Testing vs. Code Inspection vs. ... What Else? Male and Female End Users' Debugging Strategies

Neeraja Subrahmaniyan, Laura Beckwith, Valentina Grigoreanu, Margaret Burnett, Susan Wiedenbeck, Vaishnavi Narayanan, Karin Bucht, Russell Drummond, and Xiaoli Fern

Oregon State University
Corvallis, OR, USA
{subrahmn,beckwith,grigorev,burnett,narayava,xfern}@cs.orst.edu

Drexel University
Philadelphia, PA
Susan.Wiedenbeck@cis.drexel.edu

ABSTRACT

Little is known about the strategies end-user programmers use in debugging their programs, and even less is known about gender differences that may exist in these strategies. Without this type of information, designers of end-user programming systems cannot know the "target" at which to aim, if they are to support male and female end-user programmers. We present a study investigating this issue. We asked end-user programmers to debug spreadsheets and to describe their debugging strategies. Using mixed methods, we analyzed their strategies and looked for relationships among participants' strategy choices, gender, and debugging success. Our results indicate that males and females debug in quite different ways, that opportunities for improving support for end-user debugging strategies for both genders are abundant, and that tools currently available to end-user debuggers may be especially deficient in supporting debugging strategies used by females.

Author Keywords

Gender, debugging, end-user programming, end-user software engineering, strategy.

ACM Classification Keywords

D.2.5 [Software Engineering]: Testing and Debugging; H.1.2 [Information Systems]: User/Machine Systems—Human factors; H.4.1 [Information Systems Applications]: Office Automation—Spreadsheets

INTRODUCTION

What strategies do end-user programmers use when debugging? Does gender make a difference in these strategies?

Research has begun to report differences in males' and females' behaviors with software. In end-user programming, gender differences have been reported in programming environment appeal [15], playful tinkering with features [5], and attitudes toward and usage of end-user software design and debugging features [4, 6, 22]. Still, these results fail to say how males and females would *like* to approach debugging. Studying environment features, as in previous research, allows consideration of only that which exists. This paper looks beyond features to debugging strategies, to consider that which may *not* yet exist, but should.

This paper reports the results of an experiment we conducted to investigate the strategies used by male and female end-user programmers in the course of debugging spreadsheets. Strategy, which refers to a reasoned plan or method for achieving a specific goal, exists in the head, and in-the-head data are not easy to obtain reliably. Hence, our experiment is heavily triangulated, using four sources of data—questionnaire data, session replays, logged behaviors from the current experiment, and logged behaviors from a previous data set. Questionnaire data were used primarily to elicit participants' in-the-head strategies, which are not directly observable; behavior data, which are directly observable, were used to corroborate the presence of these strategies. We used three methods of analysis: qualitative methods, traditional statistical methods, and unsupervised automated data mining methods.

We were interested not only in strategies used by end-user programmers, but also in the relationships of these strategy choices to males' and females' debugging success. Therefore, our research questions were:

RQ1: What debugging strategies do end-user programmers try to use?

RQ2: Are there gender differences in the debugging strategies male and female end-user programmers try to use?

RQ3: Which debugging strategies lead to male and female end-user programmers' success?

BACKGROUND AND RELATED WORK

Past research provides reasons to suspect that end-user programmers might use debugging strategies other than those of professional programmers. It also provides reasons to suspect gender differences in males' and females' strategies, and the ties of these strategies to success. Specifically,

CHI 2008, April 5–10, 2008, Florence, Italy.

we encountered three reasons to suspect such differences. First, end-user programmers have elements in common with novice programmers, and there are known differences between these two groups' strategies [17, 19]. Second, research from other domains, both theoretical and empirical, suggests gender differences relevant to debugging strategies [4, 18, 20]. Third, recent findings of gender differences in feature usage by end-user programmers could actually be evidence of differences in males' and females' intended strategies [4, 6, 13, 22].

Regarding the first reason, we use the term "end-user programmers" to describe people who program in order to facilitate their goals, but without aspiring to become professional programmers. End-user programmers have some attributes in common with the more widely studied group known as "novice programmers." This term refers to people who, like end-user programmers, have little programming experience, but unlike end-user programmers, aspire to become expert programmers. Programming literature indicates that both experts' and novices' debugging success is related to program comprehension. Specifically, expert programmers attempt to gain a high-level understanding of the program before they begin to debug, whereas novices tend to jump immediately into the code, with very little context about the program's purpose and structure (e.g., [19]). When experts on occasion do jump immediately into the code, they, like the novices, are less likely to succeed [17].

Regarding the second reason, and pertinent to systematic program comprehension, research in the field of marketing reports gender differences in information processing. The Selectivity Hypothesis [18] proposes, and empirical research (e.g., [20]) bears out, that males and females differ in their information processing in decision making. Specifically, the hypothesis proposes that females tend to maximize the comprehensiveness of their information processing (e.g., attending to details, looking for multiple cues, and making elaborative inferences) in simple and complex tasks. In contrast, males have a propensity to use simple heuristics in information processing (e.g., single cues that are readily available) to reduce cognitive load. They deviate from the heuristic strategy only if forced to do more elaborative processing by the needs of a complex task. Gender differences in information processing could impact both the choices and the applications of debugging strategies male and female end-user programmers pursue.

Although there is no previous research on gender differences in end-user programmers' debugging strategies, there are reports of gender differences in strategies in other domains, such as psychology and education (e.g., [13]), which lends credence to the possibility that such differences might exist in end-user debugging. The closest work to this paper is end-user programming studies on feature usage, which relates to our third reason: these studies consistently show that what males and females *did* regarding feature usage was different [4, 6, 22], but do not consider strategies males and females may have been *trying* to use that led them to employ features differently. The contributions of this paper are in directly investigating the strategies end-user programmers try to use, evaluating the success of these strategies for males and females, and considering design implications of these results to better support end-user debugging.

