Explaining Debugging Strategies to End-User Programmers

Neeraja Subrahmaniyan, Cory Kissinger, Kyle Rector,
Derek Inman, Jared Kaplan, Laura Beckwith, and Margaret Burnett
Oregon State University
{*subrahmn, ckissin, rectorky, inmand, kaplanja, beckwith, burnett*}*@eecs.oregonstate.edu*

Abstract

There has been little research into how end-user programming environments can provide explanations that could fill a critical information gap for end-user debuggers – help with debugging strategy. To address this need, we designed and prototyped a video-based approach for explaining debugging strategy, and accompanied it with a text-only approach. We then conducted a qualitative empirical study with end-user debuggers. The results reveal the influences of the explanations on end-user debuggers' decision making, how users reacted to the video versus textual media, and the information gaps the explanations closed. The results also reveal issues of particular importance to explanations of this type.

1. Introduction

Debugging is hard. Not only must programmers understand the program well enough to localize an erroneous piece of source code, they must also replace it with new source code that is error free and introduces no undesirable side effects. The complexity in debugging becomes even more challenging when the programmers doing the debugging have little experience, as is the case for many end-user programmers.

To help ameliorate the difficulty of debugging by end-user programmers, there are several recent research efforts in which the *system* carries out part of a debugging strategy. For example, some end-user debugging tools, such as WYSIWYT with fault localization for spreadsheets [7], the GoalDebug tool for spreadsheets [1], the WhyLine for event-oriented Alice programs and for debugging application settings [14, 17], and Woodstein for debugging web applications [20] are based on the program slicing strategy. However, none of these tools can debug entirely unaided, because tools cannot be sure of the right way to fix errors they find. Thus, in all of these approaches, user problem-solving is a necessary part of the process.

Since most end-user programmers are not trained in debugging, how to proceed with this problem-solving process is sometimes very unclear to them. Evidence that this is the case was demonstrated in a recent study we performed [13]. In that study, we analyzed the kinds of "information gaps"—explicit statements of an absence of information—expressed by end-user programmers when working on spreadsheet debugging tasks. Fully 30% of the information gaps expressed by the participants were about what *strategy* they should be using. For example, see Figure 1. In addition, another 9% of the information gaps were *self-judgment* questions, in which the participants questioned their own abilities to do the necessary problem-solving (e.g., "I'm not sure if we're qualified to do this problem").

Thus, almost 40% of the participants' information gaps related to how to problem-solve—even given tools that performed some of the problem-solving for them. Because of this evidence, we decided to investigate how to explain *debugging strategy* (as opposed to explaining *features*) to end-user programmers.

A strategy is a reasoned plan or method for achieving a specific goal—in this case, successful problem-solving about debugging. Teaching debugging strategy to end users who are in the process of debugging is especially challenging because the audience includes "active users" [9]. Active users are engaged in their task—in this case, debugging. As minimalist learning theory's paradox of the active user [9] points out, learning is somewhat counter to the goal of completing the task, because it takes away time from the task. Blackwell's model of attention investment [6] echoes this aspect, showing how users weigh the attention cost of learning something new against the risk that the learning effort will not cause time savings.

Given these issues, we began our investigations by turning to theories and to previous empirical data. Building upon these and other design constraints, we developed a prototype with two variants of debugging strategy explanations—video demonstrations and textual explanations—within a research spreadsheet environment. In this paper, we present our approach, along with its theoretical and empirical underpinnings. We then present the results of a think-aloud study. The

F. Let's change everything. [tries changing formula]
E. Yeah, but we got the right answer.
F. Did that change the answer at all?
F. Oh wait, did I change the symbol? [changes the formula]
F. Oh, now we're down to 30 percent tested.
F. I wonder if I go like that [changes the formula back]...

Figure 1: Participants E and F discuss the debugging strategy of "changing everything".

0-7695-2987-9/07 $25.00 © 2007 IEEE
DOI 10.1109/VLHCC.2007.18

IEEE computer society

overall research question this paper investigates is:

How can an end-user programming environment explain debugging strategy "just in time" to active end-user debuggers?

2. Related work

There is a large body of literature on on-line tutoring. Some of this work (e.g., [2]) provided us with insights for designing help in general. However, this body of work is aimed at audiences whose purpose is to learn per se. In contrast to this, our work is aimed at audiences whose purpose is to get something done, from which learning can be a distraction.

