A Strategy-Centric Approach to the Design of End-User Debugging Tools

Valentina Grigoreanu, Margaret Burnett
Oregon State University
valeng@microsoft.com, burnett@eecs.orst.edu

George Robertson
Microsoft Research
ggr@microsoft.com

ABSTRACT
End-user programmers' code is notoriously buggy. This problem is amplified by the increasing complexity of end users' programs. To help end users catch errors early and reliably, we employ a novel approach for the design of end-user debugging tools: a focus on supporting end users' effective *debugging strategies*. This paper has two core contributions. We first demonstrate the potential of a strategy-centric approach to tool design by presenting *StratCel*, a strategy-based tool for Excel. Second, we show the benefits of this design approach: participants using StratCel found *twice* as many bugs as participants using standard Excel, they fixed *four* times as many bugs, and all this in only a small fraction of the time. Furthermore, this strategy-based approach helped the participants who needed it the most: boosting novices' debugging performance near experienced participants' improved levels. Finally, we reveal several opportunities for future research about strategy-based debugging tools.

Author Keywords
Debugging strategies, debugging tools, end-user software engineering, tool design.

ACM Classification Keywords
D.2.5. Software Engineering – Testing and Debugging; H.5.m. Information Interfaces and Presentations: Miscellaneous.

INTRODUCTION AND RELATED WORK
End-user programmers are people who program, not as an end in itself, but as a means to more quickly accomplish their tasks or hobbies [20]. For example, an accountant creating a budget spreadsheet would fit this description. Many studies have found end-user programmers' code to be rife with errors (e.g., [21]) and the negative consequences of these errors have been reflected in numerous news stories, many of which are recounted at the EuSpRIG site [9]. One recent example that received media attention came following Lehman Brothers' collapse. Barclays Capital agreed to purchase some of Lehman's assets but, due to a spreadsheet error resulting from hidden cells, the company purchased assets for millions of dollars more than they had intended [14]. A few weeks later, Barclays filed a motion in court asking for relief due to the mistake.

The impact of end-user programming errors like the Lehman-Barclays example is amplified by the quickly increasing complexity of end-user programs and by the large number of end-user programmers. The complexity of corporations' spreadsheets doubles in both size and formula content every three years [32]. In addition, there are tens of millions more end-user programmers than there are professional programmers [26].

In response to this problem, end-user software engineering research has begun to emerge in the spreadsheet realm and in many other areas. Debatably, the first step in this direction was taken by Backus' team when it designed Fortran in 1954 [2]. Other examples include teaching kids to create programs (e.g., [7, 17]), programming for and over the web (e.g., [16, 25]), uncovering learning barriers [18], and even programming household appliances [24].

Of particular relevance to this paper are research spreadsheet debugging tools. The hidden structure of spreadsheets is an end-user debugging pain point [19] and tools such as Davis' overlaid arrows [8], Shiozawa et al.'s dependencies in 3D [28], and Igarashi et al.'s animated dataflow visualizations [15] have sought to address it. Tools which visualize broken areas (e.g., [27]) also aim to make the spreadsheet structure more transparent. Some debugging tools improve the automatic detection of errors (e.g., Abraham and Erwig's UCheck system [1]). Others empower the user to systematically test their spreadsheets using the What You See Is What You Test (WYSIWYT) testing methodology [6].

However, we believe that a critical stone has been left unturned in the design of spreadsheet debugging tools: how tools can be designed to directly support end-user programmers' existing *debugging strategies* (users' plans of action for accomplishing a task). Building upon a recent comprehensive overview of Excel users' debugging strategies [13], this approach led to the following main contributions:

- A novel empirically-based end-user debugging tool, StratCel, created to support end-user programmers' specific debugging strategy needs.
- A positive impact on end-user debugging success: (1) twice as many bugs found by participants using StratCel compared to Excel alone, (2) four times as many bugs fixed, (3) in a fraction of the time, (4) including two bugs which both the researchers and Control group had overlooked, and (5) a closing gap in success based on individual differences.
- Participants' promising comments about StratCel's usability and its applicability to their personal projects and experiences.
- Design guidelines, based on instantiated and validated empirically-based implications for design.
- Lastly, we argue for the generalizability of this approach and list several opportunities for future research.

STRATCEL'S EMPIRICALLY-BASED DESIGN

In this section, we address the question of whether a strategy-centric approach in the design of end-user debugging tools is practical and, if so, how it can be achieved. Toward this end, we report our experience building StratCel: an add-in for the popular end-user programming environment Microsoft Excel.

In the first subsection, we provide a quick overview of the iterative approach and methods we employed in StratCel's design. In the latter subsections, we then list several candidate design guidelines from a study which reveals a comprehensive overview of Excel users' debugging strategies [13]. We also detail how we employed these candidate guidelines in our design of StratCel to see which would prove effective: we later evaluate these.

Each time we refer to a candidate design implication *from that earlier study*, we format it as follows:

Candidate 0: This is an example implication from [13].

The implications for design revealed by the earlier study fell under three categories (hence the three subsections), based on the level of strategy from which they came: (1) a *strategy* is the user's approach for the entire task, which (2) one or more *strategems* can be used in combination to achieve, and which are in turn made up of (3) clusters of low-level *moves* with a purpose (i.e., *tactics*) [3]. For the remainder of this paper, we will use these more specific definitions of the four strategy levels.