EXPERIMENT

Participants and Procedures

We brought in 37 female and 24 male undergraduate participants from a variety of majors. Only 6 females and 5 males were engineering, science, or math students; none were computer science students. All had experience working with spreadsheet formulas, but had little or no programming experience. Few background differences existed between genders. Females reported somewhat higher grades than the males (ANOVA: $F(1,59)=3.81$, $p<.06$; males: 3.32 (0.41) females: 3.51 (0.32)), but, as in prior software studies (e.g., [4]), males had significantly higher self-efficacy scores than the females (ANOVA: $F(1,59)=6.30$, $p<.02$; males: 40.96 (4.87) females: 37.73 (4.93)).

After filling out a pre-session questionnaire, the participants took a 25-minute hands-on tutorial (described below), and then debugged two spreadsheets. Their actions and the system's feedback were captured in electronic logs. A post-session questionnaire asked the participants to describe the strategies they used for finding and fixing errors.

Triangulation Procedures

As we have mentioned, strategy exists only in the minds of the participants. Hence, the only direct source of data about strategy is to ask the participant, which we did via the post-session questionnaire. Since self-reported data are not always reliable, it was critical to use as much triangulation as possible, through multiple sources of data and multiple methods of analysis, to verify them.

There were four types of data related to strategy. First were the participants' own descriptions of their strategies, coded using content analysis. Second, were electronic logs of behaviors, which could show evidence of the strategies. Third, playbacks of the logs allowed us to detect more complex behavioral patterns related to claimed strategies than we might have spotted with statistics and scripts alone. Finally, a data mining study was conducted independently in parallel [12]. The data used in the data mining study were logs of participants in [5], in which there were no significant gender differences in participants' success. The goal of triangulating with the data mining effort was to see if *independent* evidence detected *automatically* would corroborate results found by human researchers using more traditional qualitative and quantitative methods.

Software Environment

The study used a research spreadsheet environment that includes WYSIWYT ("What You See Is What You Test") testing features [8] to help end users test and debug spreadsheet formulas. ("Testing" refers to the process of executing a program with different values to find errors.) Although

the WYSIWYT features are intended to support testing-based strategies, they were flexible enough to allow participants considerable leeway in the strategies they actually used. We chose this research spreadsheet system because its features provide participants more choice of testing and debugging strategies than Excel. This environment also had a logging capability to collect the extensive activity data necessary for statistical analysis of behavior patterns.

With WYSIWYT, if the user notices that a cell's value is correct, he or she can check it off, as in Figure 1. Border colors then change, reflecting the portion of cells' formula subexpressions that have been covered by the checked-off values: borders of untested cells are red (light gray in this paper), partially tested cells are shades of purple (intermediate shades of gray), and fully tested cells are blue (black). Optional dataflow arrows also use this coloring scheme. For example, if a user checks off MinMidtrm1Midtrm2 in Figure 1, the system updates all affected cell border colors that fed into the answer of MinMidtrm1Midtrm2, the colors of any visible dataflow arrows, and a "tested %" progress bar (top of Figure 1), all reflecting the formula expressions covered by the testing so far. If a user instead notices that a value is wrong, the user can "X it out," causing the system to highlight suspect cells, as in Figure 1.

Sometimes it is not easy to conjure up useful test values. In that case, the user can press a "Help Me Test" button (not shown), which suggests values that would cover as-yet untested formula subexpressions [8].

When the user displays a formula (lower right of Figure 1), it stays displayed until closed. This device allowed participants to have multiple formulas open at once, increasing the viability of debugging strategies based on code inspection. These and all features in the environment were supported with tool tips (shaded text box in Figure 1).

Tutorial

To avoid suggesting strategies to our participants that might introduce bias, the tutorial covered features only. It did not emphasize any particular feature over another, nor did it present any problem-solving scenario that might suggest how to build a strategy using the feature. We explained the features and gave participants hands-on practice time. The tutorial covered Tool tips, Checkmarks, X-Marks, Arrows, Formula Edits, and Help Me Test. A "quick reference" handout describing the features was also provided.

Half of the tutorial sessions were presented by a male and half by a female. This design ensured that approximately 50% of participants were instructed by a same-gender instructor and 50% by an opposite-gender instructor, serving to distribute any gender effect of the tutorial presenter equally over the two genders.

Spreadsheets and Materials

All participants were asked to test two spreadsheets, Gradebook (Figures 1 and 2) and Payroll. Gradebook was seeded with six bugs and Payroll with five. Other than the layout of the cells, the spreadsheets and the seeded bugs replicated those of [4]. When designing the layout, we took care to

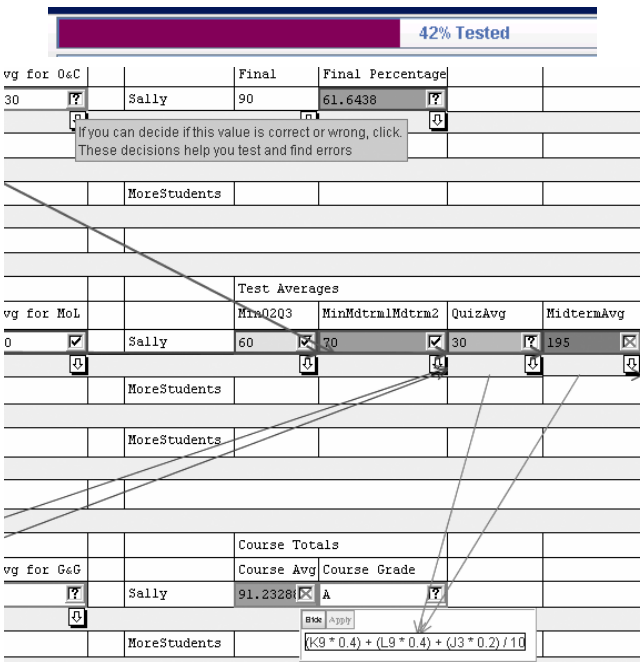

Figure 1: (Top) Tested % progress bar. (Bottom) A portion of the Gradebook spreadsheet. The user noticed an incorrect value in Course_Avg and placed an X-mark. As a result of this X and other X's and √s, six cells were highlighted as possible sources of error, with some more likely than others.

