing red ovals distract you while you were working? 1 2 3

5. Did the flashing cell backgrounds distract you? 1 2 3

Comments ________________________________
Questions 6-12 refer to the spreadsheet above. For these questions choose the best answer from the list of choices following the questions. Answers may be used more than once. Question 12 is fill-in-the-blank.

6. What does the red oval and flashing background on cellA mean? ___

For the low and immediate-style participants, this picture was used:
7. What does the little stick figure in the cellA guard mean? 

8. Why is there a stick figure and a computer on cellB's guard? 

9. What does the red oval on the cellB guard mean? 

10. What does the icon on cellC mean? 

11. Which choice below is true of the guard on cellC? 

12. Given the formula in cellB and the range guard on cellA, what do you think Forms/3 says are the valid range(s) for cellB? 

List of Choices:
A. The user and Forms/3 disagree on the valid range(s) for this cell.
B. The value falls outside the valid range.
C. The guard was supplied by the user.
D. The guard was supplied by both Forms/3 and by the user.
E. Forms/3 disagrees with the range [0 to 5].
F. The user and Forms/3 agree on the valid range(s) for this cell.
G. The guard was supplied by Forms/3.
H. The user must change this guard.
I. I'm not sure.

Please provide any other general comments/suggestions you may have regarding Guards:
_________________________________________________________________________
_________________________________________________________________________
_________________________________________________________________________
_________________________________________________________________________

If you edited/changed or added new user guards while working on the tasks then tell briefly why did you do so?
_________________________________________________________________________
13. How anxious (e.g. nervousness or uneasiness) did you feel while doing the assigned task?

Not
Slightly
Moderately
Quite
Extremely

14. How frustrated (feeling that software is getting in your way) did you feel while doing the assigned task?

Not
Slightly
Moderately
Quite
Extremely

15. How difficult was the assigned task?

Not
Slightly
Moderately
Quite
Extremely

Did you use guards (see figure to the left)? If yes, answer problem 16 (then jump to problem 18), otherwise answer problem 16.

16. When I used guards...

... the computer made bad decisions with them.

... I worried they would distract me

<table>
<thead>
<tr>
<th>Don’t</th>
<th>Agree</th>
<th>Moderately</th>
<th>Mostly</th>
<th>Totally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>Somewhat</td>
<td>Agree</td>
<td>Agree</td>
<td>Agree</td>
</tr>
<tr>
<td>Agree</td>
<td>Somewhat</td>
<td>Agree</td>
<td>Agree</td>
<td>Agree</td>
</tr>
</tbody>
</table>
from my original goal.

... I was afraid that I would not use them properly.

<table>
<thead>
<tr>
<th></th>
<th>Don't</th>
<th>Agree</th>
<th>Moderately</th>
<th>Mostly</th>
<th>Totally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agreed</td>
<td>Agree</td>
<td>Somewhat</td>
<td>Agree</td>
<td>Agree</td>
<td>Agree</td>
</tr>
</tbody>
</table>

... it seemed like they were causing problems with the spreadsheet.

<table>
<thead>
<tr>
<th></th>
<th>Don't</th>
<th>Agree</th>
<th>Moderately</th>
<th>Mostly</th>
<th>Totally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agreed</td>
<td>Agree</td>
<td>Somewhat</td>
<td>Agree</td>
<td>Agree</td>
<td>Agree</td>
</tr>
</tbody>
</table>

... I worried that they would not help achieve my goal(s).

<table>
<thead>
<tr>
<th></th>
<th>Don't</th>
<th>Agree</th>
<th>Moderately</th>
<th>Mostly</th>
<th>Totally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agreed</td>
<td>Agree</td>
<td>Somewhat</td>
<td>Agree</td>
<td>Agree</td>
<td>Agree</td>
</tr>
</tbody>
</table>

... I was afraid I would take too long to learn them.
17. I did not use guards because...

... the computer would make bad decisions with them.  

<table>
<thead>
<tr>
<th></th>
<th>Don't</th>
<th>Agree</th>
<th>Moderately</th>
<th>Mostly</th>
<th>Totally</th>
</tr>
</thead>
<tbody>
<tr>
<td>guards</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... I worried they would distract me from my original goal.  

<table>
<thead>
<tr>
<th></th>
<th>Don't</th>
<th>Agree</th>
<th>Moderately</th>
<th>Mostly</th>
<th>Totally</th>
</tr>
</thead>
<tbody>
<tr>
<td>guards</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... I was afraid that I would not use them properly.  

<table>
<thead>
<tr>
<th></th>
<th>Don't</th>
<th>Agree</th>
<th>Moderately</th>
<th>Mostly</th>
<th>Totally</th>
</tr>
</thead>
<tbody>
<tr>
<td>guards</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... it seemed like they could cause problems with the spreadsheet.  

<table>
<thead>
<tr>
<th></th>
<th>Don't</th>
<th>Agree</th>
<th>Moderately</th>
<th>Mostly</th>
<th>Totally</th>
</tr>
</thead>
<tbody>
<tr>
<td>guards</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... I worried that they would not help achieve my goal(s).  

<table>
<thead>
<tr>
<th></th>
<th>Don't</th>
<th>Agree</th>
<th>Moderately</th>
<th>Mostly</th>
<th>Totally</th>
</tr>
</thead>
<tbody>
<tr>
<td>guards</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... I was afraid I would take too long to learn them.  

<table>
<thead>
<tr>
<th></th>
<th>Don't</th>
<th>Agree</th>
<th>Moderately</th>
<th>Mostly</th>
<th>Totally</th>
</tr>
</thead>
<tbody>
<tr>
<td>guards</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

18. For the following statements (A-E) use one or more of the options (a-e) listed below.

a. %-testedness indicator  
b. Pop up Message boxes  
c. Colored cell border  
d. Guards that appear when "help" is pressed  
e. User guards (stick figure)  
f. Computer guards (computer figure)  
g. Value circled red (value out  
h. Guards circled red (conflicts
of range of guard) between user and computer guards)
i. Arrows j. Other

Things the computer did that ...
A. made me trust it less: 

B. made me disbelieve that it knew the right answers: 

C. supported my goals in this spreadsheet: 

D. made me distrust my own understanding or competence: 

E. did not seem in line with my goals in this spreadsheet: 
APPENDIX E: EXPERIMENT
SPREADSHEET FORMULAS

The grades spreadsheet with all the formulas.

Weekly pay spreadsheet with all formulas.