Iterative Approach

As Schön points out, prototyping activities are important to any tool-building endeavor, since they encourage reflection on the tool's design [29]. We first defined the tool's scope using empirical work about end-user debugging strategies, a scenario, a storyboard, and sample real users from our target population. The sample users were real participants in a previous spreadsheet study [12]. For example, the most successful female was in her twenties and had worked as an auditor for the past two years, building large and complex spreadsheets to check clients' paperwork (e.g., bank statements and personal records). As a Business major, she also used spreadsheets in her classes and her personal life, and had programmed in VB.NET for one class. Continuing with an iterative approach, we cycled dozens of times through design, implementation, testing, integration, maintenance, and usability evaluation. To guide our iterations, we continued with the earlier methods and also added walkthroughs with a paper prototype, walkthroughs of the tool itself, and sandbox pilot sessions.

The Design Impact of Strategies and To-Do Listing

Implications for design based on the overall *strategies* can help us frame the functionality of the debugging tool as a whole, because strategies are followed by the user throughout the entire task.

Candidate 1: Supporting both comprehensive (getting an overall understanding of the spreadsheet by visiting cells in a systematic order) and selective (following up on the most

Finding	Evidence
To-do listing is an end-user debugging strategem.	Used breakpoints, open-close files, paper [11] and "*...checks and X's to show me what I'd already checked*" [31].
To-do listing is poorly supported in debugging tools.	PowerShell, Forms/3, and Excel: No explicit support for *to-do listing* [31, 11, 13].
Requests for to-do listing support transcend individual differences.	Males and females using Forms/3 [31], PowerShell [11], and even integrated development environments want *to-do listing* support [30].
Danger: Relying on existing features to be repurposed.	Misuse of the features can lead to incorrect feedback from tools [22], a loss of formatting information, or simply be ineffective. Perhaps why no participants from [13] employed it in Excel.
Benefit: Shows promise in increasing debugging success.	Often used in conjunction with *code inspection,* a female success strategem [31, 11]. May remind comprehensive Participant SF about cells she found suspicious and selective Participant SM about cells he had skipped over [12].

Table 1. Summary of empirical findings about the need to support *to-do listing* in debugging environments.

relevant clues as they come along) debugging strategies by:

- Helping comprehensive users keep track of cells they want to return to later on.

- Highlighting which cells selective users have looked at versus those they might have skipped.

In other words, support for the *to-do listing* strategem (or "a user's explicit indication of the suspiciousness of code, or lack thereof" [11]) may help reduce the cognitive load of both comprehensive and selective users by helping them keep track of items they need to look at in the future. Table 1 summarizes empirical findings from seven studies encouraging support for *to-do listing*. Note that, since both of these strategies needed to be supported, StratCel does not impose an order in which to proceed through to-do items or their related information.

Candidate 2: Provide explicit support for to-do listing.

Candidate 3: Automatically generate list of items to check.

To address these implications for design, the core functionality of StratCel involves automatically generating a list of to-do items and providing actions related to managing a task list, such as setting the item's status and priority (see Figure 1). Each item in the list is a range of consistent formulas automatically consolidated into one item. Using the tool, the user can change the status of each to-do item. Item status can be: (1) unchecked, meaning that the user has not yet made a decision about whether that item was completed, (2) checked, meaning that the user has verified that item and decided s/he does not need to return to it, and (3) to-do, meaning that the user would like to return to that item later on.

This explicit support for *to-do listing* helps guard against users having to use costly workarounds which change the spreadsheet's existing formatting. While the "automatic generation" implication seems to suggest that users would have less flexibility in creating their own to-do lists, storyboards and expert walkthroughs with the prototype backed the need for this implication.

Candidate 4: Provide relevant information in the context of each to-do item.

StratCel also automatically reports information about each item to help the user identify it, including: the worksheet name, an automatically generated name (from headers), a description pulled from cell comments, the item's priority, and the item's spreadsheet address. Following walkthroughs and sandbox pilots, we decided that the priority could be encoded in a color instead of having its own field in the list (see Figure 1c).

One important implication followed by other end-user debugging tools has been to directly overlay or tie hidden information about the structure of the spreadsheet to the spreadsheet itself (e.g., [27]). Therefore, in StratCel, we synchronized cell selection and to-do item selection:

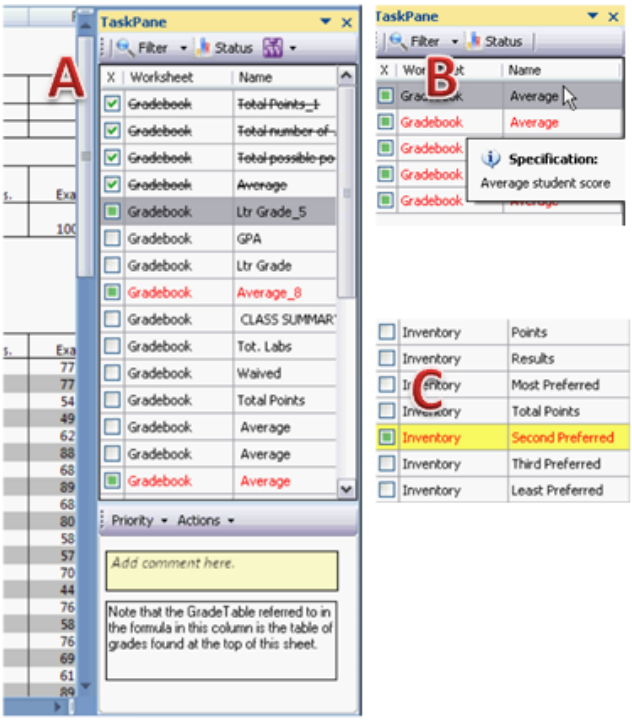

Figure 1. (a) The to-do list task pane is automatically populated with consolidated items and their properties (e.g., worksheet name, a default item name). The user can mark each item as "done" (e.g., Total Points_1), "unchecked" (e.g., GPA), or "to-do" (e.g., Average_8). Other basic to-do list management capabilities include adding a comment, (b) filtering on a status, and (c) assigning priorities to items (the darker the yellow, the higher the priority).

selecting an item in the list also highlights the cells to which that item refers, and vice-versa.