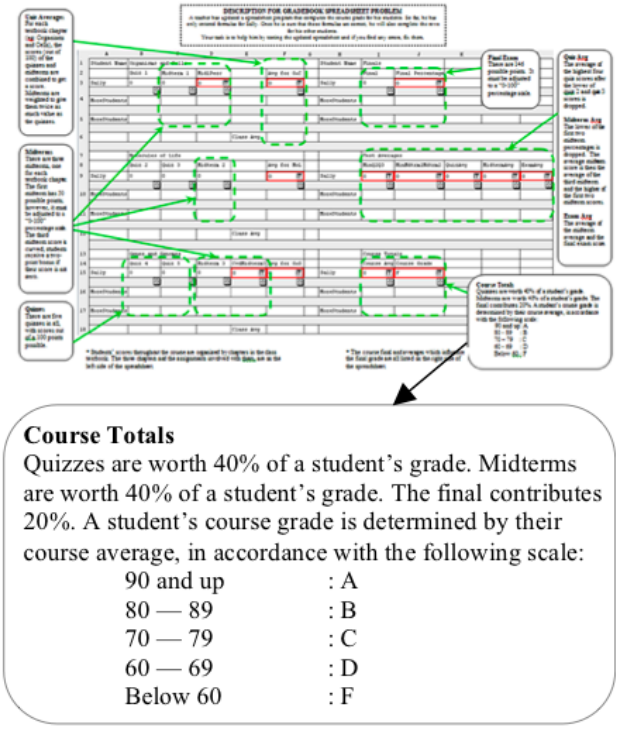

Figure 2: The description handout explained the intent of the groups of cells on the spreadsheet. (Miniaturized for space reasons, with one of the callouts blown up for readability.)

avoid potential confounds among different sequences participants might follow. For example, Western reading order was distinguishable from the order in which our description handouts introduced the cells and from the order in which the data flow through the spreadsheet.

The participants were given the spreadsheet, a handout describing it (Figure 2), and a handout with examples of two sets of correct values. The order in which the spreadsheet problems and handouts were presented to the participants was random across sessions, to avoid the order influencing participants' strategy choices. Time limits of 22 minutes (Gradebook) and 35 minutes (Payroll) were set, in order to simulate the presence of time constraints in real-world computing tasks and to prevent experimental confounds. The participants were told that a spreadsheet had been updated and that, "Your task is to test the updated spreadsheet and if you find any errors, fix them."

SUCCESS BY GENDER

We begin with the "bottom line" by gender: who had the most success debugging? We used changing a faulty formula correctly as our indicator of a bug fixed. In this study, males were significantly more successful at fixing bugs (ANOVA: $F(1,59)=12.20$, $p<.001$; males: 6.71 (2.46) females: 4.24 (2.83)).

When analyzing which strategies each gender *tried* to use, we considered our entire population. However, when analyzing strategy ties to bug fixing performance, we needed to avoid potential confounds of the females' lower performance affecting other statistical inferences. Since our data did not meet the requirements for ANCOVA, we instead tested each gender's correlations with success separately. We also compared successful males with successful females, where "successful" meant having fixed above the median number of bugs (11 in total), and "unsuccessful" otherwise. Of the 37 females, 14 were successful and 23 unsuccessful; of the 24 males, 16 were successful and 8 were not.

Our statistical methodology required an array of tests, which were used according to the following policy. To test for ties between numeric independent variables and numeric outcomes, we used Pearson's correlation test. To test group differences in numeric outcomes, we used ANOVA. To test groups' ranked outcomes or when the data were seriously skewed, we used the non-parametric Wilcoxon test. To test groups' binary outcomes (yes/no), we used the non-parametric Fisher's Exact Test, which is appropriate for such tests on sample sizes from very small to moderate.

THE STRATEGIES

The concept of strategy includes the notion of mental planning with intent. Since it is not possible to observe this directly, we asked participants to report the strategies they used on the post-session questionnaire. We then triangulated these results by looking for evidence in participants' observable actions that bore out the reported strategies.

The concepts we found in participants' responses to the post-session question produced the set of codes in Table 1. The concepts were coded simply as present or absent from their responses, except for testing. Since some end-user programmers use the word "testing" imprecisely, using it to describe anything from testing to code inspection, we could not give a great deal of credence to use of the word "testing," and this prevented presence/absence coding. Therefore, we ranked participants based on the "level" of testing concepts they reported. We identified four levels of testing in their responses: (1) talking about values or mentioning testing, (2) mentioning multiple values for double-checking results, (3) stating need for covering situations in a cell (concept of local coverage) or adding in their own test cas-

Strategy Code Definitions	Example Participant Statements
Dataflow: Following formula dependencies.	"Systematically go from the formulas that rely wholly on data cells, progressing to formulas that rely on other formula driven cells."
Testing: Trying out different values to evaluate the resulting values.	"After correcting everything I pressed the 'Help Me Test' button and double-checked the values to make sure they worked. Next I plugged in values from the given correct example pages to make sure they worked well."
Code Inspection: Examining formulas to determine their correctness.	"I then looked at the formulas to see if I could find mistakes."
Specification Checking: Comparing the description of what the spreadsheet should do with the spreadsheet's formulas.	"Going for the information given on the handouts and making sure each formula fit that description…"
Color Following: Using border colors to guide their efforts.	"Remove as many red borders as possible, then purple, until all are blue."
To-do Listing: Checking off formulas to denote that they have been examined.	"... find errors using the checks and X's to show me what I'd already checked."
Fixing Formulas: Explicitly described strategy in terms of editing formulas to fix them.	"...tried to make sense of the formulas and change them as need[ed] the best I could..."
Spatial: Following the layout of the spreadsheet.	"Usually top to bottom, left to right..."