Turning to teaching strategy for complex computer applications, [8] emphasizes the need to infuse teaching of generalized strategies along with command knowledge, in order to increase the users' effectiveness with complex computer applications. However, the focus of that work is on optimal procedures, not strategies for deeper problem solving. Moreover their approach is proposed as a teaching style for teachers in schools, universities and companies whose focus is on teaching, not just-in-time instruction for active users engaged in a task.

The Whyline [14] debugging system aims at active users engaged in debugging. It answers "why" and "why didn't" questions about the system's state. In a later extension [17], this concept was applied to debugging about the behavior of complex commercial applications such as Word. In addition to answering "Why" and "Why didn't" questions, the system included a "How do I" link to change the behavior. The answer (usually changing particular preference settings) is not a strategy, but rather a fixed sequence of steps that stop the particular undesirable behavior. There has also been research on how to effectively enforce known sets of procedural instructions. For example, Stencils [12] is an interaction technique to direct users' attention to the correct interface component, preventing them from interacting with other components, thus requiring users to correctly follow the proper procedure. However, since debugging strategies cannot be constrained to a fixed set of interactions in the interface, this work does not fulfill our goals.

In the entertainment industry, games such as Crimson Skies and Jedi Knight: Jedi Academy include hands-on tutorials and demonstrations of features and strategies. In some, strategy hints are even offered during the game. The constraints for game strategy explanations differ from those for debugging strategy in several ways. The most important is the fact that the game is omniscient, having complete information about where dangers lurk and how to avoid them, whereas in debugging, a system cannot know with certainty where bugs are or how to fix them, and therefore cannot provide guidance based on such knowledge.

3. Explaining strategy and supporting self-judgment

As we have pointed out, our goal for explaining debugging strategy was to address the 30% of the information gaps from our previous study [13] that were about debugging strategy. We also included the self-judgment information gaps in our target (an additional 9% of the prior study's information gaps). These self-judgments are instances of metacognition, reflection about one's own knowledge state. Because of the importance of metacognition to learning and problem solving [11], we view it as critical to learning about and succeeding with strategies for debugging.

We emphasize that the goal of this research was to help end-user programmers with their *debugging strategies*. We were not interested in using these explanations to explain *features*. To support features per se, other explanations (described elsewhere [21]) already exist in our prototype, delivered in the form of tool tips. The tool tips provide support about how the features work and what their visual feedback portrays, and also offer an entry point to strategy explanations, but are not otherwise of interest in this paper.

3.1 Video demonstrations or static text?

What is the right medium for explaining strategies? We considered both recorded video demonstrations ("show me") and static textual explanations ("tell me"). The strengths of recordings, reported in [3, 18], seemed likely to provide at least three advantages over textual explanations: (1) videos can *demonstrate* procedure, (2) videos can potentially induce *vicarious learning experiences*, (3) the representations in videos actually look like the activities the users are doing (a *closeness of mapping* advantage). We were therefore convinced that video demonstrations would be indispensable. But owing to distinct advantages of text, such as ability to subdivide into smaller segments, allowing the user to immediately navigate to an aspect of interest, we included textual versions of the strategy explanations as well, so that users could mix them however they preferred.

3.2 Design constraints: "Show Me" strategies

The strategy information gaps of our previous study's participants [13], end-user programmers debugging spreadsheets, fell into six categories: (1) how to find errors, (2) how to fix errors, (3) how can I test my spreadsheet, (4) why should I change values, (5) what is a good overall strategy, and (6) am I doing it right. This list served as our requirements list for content coverage (i.e., *what* to cover).

As to *how* to cover this list, we turned to Plaisant and Shneiderman's guidelines on preparing recorded demonstrations [18], which led to the constraint set in

Constraint	Type
Provide procedural instruction rather than conceptual information	Semantic Content
Keep segments short (15 to 60 seconds)	Structure
Ensure that tasks are clear and simple	Semantic Content
Coordinate demonstrations with textual documentation	Syntax
Use spoken narration	Form
Be faithful to the actual user interface	Form
Use highlighting to guide attention	Form
Ensure user control	Structure
Keep file sizes small	Structure
Strive for universal usability	Form

Table 1: Guidelines for recorded demonstrations [18].

Table 1. Due to the length constraint (Table 1), we divided the above content into 6 video snippets. In some cases though, we still could not meet the length constraint—our videos range from 1-2.5 minutes.