The Design Impact of Strategems

While *strategies* cover the entire task from start to finish, the debugging tool has multiple smaller components which further help make sure the task is accomplished accurately and quickly. For example, let us say that the first to-do item is about cell A1. Subtasks for checking off that particular item may include: examining the formula to make sure it matches the specification, testing different conditions and making sure the output is right for them, getting help when stuck, etc. These smaller components which allow users to act upon a unit of the to-do list are based on implications for design about end-user debugging *strategems* (e.g., *code inspection, specification checking*, *testing*, and *help* are strategems referred to in the previous sentence).

Candidate 5: Providing information about the nine remaining strategems in the context of each to-do item.

Researchers have so far observed ten end-user debugging *strategems*: *code inspection, control flow, dataflow, error checking, help, prior experience, spatial, specification checking, testing*, and *to-do listing*.

We have already addressed how *to-do listing* can be explicitly supported in order to facilitate the use of the *comprehensive* and *selective* debugging strategies. And *control flow* is the only strategem which StratCel does not support. Even though Excel's language is declarative, there is poor support for implementing repetition. How StratCel can better support this remains to be determined.

The remaining eight strategems are all supported in the context of each to-do item: each provides additional information about the item. For example, selecting an item in the to-do list also selects it in the spreadsheet. This displays a representative formula in the formula bar (*code inspection*) and highlights its value(s) in the spreadsheet (*testing*). Also related to formulas is the following:

Candidate 6: An easy way of accessing formulas related to the current code may help users fix more bugs through reuse.

To access more information related to the content of a formula, StratCel provides a "Help on Formula" feature to search several databases for information related to it (the *help* stratagem). Figure 2 shows the search result when looking up a formula containing both the 'IF' and 'HLOOKUP' functions in the Excel documentation (and three other information sources are also available). Another type of search which could be added to this list in the future is a search of Excel documents in a user-defined directory. This helps the user access the collective *prior experience*. Keeping track of done versus to-do items might help organize *prior experience,* while Excel's recently used formulas feature may highlight relevant formulas.

Candidate 7: Perfect viewing spatial and dataflow relationships to help users organize collected data.

Four strategems remain to be addressed: *dataflow, error checking, spatial*, and *specification checking*. A directed graph shows the *dataflow* dependencies between task items (see Figure 3). The graph can also be used to navigate the items; hovering over the items in the graph selects the related item in the task list as well as the related cell(s) in the spreadsheet. Since consistent formulas are highlighted as the graph is navigated, this also reveals the *dataflow* dependencies between *spatial* areas of consistent formulas. *Spatial* relationships can also be deduced from status borders: users can bring up borders around to-do items by clicking on a button in the ribbon. Areas of unchecked formulas are blue. To-do items are red. And items marked as "checked" have a green border. (There is an option for changing the colors to assist colorblind users.)

This way, inconsistent cells brought to the user's attention by the *feedback following* support (cells which have red borders originally), and also from the cells that get highlighted when a task item is selected.

Finally, item *specifications* are automatically generated from comments in the spreadsheet and can also be modified

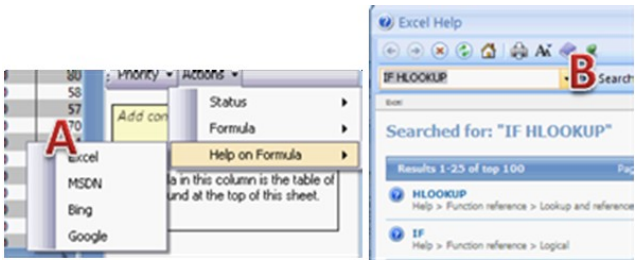

Figure 2. (a) "Help on Formula" gives the user several options (Excel Help, MSDN, Bing, and Google) in which to search key terms from the formula. For example, if the formula looks like this: =IF(E17<>"",HLOOKUP(E17,GradeTable,2),"") , then selecting Excel looks up "IF LOOKUP" in Excel's documentation. The same thing would happen with the other search engines.

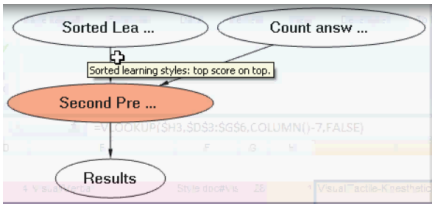

Figure 3. Dependencies between to-do items (recall that their names are automatically generated and can be modified) are displayed in a directed graph: StratCel visualizes both within and between worksheet transitions.

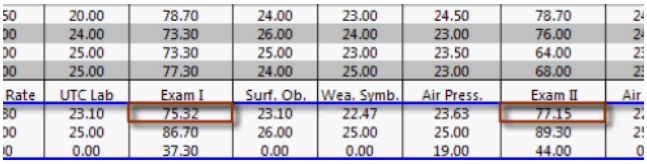

Figure 4. Users can bring up borders around to-do items by clicking on a button in the ribbon. Areas of unchecked formulas are blue. To-do items are red. And items marked as "checked" have a green border. (There is an option for changing the colors to assist colorblind users.)

by the user. They are displayed in the white box at the bottom of the task pane (see Figure 1a) and also in tooltips when hovering over items in the list (see Figure 1b) or in the navigation graph (see Figure 3).