Table 1: Participants' responses when asked post-session to describe their strategies in finding and fixing the errors.

es, and (4) covering different situations with an eye to globally covering all the situations in the spreadsheet. For example, the testing entry in Table 1 was coded as level 2.

The codes do not necessarily constitute strategies of which software engineers would approve, but we did not filter — if a participant said it was their strategy, we took their word for it. Because of this, codes are not necessarily orthogonal; if participants introduced a concept, even if it was related to another concept, we added a code for it. Two coders independently coded portions of the participants' responses and compared them, developing the codes further and iterating until an acceptable level of agreement was achieved. At the point the agreement rate reached 84%, demonstrating a reasonably robust set of codes, a single researcher coded all remaining responses. The results are shown in Table 2. Most participants (58/61) mentioned multiple strategies, in which case we assigned multiple codes.

The most common co-occurrences were testing and code inspection. They are complementary, and about three-fourths of participants who used one also used the other (Table 2). For example, one participant said that she *"first input some simple example values to start with. Then I just go from one step to the next making sure that formulas are correct and the values make sense, and checking them off..."* Replaying her activity log demonstrated exactly that, showing that she tried out numerous values that she conjured up herself, not just the ones from the handout, interspersed with examining the formulas. For example, in her first few actions she examined a formula, and immediately after, changed an input value impacting that formula, then checked off that cell when she saw its resulting value. This way of combining testing and code inspection led her to edit a formula that did indeed have a bug in it.

Four strategies stood out in what the participants said, either because of the frequency of mention of these strategies or because of significant gender differences in what they said (presented in upcoming sections). The four were dataflow, testing, code inspection, and specification checking. The remaining strategies were often adjuncts to the four major strategies—either facilitating the execution of another strategy or combining with another strategy—so we present them in the context of related major strategies.

	Dflo	Test	Code	Spec	Colr	Todo	Fix	Spat
Dflo	10	6	5	1	5	0	3	2
Test		46	34	22	5	10	10	3
Code			42	23	4	10	9	3
Spec				30	4	7	11	1
Colr					10	1	2	2
Todo						12	1	0
Fix							18	0
Spat								4

Table 2: Diagonal cells (highlighted) show the number of participants mentioning each strategy; remaining cells show co-occurring mentions.

STRATEGY USAGE TIES TO SUCCESS

Dataflow

In programming environments, dataflow is a widely supported debugging strategy: ever since Weiser's classic study identified slicing as an important strategy for debugging [25], numerous tools have been based on slicing. In spreadsheets, slicing amounts to following dataflow, i.e., cell references. The references form a dataflow chain, where the chain starts at an input value and ends at a final calculated output value.

Ten participants reported their strategy to be dataflow: half of them were successful males but none were successful females (Fisher's Exact: $p<.05$; 5/16 successful males and 0/14 successful females).

To triangulate against what the participants said, we considered their behaviors recorded in the logs. While dataflow order could be either depth-first or breadth-first, we found no clear evidence of breadth-first. Thus, we henceforth use the term "dataflow" to refer to depth-first dataflow.

We measured dataflow behavior in two different ways: the participants' number of dataflow instances (sets of multiple consecutive visits within a single chain, reported here) and the number of cells visited consecutively in one chain (not reported here, but consistent with the instance results). The mean (SD) instances for males was 303.71 (120.49), and for females 270.27 (88.02). For males, dataflow and bugs fixed significantly correlated with each other (Pearson: $r(22)=.42$, $p<.04$). For females the same relationship did not hold (Pearson: $r(35)=.20$, $p<.27$). Figure 3 shows these data.

Considering dataflow over time makes clear that, from the very beginning, males were more dataflow-oriented than females. In their first task (recall that task order was counterbalanced), males followed dataflow significantly more than females (ANOVA: $F(1,59)=4.39$, $p<.04$; males: 84.29 (71.82), females: 64.27 (40.92)). Although in Task 1, males' dataflow instances did not have a significant association to bugs fixed, by Task 2, this association had reached significance (Pearson: $r(22)=.42$, $p<.04$); but not for females (Pearson: $r(35)=.14$, $p<.36$). The mean (SD) for dataflow instances in Task 2 for males was 161.25 (62.22), and for females: 148.97 (63.14).

These behavioral patterns correspond to the successful males' discussion of strategies. For example, one successful male described his dataflow strategy as: "*...try to follow the*

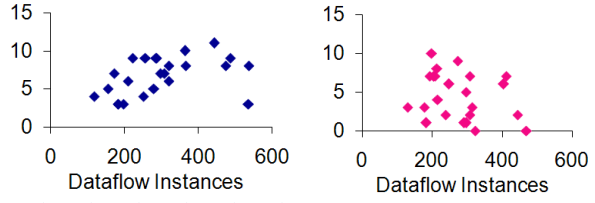

Figure 3: Correlation between total bugs fixed and number of dataflow instances. Left: male (significant), right: female.

flow of information..." Replaying his actions showed that, after a period of initial exploration, he began focusing on one dataflow chain at a time. For example, in Payroll, he began with a combination of ad hoc code inspection and testing. About 5 minutes into the task, he turned on dataflow arrows, and immediately followed one of the arrows upstream in the dataflow chain (i.e., toward the initial input). He continued to use arrows throughout the session, following them upstream within dataflow chains. Whenever he left a dataflow chain, he switched to a new dataflow chain, one chain at a time.

Were dataflow strategies related to arrows, the primary dataflow feature in the environment? For males, yes. Males' total usage of arrows correlated to dataflow usage across both tasks (Pearson: $r(22)=.52$, $p<.008$). But for females, this was not the case (Pearson: $r(35)=.10$, $p<.49$). Perhaps because of the males' association of arrows with dataflow, males used arrows almost twice as often as females as well (males: 27.52 (30.99); females 12.73 (13.79)).

