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

Author Keywords: End-user programming, end-user software engineering, debugging, interruptions, Surprise-Explain-Reward.

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

CHI 2004, April 24–29, 2004, Vienna, Austria.
Copyright 2004 ACM 1-58113-702-8/04/0004…$5.00.

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

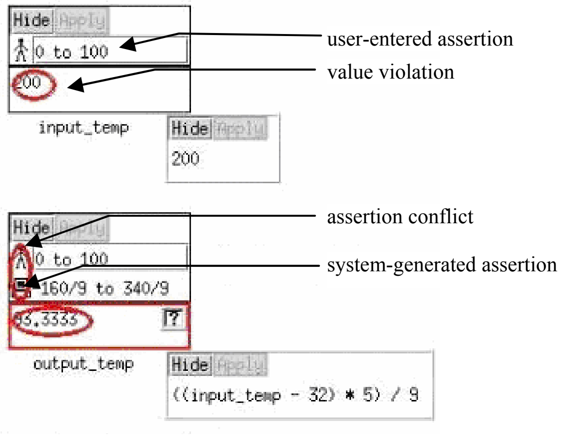


Figure 1: Assertion examples.

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*.

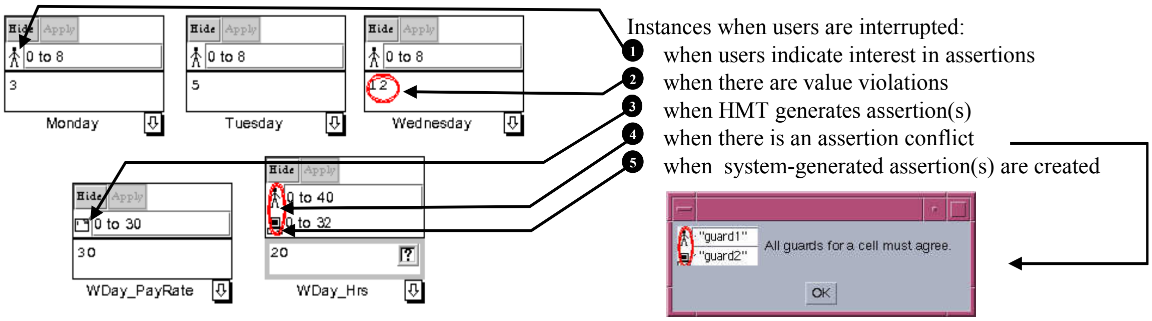
For example, in Figure 1, the user has entered an assertion for cell input_temp, which propagated through output_temp's formula to create a system-generated assertion. Since the values "200" and "33.3333" do not fulfill their cells' assertions, they are circled. Finally, the user also entered an assertion "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.


Hide Apply
0 to 8
3
Monday
Hide Apply
0 to 8
5
Tuesday
Hide Apply
0 to 8
12
Wednesday
Hide Apply
0 to 30
30
WDay_PayRate
Hide Apply
0 to 40
0 to 32
20
WDay_Hrs
Instances 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
"guard1"
"guard2"
OK

All guards for a cell must agree.

Figure 2: Instances of immediate-style interruptions in the experiment.

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 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.

Interruption style	1st Task	2nd Task
Negotiated (n=16)	13:26	3:40
Immediate (n=22)	8:34	4:49

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

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:

H1: 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 H1 cannot be rejected.

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

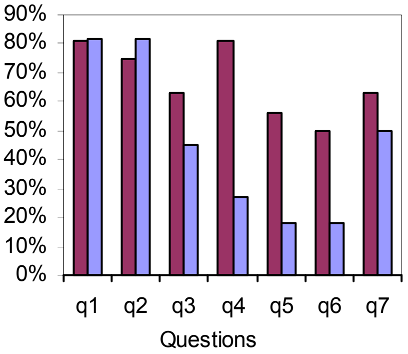


Figure 3: Participants who answered each comprehension question correctly: negotiated-style participants (dark bars) and immediate-style participants (light bars).

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

Table 2: Categorizations of the comprehension questions.

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.

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: There will be no difference in the accuracy of assertions entered by the negotiated- and 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: There will be no difference in the negotiated- and immediate-style participants' debugging productivity on the first task.

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.

Interruption style	Total bugs fixed	1st Task (Bugs per minute)	2nd Task (Bugs per minute)
Negotiated (n=16)	13 (3.24)	0.202 (0.079)	0.264 (0.061)
Immediate (n=22)	11.18 (3.56)	0.210 (0.066)	0.163 (0.069)

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

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

Interruption style	1st Task					2nd Task				
	Edit formula	Edit assertion	Use HMT	Check off value	Avg. total	Edit formula	Edit assertion	Use HMT	Check off value	Avg. total
Negotiated (n=16)	17.81	11.19	18.00	30.31	77.31	16.13	10.75	10.94	23.69	61.50
Immediate (n=22)	12.73	12.68	17.50	23.86	66.77	10.00	10.91	16.45	27.82	65.18
	p=0.0597					p=0.0231		p=0.0406		

Table 4: Average number of each type of activity engaged in by the participants. Significant differences (p<.05) are shaded.

regression coefficient is the slope of the least squares fitting of the ratings against the bugs that were corrected.