The Design Impact of Tactics and Moves

Finally, implications for design based on observed *tactics* and *moves* are the lowest-level observations. As such, they are most applicable to fine-tuning the features implemented based on the *stratagem* implications.

For example, we mentioned a *dataflow* graph for navigating the spreadsheet. The tactic of navigating dependencies in Excel led to the following implication:

Candidate 8: Include inter-worksheet relationships.

Due to this implication for design, StratCel's dependency graph feature displays both inter-worksheet relationships between to-do items as well as intra-worksheet relationships. Hovering over the nodes in the different worksheets allows the user to navigate between those worksheets.

Candidate 9: Allow users to easily identify areas of the spreadsheet on which to focus their attention (e.g., formulas, buggy formulas, unchecked formulas).

To address this implication in StratCel, users can superimpose the to-do status of task items onto the spreadsheet. While we originally used a shaded circle in each cell to display the to-do status of that cell, walkthroughs revealed that this was overwhelming when the status of many cells was displayed. We therefore switched to only coloring the outside borders of spreadsheet areas with a particular status. For example, Figure 4 depicts an area of the spreadsheet with many unchecked formulas (blue borders) and two cells with to-do status (red borders).

Candidate 10: Too much feedback about where possible errors may lie is overwhelming, so only the most likely cells to contain errors should be highlighted by default.

StratCel currently automatically highlights inconsistent formulas by setting them as to-do items (see red items in Figure 1 and Figure 4), since those have a high likelihood of being incorrect. However, other Excel error checking warnings are ignored to reduce the false-positive rate of bugs found; sometimes, too much feedback is as bad as none at all. This lends support to the *feedback following* strategem (following the environment's feedback about where an error may be [11]).

EVALUATION

To gauge the success of employing a strategy-centric approach in the design of debugging tools, we conducted a preliminary evaluation of StratCel. In so doing, we wondered whether a strategy-centric approach to the design of debugging tools would lead to an increase in debugging success, whether StratCel was intuitive to use, and what design guidelines we could pass on to designers.

Experimental Setup

Procedure and Tutorial

We employed the same procedure as [12]. Participants first received a short (about 20 minutes) hands-on tutorial about Microsoft Excel's auditing tools and StratCel's functionality on a practice spreadsheet task. The StratCel functionality presented included selecting to-do items from the list, viewing information related to the item, marking the item as "done", "to-do", or "unchecked", and adding user-defined to-do items.

Task

The task was also the same as in [12]: "Make sure the grade-book spreadsheet is correct and if you find any bugs fix them." The grade-book spreadsheet contains 1718 cells, 288 of which were formula cells, and two worksheets: one for the students' individual grades and one for summary statistics for the class. The spreadsheet is also highly formatted, containing one blue column, one yellow column, four gray columns, 30 rows with alternating colors, three different font colors, 46 cells with bold fonts, five underlined fonts, many different font faces, and all borders delimiting spreadsheet regions.

This grade-book spreadsheet is real-world. It was selected from the EUSES Spreadsheet Corpus of real-world spreadsheets [10], originating from a college. In addition, it has been used successfully in other studies (e.g., [5, 12]).

While we originally thought the spreadsheet had ten nested bugs harvested from real users, as was reported in [12] and also based on our own experience, there were in fact 12 bugs in the spreadsheet (see the Results section for how our participants used StratCel to find two bugs which had previously been overlooked). These bugs were unintentionally introduced by the professor and by spreadsheet users from [5] when they attempted to add new features to this spreadsheet. There were: six inconsistency bugs (e.g., omitting some students' grades in calculating the class average for an assignment), three propagated logic errors (e.g., using the ">" operator instead of ">="), and *three* (instead of the expected one) logic bugs on individual cells (e.g., counted lab attendance as a part of the total points). The participants had a total of 45 minutes to find and fix these bugs.

Unlike in [12], where participants were provided a handout description of what different areas of the spreadsheet were meant to do, we incorporated the descriptions directly into the StratCel tool's white "specification" field (see bottom of Figure 1a).

Participants

In this pilot study of StratCel, we used five participants of varied backgrounds and spreadsheet experience. One male and one female were self-described novices, one male was a self-described intermediate, and two females were self-described experts. Our participants were members of two Seattle area clubs: the females came from a knitting circle and the males from an archery club. None of them had seen the new tool before the study. This was the group who had the StratCel Excel add-in available to them, and we will call them the "Treatment participants".

We compared their success to the eight participants from [12]. There, three males and three females were self-described spreadsheet experts and one male and one female described themselves as intermediates (no novices). We will call these participants the "Control participants".

There was no significant difference in any background variable between the Control and Treatment groups: age (Control median: 25, Treatment median: 25), major (Control: 6 non-CS science, 2 non-science; Treatment: 3

non-CS science, 2 non-science), and computer science experience (Control median: 0.5 classes, Treatment median: 0 classes). All thirteen participants had at one point edited spreadsheet formulas for work, school, or personal reasons.

However, two of the Treatment participants (one male and one female) did have less spreadsheet experience than was accepted in the Control group; they were self-described novices. We brought these two participants in for two reasons. First, we wanted to see how they would do in comparison to the experts from the other group. Second, we wanted to see how they would do against the experts in their own group.

Analysis Methodology

Since our data were not normally distributed, we employed the Wilcoxon rank-sum test with continuity correction in analyzing our quantitative data. This is non-parametric alternative to the t-test.

We also report qualitative observations about the participants' actions and verbalizations. These analyses helped both triangulate our quantitative findings and further explain the reasons behind the statistical differences.