As the above statistics make abundantly clear, dataflow strategies worked much better for males than for females. One explanation may be the comprehensive processing preference of females described by the Selectivity Hypothesis [18, 20]. The dataflow strategy is not comprehensive; it focuses on a single path, ignoring the other cells, which is akin to the cue-oriented strategy that this theory associates with males.

Testing

Testing, in software engineering literature, is executing a program with different values, with the intent of finding errors. Hence, testing in this environment can be defined as modifying the values of the spreadsheet to see the answers produced.

In contrast to participants' mentions of dataflow, which was a trait of successful males, there were no significant gender differences between what successful males and successful females *said* about using testing as their strategy. The lack of gender difference may be a ceiling effect due to the very high frequency with which this strategy was mentioned, especially by successful participants. In fact, successful participants of both genders talked of more testing concepts than unsuccessful participants at a marginally significant level (Wilcoxon: $p<.08$). (We used Wilcoxon here because, as described earlier, our scoring scheme for statements about testing was based on levels/ranks.)

Some participants may have been encouraged toward testing because their (reported) strategy was to follow the "testedness" colors; these were the *color followers.* Ten participants described their strategies in terms of these colors. In fact, qualitative analysis of the log files showed that 34 participants in Task 1 and 31 participants in Task 2 exhibited color following behaviors. Interestingly, here a gender difference surfaced relevant to testing and success: *none* of the successful females mentioned following colors.

The parallel data mining study (with different participants) confirmed behaviorally the existence of testing as a strategy by end-user programmers. The data mining study searched, without human supervision, for interesting *sequences* of events. One behavioral pattern it identified was sequences of value edits followed by checking off and/or X-ing out values, the primary testing operations in this environment. Successful participants used this pattern significantly more than unsuccessful participants (ANOVA: $F(1,38)=5.71$, $p<.03$; successful: 8.06 (9.92) unsuccessful: 2.38 (4.18)).

In the current experiment's participants' logged behaviors, we counted (1) the number of values participants edited manually, (2) the number of values they edited manually plus uses of the Help Me Test button to generate new values, and (3) percent testedness.

All three measures echoed what the successful *males* said (Table 3). In fact, this strategy differentiated males' success levels, with significant differences in all three measures for successful versus unsuccessful males. However, this was not the case for females. Further, the *amount* of testing, as measured through both value edits and percent tested, significantly correlated to the number of bugs fixed for males, but not for females. Males' value edits: $r(22)=.47$, $p<.03$; females: $r(35)=.17$, $p<.33$. Males' percent tested: $r(22)=.46$, $p<.03$; females: $r(35)=.07$, $p<.68$.

Thus, by every behavioral measure, the testing strategy was strongly tied with males' success, but not with females' success. This could be because the females had another strategy they felt would be even more useful to them, namely code inspection.

Code inspection

Code inspection, in software engineering, means examining the computer source code to uncover errors and defects. In spreadsheets, this translates to examining formulas to judge their correctness. Code inspection in spreadsheets is widely regarded as a necessary part of spreadsheet auditing

Value edits	Value edits + Help Me Test	% Tested
Males: S vs. U: 16 SM, 8 UM		
Succ. males more: SM 59.8 (31.99) UM 26.62 (22.93) **p<.01**:F(1,22)=7.06	Succ. males more: SM 64.37 (29.85) UM 30.87 (22.72) **p<.01**: F(1,22)=7.75	Succ. males more: SM 0.796(0.141) UM 0.669(0.173) **p<.04**: F(1,22)=5.06
Females: S vs. U: 14 SF, 23 UF		
Not significant: SF 36.36 (19.34) UF 28.78 (12.28) p<.15:F(1,35)=2.13	Not significant: SF 39.21 (19.16) UF 34.39 (14.52) p<.39: F(1,35)=0.75	Not significant: SF 0.662 (0.1820) UF 0.669 (0.0940) p< .88: F(1,35)=0.22
Successful: M vs. F: 16 SM, 14 SF		
Succ. males more: SM 59.68 (31.99) SF 36.36 (19.34) **p<.02**:F(1,28)=5.88	Succ. males more: SM 64.37 (29.85) SF 39.21 (19.16) **p<.01**: F(1,28)=7.29	Succ. males more: SM 0.796 (0.141) SF 0.662 (0.182) **p<.03**: F(1,28)=5.08

Table 3: Males' and females' use of testing as it related to success. M=Males, F=Females, S=Successful, and U=Unsuccessful.

processes [21], such as the one used at the U.K. Department of Revenue and Customs [2]. The WYSIWYT environment has the unusual (for spreadsheets) ability to display multiple formulas in the same view as values, which allowed code inspection to co-occur with testing (Figure 4).

Just plain code inspection

In the questionnaire data, successful females were significantly more likely to mention code inspection as their debugging strategy than unsuccessful females (Fisher's Exact: $p<.03$; 13/14 successful females and 13/23 unsuccessful females), whereas there were no significant differences between the reports of successful males and unsuccessful males, or between successful males and successful females.

There is triangulating evidence from the parallel data mining study that code inspection exists as a behavior by end-user programmers. The machine's unsupervised mining of the data identified a behavioral pattern of consecutive formula display/undisplay sequences containing only a few formula edits, demonstrating those participants' interest in *reading* formulas without editing most of them.

To learn the extent to which our successful participants relied on code inspection behaviors, we replayed logs of the Gradebook session and noted each instance of code inspection. We chose Gradebook because its layout was more spread out than Payroll, reducing the chance of spatial overlaps dissuading participants inclined toward code inspection. Because the qualitative mechanism was time-consuming, we restricted our replays to the 30 successful participants, i.e., those who fixed above the median number of bugs in Gradebook (16 males and 14 females).