Since these guidelines are predominately geared towards the syntax, form, and structure of recorded demonstrations—not their content, we added constraints on semantic content, based on data from our earlier study and on relevant theories (Table 2). Many of these constraints are specific to presenting strategy to end-users debugging. The complete collection of demonstrations can be viewed at: www.cs.orst.edu/~burnett/Forms3/RecordedDemos/

Figure 2 shows a video snippet of two people, a male and a female (lower left), problem solving and discussing a strategy (e.g., how to find errors) on a sample spreadsheet (rest of the figure). The actions they take and the effects on the spreadsheet are visible in the video. A male/female pair of demonstrators was chosen to encourage both males and females to identify with the demonstrators, potentially leading to vicarious experiences and boosting self-efficacy [4]. Because of the especially pronounced effects of self-efficacy on female end-user programmers [5], we particularly wanted to boost females' self-efficacy. Thus, the female sometimes takes on the role of a puzzled user who then follows up on a strategy hint, which ultimately leads to her success.

3.3 Design constraints: "Tell Me" strategies

We maintained most of the constraints above for the textual version of the strategy explanations, by converting each video snippet to a set of textual questions and answers, organizing them such that each snippet became a "category" of text. Each question answered by the video snippet was included in its corresponding textual category. An example is in Figure 3. We maintained exactly the same wording, with the exception of no example spreadsheet to which the text could refer and no demonstrators. Thus the design constraints relating to examples and to vicarious learning experiences could not be met by the textual explanations.

The text was in a resizable panel attached to the right of the spreadsheet. The user could select a topic and press either "Show me" or "Tell me". The panel was labeled "Strategy Hints" rather than "help," aiming to be acceptable even to those who do not like to admit to needing help.

3.4 Layering and navigation

We used the layering notions from minimalist learning [10], which suggest that information be revealed in "layers" that build upon previous information. We thus linked each video strategy explanation from tool tips for features playing a role in that strategy. For example, the Make Testing Progress video snippet was linked from the tool tips for arrows, the checkmark and the testing progress bar, which play roles in making

Table 2: Design constraints specific to strategy explanations for end users debugging.

Constraint	Type	Source / Rational
Present information in a concrete to general sequence	Semantic Content	Begin by showing a concrete "how to" example. The generalization then recounts the essence of the strategy, allowing application of the strategy to other cases.
Include head shot of person performing actions	Form	To give the user someone to relate to, presenting the opportunity for a vicarious experience aimed at improving the user's self-efficacy [4] and hence self-judgment capabilities.
Interpret feedback	Semantic Content	To avert problems in the accuracy of the users' self-judgments [13]. (Relates also to Norman's "Gulf of Evaluation".)
Mention oracle problem and what to do if values are hard to decide	Semantic Content	Empirical evidence shows that end users struggle with this issue when debugging (e.g. [13]).
Make benefits clear	Semantic Content	According to the model of Attention Investment [6], this will have a significant impact on follow-up actions the user will take.
Keep active user in mind	Semantic Content	Our explanations are intended to help "active users" finish debugging a spreadsheet now, not users whose primary goal is to learn about debugging [9].
Use informal terminology	Syntax	Intimidating vocabulary about debugging and program faults could interfere with the user relating to the demonstrators.
Verbalize the reasoning of the speaker	Semantic Content	To keep user apprised of the *reasoning* portion of the strategy (not just the UI actions), the demonstrator should think out loud.

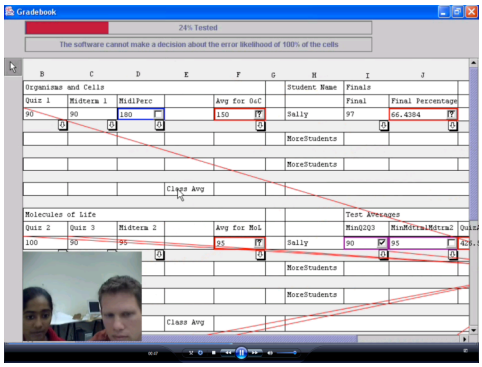

Figure 2: A video explanation.