Improving Debugging Success

The Treatment participants performed better by every success measure: the number of bugs found, the number of bugs fixed, the time to each bug find and bug fix, the reduced impact of individual differences, and participants' verbalized satisfaction with StratCel. To further help designers build better end-user debugging tools, we also highlight those empirically-based *guideline candidates* which had the biggest impact on our participants' success by listing them as *design guidelines* in this subsection.

Number of Bugs Found

In general, participants who had StratCel available to them were better at finding bugs. They found more bugs, including two previously unnoticed bugs, faster, and with less variability resulting from individual differences.

Specifically, Treatment group participants found significantly more bugs (Rank-sum test: Z=-2.639, p=0.0042) than the Control group participants. Figure 5 shows the distribution of bugs found by Control participants (*M:* 4.50, *SD:* 2.70) and Treatment participants (*M:* 9.00, *SD:* 0.89). This difference is striking: only one of the participants from the Control group found nine bugs, whereas all of the Treatment participants found at least nine bugs. (Both Treatment novices performed at least as well as the Control experts.)

Qualitative observations of *how* participants used StratCel revealed several reasons for this sharp increase in bug finding success. The first was Candidate 10: *Too much feedback about where errors may lurk is as bad as no feedback at all.* Since StratCel set inconsistent formulas as to-do items by default, all five participants found those six bugs. For example, to do this, the intermediate male

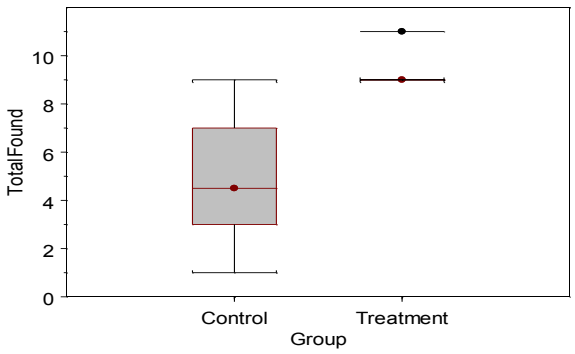

Figure 5. Participants in the Treatment group (right) found significantly more bugs than the Control group participants (left).

participant immediately filtered the task list to only show items automatically set as to-do: inconsistent formulas. Figure 1b shows his list right after filtering.

The novice Treatment male and an experienced Treatment female employed our response to Candidate 9 to find the inconsistent formulas: *Easily find areas of the spreadsheet on which to focus their attention*. He brought the status borders up immediately at the start of the task to view the items which were automatically given to-do status (i.e., a red border).

The remaining two female participants (one novice and one expert) used a different method: they both walked through the list one item at a time, starting at the top, and only took on inconsistency items once they reached them in the to-do list. One mentioned she was also able to tell where inconsistencies laid based on the address of each to-do item being shown. For example, if an item covered the range from "A1:A3, A5" that is what showed up in the "address column" of that to-do item. This allowed her to quickly notice A4 was missing, which therefore must have been an inconsistent formula:

> *"This was really helpful because it has a way to say these are all your formulas… These are the ones you need to go look at. And I like this part [the address field] which shows me where I can find all of the formulas, so I can see them. For example, on this one, I could see there was a gap for E16 and I could go back and look specifically at that cell, because I expect it to be the same, and see what's going on."*

Overwhelmed by the number of false-positive bug warnings (Excel green triangles in cell corners), most of the Control group participants were unable to find these inconsistencies. Our Treatment participants, however, found inconsistencies in the spreadsheet much more easily (all five participants found and fixed all six inconsistency errors) and in a variety of ways. Thus, we would like to reiterate three of the

empirically-based candidate guidelines mentioned earlier but, this time, as validated design guidelines for end-user debugging tools:

> ***Design Guideline 1:*** *With automatic error detection tools, it is critical to value quality (low number of false-positives) over quantity (detecting more possible types of errors). Only cells containing likely errors should be highlighted by default.*
>
> ***Design Guideline 2:*** *As most tools currently already do, important information about cells (e.g., to-do status) should be overlaid onto the spreadsheet to give the user a quick overview of the to-do status of both individual cells and of the overall spreadsheet.*
>
> ***Design Guideline 3:*** *Some users prefer to get a comprehensive understanding of the spreadsheet before fixing bugs (e.g., the novice female), whereas others will start by trying to fix apparent bugs right away (e.g., the intermediate male). Since both approaches have advantages and disadvantages, both should be supported.*

All participants found at least nine bugs. Other than the six inconsistency bugs, there were four other bugs which the researchers had inserted [12] and two more which were not observed by either the researchers or the Control participants, but which were found and fixed by the users in this study! These unnoticed bugs, while fairly easy to fix once spotted, were well-hidden: one individual cell was in the upper-right corner of the spreadsheet, and the second was hidden in the middle of the second worksheet.

These two previously evasive bugs were the crowning glory of the usefulness of StratCel in bug finding: some hidden bugs can evade the eyes of many experts and novices alike. However, the to-do list enabled participants to give an equal amount of attention to each item: even items in the top-left corner of the first worksheet and cells in the middle of the second worksheet.

> ***Design Guideline 4:*** *Strategy-based tools should provide explicit support for to-do listing.*
>
> ***Design Guideline 5:*** *To improve debugging of end-user programs, it helps to automatically generate a list of items to check so that all areas of the code are given equal attention.*

Number of Bugs Fixed

Just as with the number of bugs found, Treatment participants also fixed significantly more bugs (Rank-sum test: Z=-2.8905, p=0.0019) than the Control group participants. Figure 6 shows the distribution of bugs fixed by Control participants (*M:* 2.00, *SD:* 2.3299) and Treatment participants (*M:* 8.00, *SD:* 1.3038). Thus, while Treatment participants found twice as many bugs on average than Control participants, the difference in bugs fixed is even more striking: Treatment participants fixed *four times* more errors on average! (This time, the male and female Treatment novices performed better than even the most successful Control participant.)