After several viewings of the logs, we converged on a definition of an instance of code inspection: an instance began when a participant (1) had two or more formulas displayed simultaneously solely for reading (i.e., no editing was being done), or (2) displayed multiple formulas consecutively in rapid succession with no intervening actions. The instance ended when a participant hid all formulas or when a formula or value was edited. We counted the number of instances and the number of formulas displayed per instance.

As Table 4 shows, by either measure, successful females used code inspection significantly more than successful males.

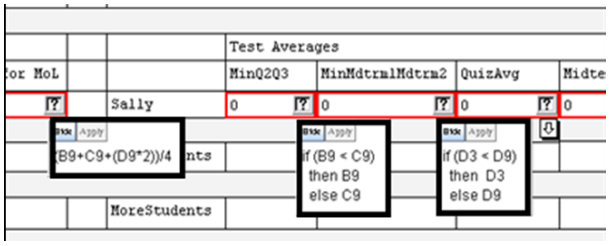

Figure 4: A participant's session during code inspection. We have superimposed dark rectangles on the screenshot to highlight the formulas being displayed.

To-do listing: a variant of code inspection

To-do listing is a variant of code inspection in which participants used checkmarks to check off formulas they had approved or fixed, and used X-marks to form a to-do list of formulas still needing work.

Twelve participants described to-do listing as their strategy. The parallel data mining study also automatically picked out pattern sequences of to-do listing's components, namely sequences of posting and hiding several formulas consecutively, followed ultimately by checking one off.

Neither our questionnaire data nor the parallel data mining study showed significant ties between to-do listing and any particular groups. However, turning to the behavior logs, we were able to analyze "early" to-do listing behaviors for ties to groups. To accomplish to-do listing, participants co-opted checkmarks and X-marks, which are for testing. This introduced difficulty in distinguishing testing from to-do listing. To avoid this problem, we focused exclusively on early to-do listing, before useful test values were present. We counted checkmarks or X-marks *placed on formulas the participant had viewed*, provided that all the values were still 0 (the initial values). Early to-do listing was thus a very conservative measure of to-do listing.

Using the above measure, in the participants' first task there were no gender differences in to-do listing, but by the second task, females did significantly more to-do listing (Wilcoxon, $p<.02$). (Because many participants did not place checkmarks and X-marks before any input values were changed, the number of zero counts skewed these particular data far from normal; thus we used solely non-parametric tests in this analysis.) Counts of participants confirmed this phenomenon, with significantly more females involved in early to-do listing than males (Fisher's Exact: First task: $p<.78$, 13/37 females and 7/24 males; Second task: $p<.03$, 13/37 females and 2/24 males).

To-do listing is simply code inspection plus the ability to track code inspection status. In independent interviews conducted in 2006 by the spreadsheet analysis company i5Logic, all six professional auditors interviewed requested the equivalent of to-do listing (personal communication Aug. 29, 2007 with Matthew Johnen, CEO of i5Logic). Since a significant number of participants, especially females, tried to engage in to-do listing, this seems to be an opportunity area for end-user debugging tools.

	Number of code inspection instances	Total displayed formulas in all instances
16 SM, 14 SF	SF>SM $F(1,28) = 4.99$ **$p < .04$**	SF>SM $F(1,28) = 5.54$ **$p < .03$**

Table 4: Code inspection behavior results (SF=Successful females, SM=Successful males).

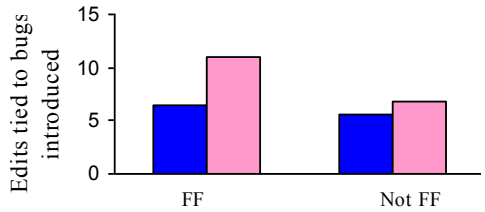

Figure 5: Mean number of edits to cells where participants introduced new bugs. By Formula Fixers (left) and the other participants (right). Dark: male, Light: female. All males mean (SD): 5.83 (3.68); all females: 7.92 (5.04).

Fixing formulas: code inspection gone wrong

While it is debatable whether *fixing formulas* is a bona fide strategy, when participants said editing formulas was their strategy, we took them at their word. What differentiates formula fixing from code inspection is that code inspectors spoke of *evaluating* the validity of code, whereas formula fixers spoke explicitly only of *changing* code.

Fixing formulas was not a good strategy, and females were particularly subject to this pitfall. Most of the formula fixers were unsuccessful females: 8/10 females who mentioned this strategy were unsuccessful compared to only 3/8 males. For females, unlike males, mentioning fixing formulas as a strategy correlated to bugs introduced (Pearson: $r(35)=.37$, $p<.03$); see Figure 5. The implication is that most participants who mentioned this spent their time in an ad hoc manner, editing formulas, without differentiating buggy formulas from others.

Code inspection and theory

One explanation of why females used more code inspection could be the known gender differences in perception of risk. Blackwell's Attention Investment Model [7] explains the role of perception of risk in problem solving. According to the model, a user weighs several factors before taking action: perceived benefits and pay-off, perceived cost, and perceived risks. When the user has choices of actions, high perceived risk and cost of an action that are not offset by the perceived benefits and pay-offs are likely to result in a decision not to pursue the action. Research has shown that females perceive higher risk than males in a wide variety of situations [9], including intellectual risk taking. Inspecting formulas could be perceived as low-risk because it does not require using unusual features; other studies have reported a disinclination by females to use unusual features [4, 6, 22].

Focusing now on the successful females, why did they use more code inspection than successful males? We believe at least part of the explanation comes back to the Selectivity Hypothesis. Recall its explanation of females' apparent lack of interest in dataflow strategies: females' tendency toward detailed, elaborative information processing strategies were the opposite of dataflow's cue-following approach. Yet, elaborative processing is consistent with code inspection, which is by definition examining many formulas.

Specification Checking

Specification checking refers to participants' inclination to look for errors in formulas by consulting the spreadsheet specifications, i.e., the description handouts (Figure 2). While 30 participants reported specification checking, there were no significant differences in reports of specification checking between the successful males and females. To determine how to verify specification checking in the logs, we turned to the log of a female participant who described checking specifications on her written questionnaire.