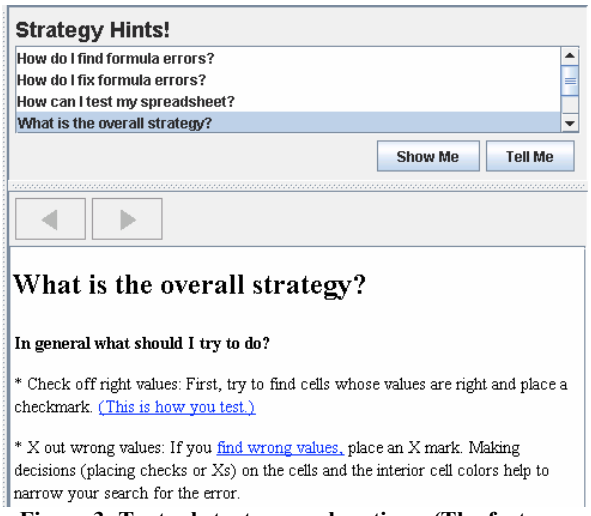

Figure 3: Textual strategy explanations. (The features mentioned are discussed in Section 4.2.)

testing progress. The text version had hyperlinks within each topic, enabling the user to navigate to further topics for deeper understanding.

According to the attention investment model [6], keeping users' perceived costs low increases their probability of using a feature. To keep perceived costs of *learning to use* and then *using* the explanations low, we used standard navigation devices for the videos and text. For example, the videos were played using Windows Media Player in a browser, so they could be paused/stopped/fast forwarded in standard ways. The snippets and text were short, and we displayed the time used by each video, to help users assess time costs.

4. Experiment

We made both of the above forms of strategy explanations, as well as the tool-tips, available to the participants at all times during their task. We used this set-up to investigate the following questions:

RQ1: Are the strategy explanations successful in closing strategy and self-judgment information gaps?

RQ2: In what ways do strategy explanations influence the participants' debugging strategy choices?

RQ3: What attitudes will users have about the video and textual explanations, and which will they prefer?

RQ4: What information processing styles will users follow with the strategy explanations?

RQ5: What layering will encourage perusal of the strategy explanations?

4.1 Participants and procedure

Strategy is a reasoned plan that exists only in the participant's head. Since we wanted to collect strategy data, we needed participant verbalizations, and hence used the think-aloud method. Each participant was audio and video taped. There were 10 participants: 7 males and 3 females. The participants were undergraduates from a variety of majors, excluding computer science. They were required to have prior experience using spreadsheets, but could not have much programming coursework or experience.

The participants took a 25-minute, hands-on tutorial, during which they debugged a sample spreadsheet. The tutorial was careful not to disclose either strategy or feature-oriented explanations. Participants were asked to click on both "Show me" and "Tell me", to get an idea of what such explanations contained. However they were allowed to view only a very small portion of the explanations, as we did not want to give away the strategy hints too early.

The main task was debugging Payroll, which contained 5 faults. The time limit was 50 minutes. After 20 minutes, they were briefly interrupted and asked to look at a strategy hint of their choice (either video or text). This was because our pilot studies suggested that some participants might not use any strategy explanations, which would have prevented collecting the data necessary for our research questions. At the end of the task, a post-session questionnaire gathered participants' opinions of the video and textual explanations.

4.2 Environment

The spreadsheet environment for debugging was Forms/3, including WYSIWYT (What You See Is What You Test) [7]. WYSIWYT is a collection of testing and debugging features that allow users to incrementally "check off" or "X out" values. When users check off a correct value, the cell and others contributing to it may become more tested, which is reflected by their border colors (Figure 4). X'ing out an incorrect value triggers fault likelihood calculations, which highlight cells likely to contain faulty formulas. In addition, arrows show dataflow relationships between cells and are colored to reflect testedness.

5. Methodology for analysis

Our methodology for analysis was largely based on codes. We devised our coding scheme, tested its stabil-

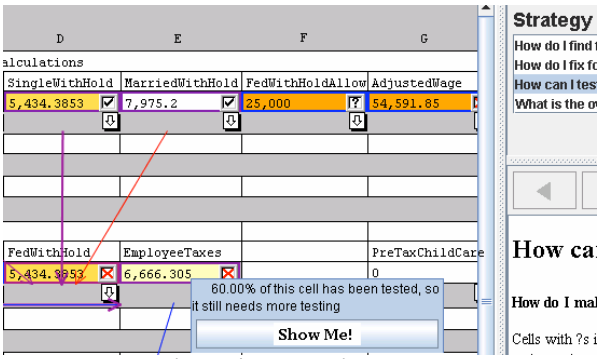

Figure 4: The environment.

ity on a portion of the data, and then applied the verified codes to the entire data set. We used three code sets to record the three types of data of interest to us: information gaps encountered, strategies used, and participants' reactions.