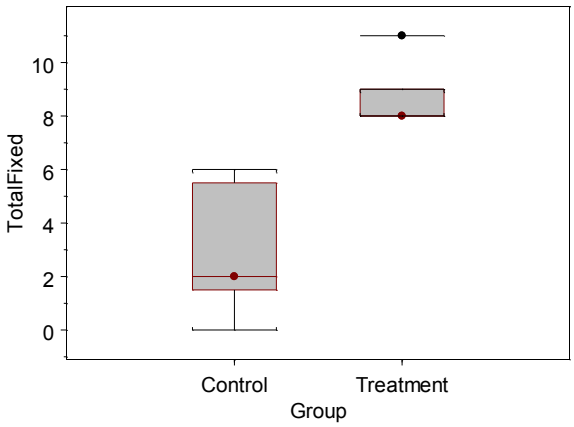

Figure 6. Treatment participants (right) fixed significantly more bugs than Control participants (left).

What caused the striking difference in the number of bugs fixed? A major contributor was that Treatment participants had found more bugs, therefore also having the opportunity to fix more. Furthermore, the six inconsistency bugs were trivial fixes once the users had found them. Had the Treatment group participants only fixed the inconsistencies, they would have already fixed three times more bugs than the Control participants on average.

The two to five additional bug fixes varied by participant, but the methods by which they were fixed always involved the additional information given in the context of an item. For example, the intermediate male used Excel's "Recently Used" function library to find a formula used in a different spreadsheet (the tutorial spreadsheet) which could have been used to fix one of the most complicated bugs in the spreadsheet. All of the participants employed the descriptions provided for each item. These helped them fix two bugs consistently: two bugs on individual cells which were easy to overlook without StratCel pointing them out, but straightforward to fix once there (two cells incorrectly took into account labs as a part of the total grade): none of the Control participants found or fixed either of those bugs, and the researchers only knew about one of the two. Each of the features available in StratCel was used by at least one participant, backing the importance of showing related information in the context of each to-do item.

> ***Design Guideline 6:*** *Information about the remaining strategems should be provided in the context of each to-do item to provide more information on which to base a bug fix.*
>
> ***Design Guideline 7:*** *Viewing formulas related to an item (e.g., the consistent formulas in an inconsistency*

case, recently used formulas, or formulas used in files in a certain directory) might be particularly useful for improving debugging success.

Time to Each Bug Find and Fix

Spreadsheet debugging is often a time-sensitive activity, whether a trained accountant does it [23] or a young clerk as was the case in the Lehman-Barclays mix-up. Thus, another important measure of debugging success in addition to the number of bugs found and fixed is how long it took participants to find and fix those bugs.

On average, Treatment participants found and fixed each bug consistently faster than the Control participants. The Wilcoxon rank-sum test allows us to measure statistical difference in bugs found and fixed based on order, without worrying about missing data such as those of participants who never found or fixed a bug.

The advantage of Treatment participants was clear from the very beginning of the task. Treatment participants found the first bug significantly faster (Rank-sum test: $Z=2.62$, $p=0.0044$) and fixed it significantly faster (Rank-sum test: $Z=2.8663$, $p=0.0021$) than the Control participants. Treatment participants also found and fixed all of the remaining bugs significantly faster than Control participants (up to the tenth bug found, after which there was not enough data to prove significance, with only one Treatment participant finding and fixing eleven bugs total).

Thus, when time is short, StratCel users should be able to more quickly pinpoint errors and their solutions from the very start and keep that advantage throughout the task. It also appears that the more complex the spreadsheet is, the more useful StratCel will become, though this remains to be tested in future studies.

Closing Gaps Based on Experience and Gender

Another surprising discovery was that the Treatment participants performed very similar to one another, despite their individual differences. In previous studies on end-user debugging, both gender (e.g., [4]) and experience (e.g., [11]) have impacted end-user debugging success.

Also, recall that even the novices from the Treatment group performed at least as well as the most experienced and successful Control participants. When comparing Treatment novices to Treatment experts, there was little variation between the Treatment particiants, despite their very different backgrounds: the SD was twice as great for the Control group than the Treatment group. Treatment novices did not do much worse than Treatment intermediates and experts. In particular, for the Control group, bugs found ranged from 1-9 and bugs fixed from 0-6. In the Treatment group, bugs found ranged from 9-11 and bugs fixed from 8-11. Since there is a much less pronounced difference between the less experienced and the more experienced participants in the Treatment group, it appears that StratCel helps everyone, and especially less experienced users. The following quote comes from the novice Treatment female:

"I feel like it would be extra useful for someone like me who, well, I can use Excel and I can figure it out, but, like, I'm definitely not an expert at Excel. […] I think the only problems I had were with the Excel functions I hadn't learned. This is like a really good way of helping me keep track of what I've done and not get lost."

In terms of gender, comparing the median number of bugs found (CF: 4.5, TF: 9.0, CM: 5.0, TM: 9.0) and fixed (CF: 3.5, TF: 9, CM: 2.5, TM: 8.5) by females and males in the Control and Treatment groups, we noticed that there were few gender differences between them. Even so, Treatment participants were a little closer to each other than Control participants in terms of success: meaning that StratCel helped both males and females.

Overall Experience: StratCel's Usability

While we did not ask our participants for feedback beyond their verbalizations during the task, the participants were nevertheless anxious to give it.