In the Gradebook task, she began by visiting the cells described by the first callout in the printed description (Figure 2). We defined a "visit" to a cell as any physical touch of that cell, such as checking off its value, opening its formula, etc. After several minutes, she moved to two cells that calculate the "Final Exam" score which, in Western reading order of the description (left to right, gradually downward), are explained together in the second callout. About 8 minutes into the task, she moved to the cells described by the third callout. This callout describes cells that come *after* the next ones in Western reading order in the spreadsheet itself, clearly showing adherence to the description's order, rather than to the order of the spatial layout of the spreadsheet.

We chose such sequences as a conservative indicator of specification checking behavior. Specifically, we counted the number of times a participant visited a cell that was either in the same block of cells (i.e., those described by a single callout) as the previously visited cell or in the following block, according to the callouts. A participant's score was the maximum number of description order visits in a row as a measure of their dedication to this strategy. We note that, while a few participants said they used a *spatial strategy* (e.g., following the layout of the spreadsheet using Western reading order), in our experimental design, we had ensured that other orders—especially cell layout order—would not be the same as the description callout order in either Western reading order or column order (top to bottom, gradually rightward). Thus, these orders were not confounded with the specification checking indicator.

Using this measure, females' specification checking score positively correlated to bugs fixed (Pearson: $r(35)=.33$, $p<.04$), but males' use of the strategy had no such correlation. See Figure 6. Although it may appear untidy, the result was resilient to outliers; in fact, removing outliers identified via studentized residuals produced $p<.02$. Interestingly, although use of this strategy correlated to *success* for fe-

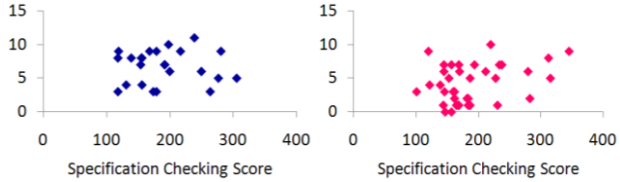

Figure 6: Specification checking's ties to the number of bugs fixed. Left: Male, Right: Female (significant).

males and not males, differences in total *usage* of this strategy by males and females were not significant (males: 190.08 (54.77), females: 189.73 (57.59); successful males: 184.81 (46.09), successful females: 210.21 (67.00)).

What made specification checking a successful strategy for females, if it was not the difference in how much it was used? As in code inspection, the Selectivity Hypothesis gives a possible explanation. Specification checking can provide a big picture of how the spreadsheet's cells should interact, aligning well with the comprehensive processing strategy. Another possible explanation may be that verbal-oriented users were attracted to reading these descriptions of functionality. Although there is controversy about gender differences in verbal ability, some studies have reported females to exhibit higher verbal performance (e.g., [14]).

DISCUSSION AND IMPLICATIONS

We summarize the answers to our research questions as follows:

RQ1: Debugging strategies end-user programmers tried to use: Participants described eight strategies: dataflow, testing, code inspection, specification checking, color following, to-do listing, fixing formulas, and spatial.

RQ2: Gender differences in debugging strategies and *RQ3: Ties to success:* There was significant evidence that males and females tried to engage in different strategies. Males engaged in dataflow and testing to much greater extents than females did. More to the point, these two strategies were successful primarily for males, but not for females. Instead, the females' major successful strategies were code inspection and specification checking. This difference may explain why one feature-oriented study of end-user debugging found that males used testing features more, and females worked on formulas more [4].

In support of the males' strategies, like many other end-user environments (e.g., the Why Line [16], UCheck [1], and Woodstein [24]), WYSIWYT supports dataflow. It goes further than other end-user environments in supporting testing, but many end-user environments, including virtually all spreadsheets and programming-by-demonstration systems, put at least some emphasis on evaluating the program through its outputs.

Implications for design are that dataflow and testing represent important strategies for male end-user debuggers' success, and should be supported in end-user programming environments. Examples of supporting devices from our experiment included colors to guide testing, dataflow arrows, Help Me Test, and providing example values. Environments with limited support for dataflow and testing may benefit by increasing their support for systematic use of these strategies.

Regarding females, the main strategies with which they succeeded, code inspection and specification checking, are well respected in both the software engineering [3] and the spreadsheet auditing communities [21]. Yet, a recurring theme in our results is that these strategies by females are mismatched with the features available, in this environment and in most other end-user programming environments.

For example, code inspection, possible but a little costly in our environment, is not viable in many environments. Most spreadsheet systems do not allow displaying multiple formulas without tedious mode switching. Some programming-by-demonstration systems provide no view of the program at all. To-do listing, i.e., tracked code inspection, was not supported. When to-do listers cleverly repurposed the testing features to track the status of formulas, the resulting colors were misleading, since the system was reasoning about values it thought the user had approved, not about the formulas they had actually inspected. Specification checking was supported through a handout only, not through features in the environment; it is likewise absent from other end-user programming environments.

Implications for design following from our code inspection and specification checking results are that these are important strategies for female end-user debuggers' success, and should be supported in end-user programming environments. One possible way to do so was suggested by the females' behaviors: they used checkmarks and X-marks to track code inspection. If specifications could also be supported in end-user programming environments, these tracking devices might also be tied to the specifications.

And what of the large number of unsuccessful females? One possibility is that these females did poorly in debugging simply because they used poor strategies. A case in point was the fixing formulas strategy, an unfortunate variant of code inspection. But this "bad" strategy may be transitioned into a better one if appropriate support were available. Guiding users toward a particular strategy in the context of real work may be possible using the principles of minimalist learning [10]. One direction that shows promise uses minimalist learning layered with short "how to" videos of debugging strategy tips [23].

Finally, note that no single female is likely to have every trait statistically associated with females, nor is any single male likely to have every trait statistically associated with males. Thus, we do not propose gendered software such as "spreadsheets for females." Rather, we hope software developers will find ways to support the strategy needs we have reported so as not to penalize anyone, male or female, regardless of their strategy preferences.