The information gap code set was reused from our previous study [13]. The ones of interest to this study are (1) strategy information gaps (expressing a need for strategy information), (2) self-judgment information gaps (expressing a negative opinion or lack of information about their own abilities), and (3) oracle/specification information gaps (difficulty deciding if a formula or value is right). The strategy code set was drawn from another study of ours, in which participants described their own strategies after debugging a spreadsheet task. The only codes of interest to this study are (1) testing (working with *values* displayed to find errors) and (2) code inspection (reading the *formulas* to find errors). Finally, we coded participants' reactions. We devised a simple code set to flag moments when the participants were noticeably frustrated, engaged or disengaged, clearly comprehending, or expressing confidence about their own abilities. Two of the researchers independently applied these codes to random segments of the transcripts. After a few test runs, agreement ranged from 89% to 96%, indicating that the codes were robust. At that point, one of the researchers completed the coding of the transcripts.

6. Results

6.1 Information gaps: what got closed?

We began by considering every instance in which a new information gap was opened. We then tracked each gap to see if the participant showed either verbally or through their actions that the particular gap had been resolved. For example, the following statement by participant F2 showed an information gap: "It's still purple. What am I missing?" (Purple told her that part of the cell's formula still had not been tested.) Later F2 closed this information gap:

> Let's turn on the thing [arrows on]…there we go. So I haven't tested [tool tips appears]. The relationship between G13 and H13 is 0% tested. Oh!!

There were 130 gaps opened in total. Of interest to this study were the 30% about self-judgment, the 25% about oracle/specifications, and the 7% about strategy. (The latter low number may be due to early viewing of the explanations.) Of the gaps opened, 49% of the self-judgment gaps, 75% of the oracle/specification gaps, and 56% of strategy gaps were closed.

The high rate of closing oracle/specification gaps surprised us. It may be due in part to the explanations' influences on the participants' debugging activities, which we discuss in the next section.

The number of information gaps opened and closed in each 10-minute interval (Figure 5) shows interesting patterns. According to curiosity theory [15], two types of advances in knowledge can trigger curiosity: incremental and insights. Incremental knowledge closes a gap or two, but does not lead immediately to a solution. Still, each increment in knowledge can produce more curiosity, raising new questions (gaps). This phenomenon, plus closing out initial easy gaps early on, matches the pattern until the third interval.

But what happened in the third and fourth intervals? Solving most challenging problems starts off incrementally [15], but eventually, a flash of insight can occur, resolving a number of gaps. The fourth interval seems to contain such insights, showing a spike in gaps closed ("Aha"!). According to the theory, when an insight is just around the corner is when curiosity rapidly increases [15], leading to a surge in gaps opened, as in intervals three and four.

6.2 Influences on debugging strategy choices

6.2.1 Direct influence on strategy choices

Because of the participants' freedom of choice, extensively analyzing direct influences was not possible. However, we were able to compare the effects in one situation that featured relatively uniform exposure: how choosing to read/view a strategy explanation during the tutorial impacted participants' strategy during the first 10 minutes of the task. Participants were free to access any feature, including explanations, during the last 2 minutes of the tutorial. The 3 participants who accessed explanations during that time chose the explanation on testing. We compared their strategies with the strategies of those who had not.

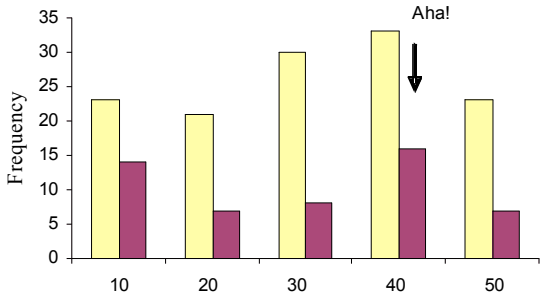

Figure 5: Gaps opened (light) and closed (dark) over time.

Figure 6 shows the frequency of *code inspection* and *testing* strategies for each participant. The circled participants accessed the explanation on testing, and they are the only three whose strategies rested heavily on testing. Fisher's Exact test (which is an appropriate test for small sample sizes) showed that the testing-explanation viewing participants were significantly heavier users of testing strategies than the other participants ($p=0.0083$).

6.2.2 Three categories of influences

Participants' gains and losses from strategy explanations fell into three categories: critical gains to success, partial gains, and misinterpretations.

In the first category, for two participants (males), strategy explanations played a particularly critical role in their success. Both started out debugging in a methodical fashion, but soon needed help to refine their strategies or to overcome specific information gaps. For example, participant M3 was confused about how border colors related to testedness.

> [cell turns from purple to blue] Interesting. I guess may be the blue means that I've checked so it's, I don't know, versus purple, versus red. I'm not sure.