Several comments revealed possible iterative improvements to the tool. For example, participants had a feature available to add to-do items to the automatically generated list. The most successful Treatment female used it as a way to add two comments for the next person who will look at the spreadsheet: one about how little she trusts the spreadsheet and a second about a change she would have liked to have made to one of the formulas in the future. The most successful male also added a custom to-do item, but he did so by mistake. Their feature request was to add the functionality of *removing* items from the to-do list.

Another improvement requested by the two experienced females was the capability to sort the to-do list by clicking on the field headers. One of the potentially most critical problems with the to-do functionality is that it is too easy to check off items as done, to never be returned to again. One of the experienced females put it this way:

"The only thing that I was thinking about is that it's really easy to say 'Oh, I've looked at this.' and just check it off. And I don't know if there could be a way to make sure that that's what they meant. […] So, I actually had something… Where I went through, and I think I'm on one line but I'm actually on another when I check off the task being done. But I think that's just... A user has to be smart enough to know not to do that. There's only just so much that you can help a user avoid."

One possibility for making sure that the user really meant to check something off would be to list each of the "strategem tool components" (e.g., the specification) as individual subtasks for each task. This way, users would have to check off several subtasks in order to achieve an overall "check"

for the item. Further research is needed to what the best method is for doing this.

Overall, however, the participants' unrequested comments were very positive, and most immediately thought of ways to apply StratCel to their own day-to-day tasks. Here are selected few of the quotes:

- *"So, can I use your tool? You should sell this and make a million dollars!"*
- *"I think this would be useful for my complex accounting spreadsheets. If you would like to share the tool, I would love to try it on those."*
- *"Looking at [StratCel], I was thinking I have to have a way of tracking my [knitting] patterns. So things that… Ok. I have a pattern and I have steps I have to go through. And I need a way to track them."*
- *"And this is straight-forward and makes a lot of sense. When you look at it, you know what it is. There are lots of tools, where you can tell that people said, 'well… there's just a workaround and you can just do it this way'. But this one, it just seemed very straightforward and it builds on everything from Excel."*

CONCLUSIONS AND FUTURE WORK

In this paper, we have shown that a *strategy-based approach* alone can be effectively applied in the design of debugging and troubleshooting tools to improve the correctness of end-user programmers' code.

As a part of this effort, we instantiated our approach in StratCel: a new strategy-based add-in for one of today's most widely used end-user programming environments, Excel. StratCel addresses implications for design at four strategy levels.

Our results showed that tools can be built to support a comprehensive understanding of strategies directly. We employed implications derived from higher strategy levels (strategems and strategies) to frame the functionality of the tool as a whole, while implications based on lower levels of strategy (moves and tactics) helped us fine-tune individual features. For example, support for the *to-do listing* strategem provided a way to reduce end-user programmers' cognitive load, by helping *comprehensive* participants better keep track of to-do items to revisit and by helping *selective* participants see which formulas they had skipped. The remaining nine strategems defined the core activities which needed to be supported within the context of each to-do list item (e.g., specifications, help about the formula as a whole, etc.) in our instantiation. Finally, the implications from the lower strategy levels (moves and tactics) helped us fine-tune the features supporting each strategem: for example, making sure that the *dataflow* dependencies showed inter-worksheet relationships and facilitated navigating between items on different worksheets.

Even for an environment as mature as Excel, the addition of a strategy-based tool did improve end-user programmers' debugging success using many measures:

- Participants who had StratCel available to them found twice as many bugs, fixed four times as many bugs, and in only a fraction of the time.
- While StratCel helped everyone, it was particularly helpful to less experienced users. StratCel also helped males and females equally.
- Participants found StratCel intuitive to use and immediately thought of ways in which the tool applied to their day-to-day work.

This approach to end-user debugging tool building has raised many questions, opening the door to opportunities for future research.

- The current instantiation of StratCel centers on the *to-do listing* strategem, supporting the other strategems within the context of each to-do item. A future goal might be to create a new tool which centers around one of the other strategems (say *code inspection* or *testing*) and which supports all other nine strategems within the context of either a formula or of an output value, in those two cases respectively. Would the addition of another strategy-centered tool improve users' success even further?
- Even within its current instantiation of the implications for design, each of StratCel's components can be improved with further research. For example, StratCel currently only highlights inconsistency errors, but both Excel and other tools provide many other automatically generated warnings. An ordered list of the available automatic spreadsheet error detection algorithms and their false-positive rates, would be required to further improve the *error checking* component, in order to know which algorithms to turn on by default.
- Finally, related empirical work has drawn parallels across programming populations and environments: from spreadsheets, to scripting environments, and integrated development environments (recall Table 1). Can StratCel's core functionality be transferred to one of these other environments? If so, will it also lead to increased debugging success there? Do these concepts change when users are not able to manipulate the code directly and have to work at a higher level of abstraction (e.g., when troubleshooting a printer failure)?

In summary, we have shown that a strategy-based approach to building debugging tools is both achievable and beneficial. Powerful but disconnected features may be the approach of the past, and be replaced by features which work together to support users' effective debugging and troubleshooting strategies.

ACKNOWLEDGMENTS

We would like to thank Roland Fernandez for his thoughtful input on this paper's write-up. And we are also thankful to the participants of our study. This work was supported in part by the EUSES Consortium under NSF 0325273 and by NSF 0917366.