CONCLUSION

This paper reports our investigation into strategies end-user programmers try to use when debugging. Because strategies exist in the head, we triangulated extensively. Thus, we triangulated what our participants said with replays of what they did, logs of what they did, and logs of what an independent set of participants did. Our analysis used qualitative, quantitative, and unsupervised data mining methods.

Of the eight strategies we identified, there was ample triangulated evidence of seven of them (all but spatial), strongly indicating that these strategies are real—and each of these seven strategies also showed significant gender differences.

Especially important, the debugging strategies that worked well for males were not the ones that worked well for females. Dataflow and testing (and its cousin, color following) worked well for males, and code inspection and specification checking worked well for females. Finally, the unsuccessful fixing formulas strategy was a pitfall to which females were more prone than males.

Female participants attempted a number of strategies that are reasonable, but are almost entirely unsupported in end-user programming environments. Males' strategies, on the other hand, are supported in many environments, to at least some extent. These results reveal several opportunities for end-user debugging tools to better support end-user programmers, both male and female.

ACKNOWLEDGMENTS

This work was supported in part by Microsoft Research and by NSF CNS-0420533, ITR-0325273 and CCR-0324844.

REFERENCES

1. Abraham, R., Erwig, M. UCheck: A spreadsheet unit checker for end users, *J. Vis. Langs. Comput. 18*, 1 (2007), 71-95.
2. Anonymous, H. M. Customs and Excise Computer Audit Service, Methodology for the Audit of Spreadsheet Models, 2001. http://customs.hmrc.gov.uk/channelsPortalWebApp/channelsPortalWebApp.portal?_nfpb=true&_pageLabel=pageVAT_ShowContent&id=HMCE_PROD_009443&propertyType=document (downloaded Aug. 28, 2007).
3. Basili, V., Selby, R. Comparing the effectiveness of software testing strategies, *IEEE Trans. Soft. Eng. 13*, 12 (1987) 1278-1296.
4. Beckwith, L. Burnett, M., Wiedenbeck, S., Cook, C., Sorte, S., and Hastings, M. Effectiveness of end-user debugging software features: Are there gender issues? In *Proc. CHI 2005*, ACM Press (2005), 869-878.
5. Beckwith, L. Kissinger, C., Burnett, M., Wiedenbeck, S., Lawrance, J., Blackwell, A., and Cook, C. Tinkering and gender in end-user programmers' debugging, In *Proc. CHI 2006*, ACM Press (2006), 231-240.
6. Beckwith, L., Inman, D., Rector, K., and Burnett, M. On to the real world: Gender and self-efficacy in Excel, In *Proc. VLHCC,* IEEE (2007).
7. Blackwell, A. First steps in programming: a rationale for attention investment models. In *Proc. VLHCC,* IEEE (2002), 2-10.
8. Burnett, M., Cook, C., and Rothermel G. End-user software engineering. *Comm. ACM 47*, 9 (2004), 53-58.
9. Byrnes, J., Miller, C., and Schafer D. Gender differences in risk taking: A meta-analysis. *Psych. Bulletin 125* (1999), 367-383.
10. Carroll, J. (Ed.), *Minimalism Beyond "The Nurnberg Funnel"*, MIT Press, Cambridge, MA, 1998.
11. Cross, N. Expertise in design: An overview. *Design Studies* 25, 5 (2004), 427-441.
12. Fern, X., Komireddy, C., Burnett, M. Mining interpretable human strategies: A case study, In *Proc. ICDM,* IEEE (2007).
13. Gallagher A., De Lisi R., Holst P., McGillicuddy-De Lisi A., Morely M., Cahalan C. Gender differences in advanced mathematical problem solving, *J. Experimental Child Psychology 75*, 3 (2000), 165-190.
14. Halpern, D. *Sex Differences in Cognitive Abilities, 3rd Edition*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 2000.
15. 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.
16. Ko, A. and Myers, B. Designing the Whyline: A debugging interface for asking questions about program failures. In *Proc. CHI 2004*, ACM Press (2004), 151–158.
17. Littman, D., Pinto, J., Letovsky, S., and Soloway, E. Mental models and software maintenance. In E. Soloway and S. Iyengar (Eds), In *Proc. ESP*. Ablex, Norwood, NJ (1986), 80-98.
18. Meyers-Levy, J. Gender differences in information processing: A selectivity interpretation. In P. Cafferata & A. Tybout, (Eds) *Cognitive and Affective Responses to Advertising.* Lexington, Ma, Lexington Books, 1989.
19. Nanja, N. and Cook, C. An analysis of the on-line debugging process. In G. M. Olson, S. Sheppard, and E. Soloway (Eds.), In *Proc. ESP*. Ablex, Norwood, NJ, 1987.
20. O'Donnell, E. and Johnson, E. The effects of auditor gender and task complexity on information processing efficiency. *Int. J. Auditing 5* (2001), 91-105.
21. Powell, S., Baker, K., Lawson, B. An Auditing Protocol for Spreadsheet Models, Jan. 2007. http://mba.tuck.dartmouth.edu/spreadsheet/product_pubs.html (downloaded Aug. 28, 2007).
22. Rosson, M., Sinha, H., Bhattacharya, M., Zhao, D. Design planning in end-user web development, In *Proc. VLHCC*, IEEE (2007).
23. Subrahmaniyan N., Kissinger, C., Rector, K., Inman, D., Kaplan, J., Beckwith, L., Burnett, M., Explaining debugging strategies to end-user programmers, In *Proc. VLHCC*, IEEE (2007).
24. Wagner, E. and Lieberman, H. Supporting user hypotheses in problem diagnosis on the web and elsewhere. In *Proc. IUI*, ACM Press (2004), 30–37.
25. Weiser, M. Programmers use slices when debugging, *Comm. ACM 25*, 7 (1982), 446-452.