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.

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.

Interruption style	Regression coefficient	T-value	Signif.
Grades:			
Negotiated (n=16)	1.214	2.251	0.0410
Immediate (n=22)	0.240	0.590	0.9534
Weekly Pay:			
Negotiated (n=16)	0.683	2.650	0.0190
Immediate (n=22)	0.301	1.420	0.1711

Table 5: Regression analyses of actual bugs corrected vs. perceived bugs corrected.

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

1. Bailey, B.P., Konstan, J.A., and Carlis, J.V. Measuring the effects of interruptions on task performance in the user interface. *IEEE Proc. Conf. Systems, Man, and Cybernetics* (2000), 757-762.
2. Burmistrov, I. and Leonova, A. Do interrupted users work faster or slower? The micro-analysis of computerized text editing task. *Human-Computer Interaction: Theory and Practice (Part I) – Proc. HCI International 2003, Vol. 1*. (J. Jacko and C. Stephanidis, eds.) Lawrence Erlbaum Associates, Mahwah, NJ, 2003, 621-625.
3. Burnett, M., Atwood, J., Djang, R., Gottfried, H., Reichwein, J., and Yang, S. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *J. Functional Programming 11*, 2 (2001), 155-206.
4. Burnett, M., Cook, C., Pendse, O., Rothermel, G., Summet, J., Wallace, C. End-user software engineering with assertions in the spreadsheet paradigm. *Proc. 25th Int. Conf. Soft. Eng.* (2003), 93-103.
5. Carroll, J. *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. MIT Press, Cambridge, MA, 1990.
6. Corbett, A. and Anderson, J. Locus of feedback control in computer-based tutoring: Impact on learning rate, achievement and attitudes. *Proc. CHI 2001*, 245-252.
7. Czerwinski, M., Cutrell, E., and Horvitz, E. Instant messaging: Effects of relevance and time. *People and Computers XIV: Proc. HCI 2000, Vol. 2* (S. Turner and P. Turner, eds.), British Computer Society, 2000, 71-76.
8. Fisher, M., Cao, M., Rothermel, G., Cook, C., and Burnett, M. Automated test generation for spreadsheets. *Proc. 24th Int. Conf. Soft. Eng.* (2002), 141-151.
9. Hess, S., Detweiler, M. Training to reduce the disruptive effects of interruptions. *Proc. Human Factors and Ergonomics Society Annual Mtg.* (1994), 1173-1177.
10. Hudson, S., Fogarty, J., Atkeson, C., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J., and Yang, J. Predicting human interruptibility with sensors: A Wizard of Oz feasibility study. *Proc. CHI 2003*, 257-264.
11. Ko, A.J. and Myers, B.A. Development and evaluation of a model of programming errors. *Proc. IEEE Symp. Human-Centric Computing Languages and Environments* (2003), 7-14.
12. Ko, A.J. and Myers, B.A. Designing the whyline: A debugging interface for asking questions about program failures. *Proc. CHI 2004* (to appear).
13. Mathan, S. and Koedinger, K. Recasting the feedback debate: Benefits of tutoring error detecting and correction skill. *Int. Conf. Artificial Intell. Education* (2003).
14. McFarlane, D.C. Comparison of four primary methods for coordinating the interruption of people in human-computer interaction. *Human-Computer Interaction 17*, 1 (2002), 63-139.
15. Miller, R. and Myers B. Outlier finding: Focusing user attention on possible errors. *Proc. User Interface Soft. and Technology* (2001), ACM Press, 81-90.
16. Pongched, P. *A More Complex Model of Relevancy in Interruptions*. Human-Computer Interaction Capstone, School of Computer Science, DePaul University, Chicago, IL (2003). http://www.spong.org/~pechluck/HCI/content-of-interruptions.pdf
17. Ruthruff, J., Creswick, E., Burnett, M., Cook, C., Prabhakararao, S., Fisher II, M., and Main, M. End-user software visualizations for fault localization. *Proc. ACM Symp. Soft. Visualization* (2003), 123-132.
18. Speier, C., Valacich, J., and Vessey, I. The effects of task interruption and information presentation on individual decision making. *Proc. 18th Int. Conf. Information Systems* (1997), 21-36.
19. Wagner, E. and Lieberman, H. An end-user tool for e-commerce debugging. *Proc. Int. Conf. Intelligent User Interfaces* (2003), 331-331.
20. Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., Durham, M., and Rothermel, G. Harnessing curiosity to increase correctness in end-user programming. *Proc. CHI 2003*, 305-312.