REFERENCES

1. Abraham, R. and Erwig, M. UCheck: A spreadsheet unit checker for end users. *Journal of Visual Languages and Computing 18*, 1 (2007), 71-95.
2. Backus, J. The History of Fortran I, II, and III. *IEEE Annals of the History of Computing 20*, 4 (1998), 68-78.
3. Bates, M. Where should the person stop and the information search interface start? *Information Processing and Management 26*, 5 (1990), 575–591.
4. Beckwith, L., Burnett, M., Wiedenbeck, S., Cook, C., Sorte, S., and Hastings, M. (2005) Effectiveness of end-user debugging software features: Are there gender issues? In *Proc CHI 2005*, ACM Press (2005), 869-878.
5. Beckwith, L., Inman, D., Rector, K., and Burnett, M. On to the real world: Gender and self-efficacy in Excel, In *Proc. VL/HCC 2007*, IEEE Press (2007), 119-126.
6. Burnett, M., Cook, C., and Rothermel, G. End-user software engineering. *Communications of the ACM 47*, 9 (2004), 53-58.
7. Cypher, A. and Smith, D. KidSim: End-user programming of simulations. In *Proc. CHI 1995*, ACM Press (1995), 27-34.
8. Davis, J.S., Tools for spreadsheet auditing, *International Journal of Human-Computer Studies 45* (1996), 429-442.
9. EUSPRIG 2009. Spreadsheet mistakes news stories, European Spreadsheet Risks Interests Group site, http://www.eusprig.org/stories.htm.
10. Fisher II, M. and Rothermel, G. The EUSES Spreadsheet Corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanism. In *Proc. 1st Workshop on End-User Software Engineering* (2005), 47-51.
11. Grigoreanu, V., Brundage, J., Bahna, E., Burnett, M., ElRif, P., and Snover, J. Males' and females' script debugging strategies. In *Proc. Second International Symposium on End-User Development*, Springer-Verlag (2009), 205-224.
12. Grigoreanu, V., Burnett, M., Wiedenbeck, S., Cao, J., and Rector, K. Females' and males' end-user debugging strategies: A sensemaking perspective. Technical Report: http://hdl.handle.net/1957/12074 (2009). (Under Review)
13. Grigoreanu, V., Burnett, M., and Robertson, G. (2009) Design implications for end-user debugging tools: A strategy-based view. Technical Report: http://hdl.handle.net/1957/12443 (2009). (Under Review)
14. Hayes, F. Rules for Users. http://www.pcworld.com/businesscenter/article/152509/rules_for_users.html, 2008.
15. Igarashi, T., Mackinlay, J., Chang, B. W., and Zellweger, P. Fluid visualization of spreadsheet structures, In *Proc. Symposium on Visual Languages 1998*, IEEE Press (1998), 118-125.
16. Kandogan, E., Haber, E., Barrett, R., Cypher, A., Maglio, P., and Zhao, H. A1: end-user programming for web-based system administration. In *Proc. UIST 2005,* ACM Press (2005), 211-220.
17. Kelleher, C., Pausch, R., and Kiesler, S. Storytelling alice motivates middle school girls to learn computer programming. In *Proc CHI 2007*, ACM Press (2007), 1455-1464.
18. Ko, A., Myers, B., and Aung, H. Six learning barriers in end-user programming systems. In *Proc. VL/HCC 2004,* IEEE Computer Society (2004), 199-206.
19. Nardi, B. and Miller, J. The spreadsheet interface: a basis for end user computing. In *Proc. INTERACT 1990*, Elsevier (1990), 977 – 983.
20. Nardi, B. *A small matter of programming: Perspectives on end-user computing*. MIT Press, 1993.
21. Panko, R. and Orday, N. Sarbanes-Oxley: What about all the spreadsheets? European Spreadsheet Risks Interest Group, 45 pages, 2005.
22. Phalgune, A., Kissinger, C., Burnett, M., Cook, C., Beckwith, L., and Ruthruff, J. Garbage in, garbage out? An empirical look at oracle mistakes by end-user programmers, In *Proc. VL/HCC 2005*, IEEE Press (2005), 45-52.
23. Powell, S., Baker, K., Lawson, B. An Auditing Protocol for Spreadsheet Models, *Information and Management 45*, 5 (2008), 312-320.
24. Rode, J., Toye, E., and Blackwell, A. The fuzzy felt ethnography - understanding the programming patterns of domestic appliances. *Personal and Ubiquitous Computing 8*, (2004), 161-176.
25. Rosson, M., Sinha, H., Bhattacharya, M., and Zhao, D. Design planning in end-user web development. In *Proc. VL/HCC 2007*, IEEE Computer Society (2007), 189-196.
26. Scaffidi, C., Shaw, M., and Myers, B. Estimating the number of end users and end-user programmers. In *Proc. VL/HCC 2005*, IEEE Computer Society (2005), 207-214.
27. Sajaniemi, J. Modeling spreadsheet audit: A rigorous approach to automatic visualization. *Journal on Visual Languages and Computing 11*, 1 (2000), 49-82.
28. Shiozawa, H., Okada, K., and Matsushita, Y. 3D interactive visualization for inter-cell dependencies of spreadsheets. In *Proc. InfoVis 1999*, IEEE Press (1999), 79-82.
29. Schön, D. *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books, 1983.
30. Storey, M., Ryall, A., Bull, R., Myers, D., and Singer, J. TODO or to bug: Exploring how task annotations play a role in the work practices of software developers. In *Proc. Software Engineering 2008*, 251-260.
31. Subrahmaniyan, N., Beckwith, L., Grigoreanu, V., Burnett, M., Wiedenbeck, S., Narayanan, V., Bucht, K., Drummond, R., and Fern, X. Testing vs. code inspection vs. what else? Male and female end users' debugging strategies, In *Proc. CHI 2008*, ACM Press (2008), 617-626.
32. Whittaker, D. Spreadsheet errors and techniques for finding them. *Management Accounting 77*, 9 (1999), 50–51.