After reading the explanation on testing, he was able to close this information gap and using the information to ultimately fix all remaining bugs.

> So of the cells that I've done, there are a few that are untested and a few that are tested, or need testing.

More than 50% of these two participants' information gaps were closed. In fact, 100% of M3's gaps were closed. Still, neither of these males chose explanations voluntarily—they read the explanations only when they were required to pick one at the 20-minute point. It remains an open question as to how to motivate users to use explanations when needed. Also, not all users need any help, as was true of another participant (M6, who solved all the bugs without use of hints from the explanations). One key may lie in understanding how to make the benefits clear [19] to motivate users to turn to explanations when needed.

In the second category (four participants), the explanations helped, but perhaps not enough. For example, participant M2 watched/read many of the explanations, but was usually impatient, thus taking only a little knowledge from an explanation. They did seem to help him find bugs, but increasing complexity of the bugs still to fix led to more and more difficulty and sinking confidence.

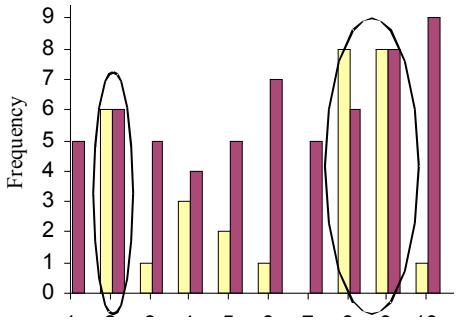

Figure 6: Influence of the testing strategy explanation. Code inspection: dark. Testing: light.

> The Monthly Adjusted Gross Pay is not correct [found a bug?]. Can't be sure of that. Ok, I get it—no, I don't. To be honest, I have never done anything to do with taxes.

Participant M2's impatience may have been due at least in part to a mismatch of the explanations to his information processing style. We will return to this point in Section 6.4.

In the third category, there were three participants for whom the strategy explanations helped at first, but at some point they misinterpreted an explanation. This led to a series of unfortunate events from which they could not recover, even with further visits to the explanations. For example, F3's strategy involved a lot of testing (encouraged by the video). But she became so involved in testing, she neglected the task of fixing bugs. She found bugs, but "fixed" them by hard coding the formulas so that they would produce the value she was looking for, so that she could continue testing.

> There is something I'm missing [changes LifeAmount to 10000] I think its 248. [changes cell reference to 248] times .08 [changes .01 in 'else' to .08] [changes .08 to .075].That's too high, so .079, .078... [changes to .0785, then .0784]...

This category suggests a need to steer misled people back on track, even when they do not realize they are off the track. For example, F3 never realized something was wrong. A challenge remains as to how to recognize debugging strategy misconceptions and direct users to a more effective strategy.

6.3 Video vs. text

6.3.1 Preferences

In rating their preferences on the post-session questionnaire, one participant strongly preferred video over text, and two participants strongly preferred text over video. Of the remaining participants, one preferred the two equally, and the rest had a mix of reactions, but most preferred text a little more than video.

We expected the videos to have a clarity advantage, and participants confirmed this. The most common positive comment was "They were clear to follow." In contrast, the most common positive comment about the text pertained to time cost of getting information: "Useful as a quick reference." The most frequent negative comment for *both* the videos and text was also about time, namely that they were time-consuming.

Discussion: The time-oriented comments underscore the participants' task-oriented attitudes, minimizing time learning in favor of doing [9], and also point to the cost/benefit tradeoffs of the attention investment model [6]. Note also that it is not possible to draw con-

clusions about preferences for *visual* media from this result. The videos were not just visual—they were different in many ways from textual explanations, using concrete examples, a mix of animated and static visual material, and audio (the spoken part).

6.3.2 Using videos vs. text: approach, follow-up

We analyzed video vs. text usage for two situations: approaching a topic for the first time, and following up on the same topic later. Surprisingly, the first-time approach to a topic was almost always done via text. In fact, 5 of the 10 participants used *only* textual explanations for first-time approaches. Four used both media for this, and one participant used only videos for first-time approaches. The participant counts were consistent with the instance counts: by any measure, for first-time approaches, the most popular medium was text.

For follow-up visits to a topic, the mix was different. The likelihood of follow-up being done via video versus text was almost identical. Of the five participants who followed up, one used videos for this purpose, and the rest used a mix of video and text.

There were multiple reasons for following up with an explanation. At one end of the spectrum was the need for a quick reference to a particular aspect. At the other end of the spectrum was a need for clarification via a detailed example, which could only be done using the video. For example, one of the female participants went back to the testing video to understand *how* to find different test situations.

> So how do I find the different situations? [Clicks on "show me"] Oh right, the arrow connects is a good one. Forgot about that.

These results make clear that the mix of text and video was very important to our participants.

6.3.3 Video vs. text: success and self-efficacy

Videos seemed to be particularly helpful to participants whose debugging performance was less than stellar (i.e., those who fixed fewer bugs than the median). The videos were marked as being helpful by all 5 of these "unsuccessful" participants, plus 1 of the successful participants. The text, on the other hand, was marked by all 5 of the successful participants but not as uniformly by the unsuccessful participants: 3 of the 5 unsuccessful participants marked it. Clearly, the successful participants derived less benefit from the videos, and hence did not need to bear the additional time cost, whereas for the unsuccessful participants, the opposite was the case.

Recall that one of the goals of the video was intended to be a vicarious boost of self-efficacy. A previous study has indicated that this may be particularly needed by females [5]. In fact, 2 of the 3 female participants (the third female did not use the videos) agreed with the questionnaire's query as to whether the videos "made me feel more confident". Interestingly, *none* of the 7 males marked this comment.

6.4 Information processing styles

How did participants process the explanations they accessed? Information selectivity theory provides some insights. Consider male M4, who brought up a video near the beginning of the task and watched some of it, but closed the video as soon as he learned something he could try (using checkmarks when a cell's values are correct and X-marks when they are wrong). His choice to return to the task without watching more of the video is consistent with "heuristic processing" [16], or following up one cue before looking for other cues. (This style is statistically associated with males.)

On the other hand, female F2 decided to re-watch a video she had watched in the tutorial. She navigated to the part of the video that discussed strategy when a cell turns purple. (She had purple cells in her spreadsheet at that point.) As soon as she re-watched this part, her face lit up, and she exclaimed "Oh!" Despite having gotten this relevant information, she did not stop the video; she continued to watch till the end. This style of comprehensively gleaning all possible information before moving onto apply it, is consistent with the "comprehensive" style of information processing [16]. (This style is statistically associated with females.)

Discussion: Gender differences aside, these two styles of information processing suggest the need for supporting both "modes" of viewing the explanations. For heuristic- (cue-following) style users, direct access to small snippets of information may be ideal. Our study may have served these users reasonably well with text but, since the videos were sometimes over two minutes long, the video snippets were probably better suited to comprehensive-style users.

6.5 Layering strategy explanations

Did layering matter? Recall that we supported both direct access to the strategy explanations (through the panel), and through "layers" [10] of additional information via the "Show me" button inside the tool tips and the hyperlinks among the textual explanations.

A count of the accesses to the strategy explanations shows that, for videos, participants used direct and layered accesses about equally. This suggests that providing layered access to make this medium accessible is quite important. It further suggests that adding strategy hooks to feature-centric help (e.g., the "Show Me" button in the tool tips) is an effective way to attract active users' interest to strategy explanations. However, in contrast to video access through layering, participants hardly used layering at all through the hyperlinks for the textual explanations.

Discussion: Hyperlinks among the textual strategy explanations are layering in the sense that they offer

new information to supplement existing information; still, the "distance" between layers is small in that both layers pertain to strategy. This is different from the tool tips/video connection, in which the tool tips were feature-oriented, and the offer ("Show me" button) was to provide information of a different type, namely strategy. Thus, the increment in information content promised may have seemed larger in the tool tip/video case, which can increase curiosity [15].

Another factor may have been the usefulness of the information just read. If a tool tip about a feature was not useful alone, the participants may have felt the need to find out about strategy. In the hyperlink case, having just gotten a strategy hint, these active users [9] may have felt more inclined to try it out on the task at hand than to read more strategy hints.

These results suggest that feature-to-strategy information layering is quite effective, but it is not clear if strategy-to-strategy information layering is effective.

7. Conclusion

In this paper, we have presented an approach, derived from theory and prior empirical results, to explaining debugging strategies to end-user programmers, and evaluated it. Some of our results were:

Positive influences: There was a statistically significant effect on participants' strategy choices. Females were also particularly responsive to the confidence-boosting goal. Regarding information gaps, 56%, 49%, and 75% of the strategy, self-judgment, and oracle/specification gaps, respectively, were closed.

Presentation: Pronounced differences in participants' use of different media (video versus textual) and information processing style (heuristic versus comprehensive) demonstrated the critical importance of supporting a mix of presentation choices.

Issues: The explanations were not a panacea. Issues included participant misinterpretations of the explanations and lack of motivation or interest in them. These issues identify interesting open questions for further research.

We emphasize that the above relate to cost-benefit-risk decisions users make, as Blackwell's model of attention investment predicts and our participants bore out. In fact, the participants' explicit focus on time costs particularly underscore the importance of considering attention investment and the "active user" audience in designing approaches to explain debugging strategies to end-user programmers.

References

[1] Abraham, R. and Erwig, M. Goal-directed debugging of spreadsheets, *IEEE Symp. Visual Langs. Human-Centric Comp.*, 2005, 37-44.

[2] Anthony, L., Corbett, A. Wagner, A., Stevens, S., Koedinger, K. Student question-asking patterns in an intelligent algebra tutor, *Conf. Intell. Tutoring Sys.*, 2004, 455-467.

[3] Baecker, R. Showing instead of telling, *SIGDOC,* 2002.

[4] Bandura, A. Self efficacy: Toward a unifying theory of behavioral change. *Psych. Review* 84(2), 1977, 191-215.

[5] Beckwith, L. Burnett, M., Wiedenbeck, S., Cook, C., Sorte, S., Hastings, M. Effectiveness of end-user debugging software features: Are there gender issues? *ACM Conf. Human Factors Comp. Sys.*, 2005, 869-878.

[6] Blackwell, A. First steps in programming: A rationale for attention investment models, *IEEE Symp. Human-Centric Comp. Langs. Envs*., 2002, 2-10.

[7] Burnett, M., Cook, C., Rothermel, G. End-user software engineering, *Comm. ACM*, Sept. 2004, 53-58.

[8] Bhavnani, K. S., Reif, F., John, B. Beyond command knowledge: Identifying and teaching strategic knowledge for using complex computer applications, *ACM Conf. Human Factors Comp. Sys.*, 2001, 229-236.

[9] Carroll, J. and Rosson, M. Paradox of the active user, In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, J. Carroll (Ed.), MIT Press, 1987.

[10] Farkas, D. Layering as a safety net for minimalist documentation, In Carroll, J. (Ed.), *Minimalism Beyond the Nurnberg Funnel,* MIT Press, 1998, 247-274.

[11] Forrest-Pressly, D., MacKinnon, G., Waller, T. *Metacognition, Cognition, and Human Performance,* Academic Press, 1985.

[12] Kelleher, C. and Pausch, R. Stencils-based tutorials: Design and evaluation, *ACM Conf. Human Factors Comp. Sys.*, 2005, 541-550.

[13] Kissinger, C., Burnett, M., Stumpf, S., Subrahmaniyan, N., Beckwith, L., Yang, S., Rosson, M. Supporting end user debugging: what do users want to know? *Advanced Visual Interfaces*, ACM Press, 2006, 135-142.

[14] Ko, A., Myers, B. Designing the Whyline: A debugging interface for asking questions about program behavior, *ACM Conf. Human Factors Comp. Sys.*, 2004, 151-158.

[15] Lowenstein, G. The psychology of curiosity, *Psychological Bulletin,* 116, 1, 1994, 75-98.

[16] Meyers-Levy, J. Gender differences in information processing: A selectivity interpretation. In *Cognitive and Affective Responses to Advertising* (Eds, Cafferata, P. and Tybout, A.) Lexington Books, Lexington, MA, 219-260.

[17] Myers, B., Weitzman, D., Ko, A., Chau. H. Answering why and why not questions in user interfaces, *ACM Conf. Human Factors Comp. Sys.*, 2006, 397-406.

[18] Plaisant, C. and Shneiderman, B. Show me! Guidelines for producing recorded demonstrations, *IEEE Symp. Visual Langs. Human-Centric Comp.*, 2005, 171-178.

[19] Venkatesh, V. and Morris, M. Why don't men ever stop to ask for directions? Gender, social influence, and their role in technology acceptance and usage behavior, *MIS Quarterly,* 24(1), 2000, 115-139.

[20] Wagner, E. and Lieberman, H. Supporting user hypotheses in problem diagnosis on the web and elsewhere, *ACM Int. Conf. Intell. User Interfaces*, 2004, 30-37.

[21] Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., Durham, M., Rothermel, G. Harnessing curiosity to increase correctness in end-user programming, *ACM Conf. Human Factors Comp. Sys.*, 2003.