Gender Differences in End-User Debugging, Revisited: What the Miners Found

Valentina Grigoreanu*, Laura Beckwith*, Xiaoli Fern*, Sherry Yang†, Chaitanya Komireddy*, Vaishnavi Narayanan*, Curtis Cook*, Margaret Burnett*

*Oregon State University
Corvallis, Oregon
{grigorev,beckwith,xfern,komirech,narayava,cook,burnett}@eecs.orst.edu

†Oregon Institute of Technology
Klamath Falls, Oregon
Sherry.Yang@oit.edu

Abstract

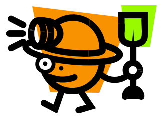
We have been working to uncover gender differences in the ways males and females problem solve in end-user programming situations, and have discovered differences in males' versus females' use of several debugging features. Still, because this line of investigation is new, knowing exactly what to look for is difficult and important information could escape our notice. We therefore decided to bring data mining techniques to bear on our data, with two aims: primarily, to expand what is known about how males versus females make use of end-user debugging features, and secondarily, to find out whether data mining could bring new understanding to this research, given that we had already studied the data manually using qualitative and quantitative methods. The results suggested several new hypotheses in how males versus females go about end-user debugging tasks, the factors that play into their choices, and how their choices are associated with success.

1. Introduction

Although there has been a fairly wide interest in gender differences in computing professions and education, as well as in gaming, there has not been much research on how gender differences[1] interact with end users' use of *software features*, a research area we have begun to pursue which we term *gender HCI*. Our particular focus is on questions related to end-user software development. Our goal is to learn how to design end-user programming environments such that they support end-user programmers of both genders.

Most of our work so far in this area has followed a theory-driven approach, in which theories from psychology, education, and HCI have been used to generate hypotheses which have then been investigated via empirical studies. However, a disadvantage in deriving empirical hypotheses from only established theories is that these theories do not take into account the specific needs and issues that arise in end-user programming. Research situations such as this are often referred to as "ill-structured" problems [22]. Such problems contain uncertainty about which concepts, rules, and principles are pertinent to the problem. Further, the "best" solutions to ill-structured problems depend on the priorities underlying the situation. In such problems, in addition to hypothesis testing and application, there is also the need for hypothesis generation. Such problems are candidates for ultimately deriving new theories from data and patterns.

Toward this aim, we have previously used manual qualitative analysis techniques [23], inspecting data on software feature usage in search of useful patterns leading to hypotheses. Although the results of these efforts have been fruitful, still, as humans we are fallible, especially given large amounts of detailed data. We suspected that there may be important information that we were overlooking. Therefore, we employed a methodology change: turning to data mining techniques to find feature usage patterns that we may have missed.



In this paper we report the results of revisiting data we had already analyzed, but this time using a data mining approach. Using this approach, we focus on gender differences in *how* features are used, with the aim of gaining new insights into our previous reports of *when* and *how much.*

Our aim was to derive new hypotheses about females' and males' strategies, adding to the growing foundation for understanding the gender differences in end-user programming situations—by "listening" to the participants, through their data, from the ground up.

[1] While individual differences, such as experience, cognitive style, and spatial ability, are likely to vary more than differences between gender groups, research from several domains has shown gender differences that are relevant to computer usage [4, 10, 16].

2. Background and Related Work

We began our gender HCI research by generating hypotheses [3] from relevant theoretical work from other domains, including self-efficacy theory [2], educational theories such as minimalist learning theory [11], and the model of attention investment [8]. Several of these hypotheses also came from empirical results from others' investigations into gender differences in educational, work, and entertainment settings. We followed up on some of these hypotheses by conducting empirical studies of these gender differences, including both qualitative (e.g., [5]) and quantitative [4, 6] results. Many of our findings have related self-efficacy to the way females interact with software. Self-efficacy is a form of confidence in one's ability, specific to the task at hand [2]. Gender differences regarding computer related confidence have been widely studied, revealing that females (both computer science majors and end users) have lower self-confidence than males in their computer-related abilities [7, 10, 14, 16]. Previous work also found that tinkering with features can be helpful to both females' and males' success, but that males sometimes overdo it, which can interfere with their success [6].

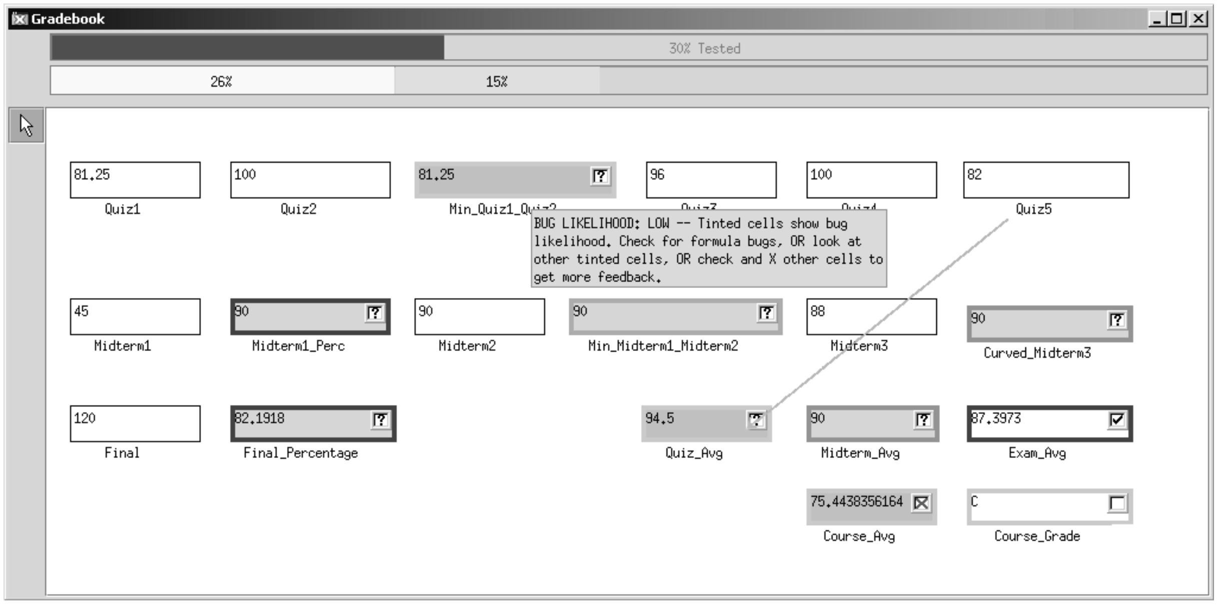
In this paper we apply data mining to uncover additional patterns of interest. Data mining, also known as knowledge discovery in databases (KDD), is typically used to find hidden patterns in data to help understand data and make predictions about future behavior. In this study, we apply sequential pattern mining [1] to our HCI data.

Sequential Pattern Mining was first introduced in the context of retail data analysis for identifying customers' buying habits [1] and finding telecommunication network alarm patterns [15, 17]. It has since been successfully applied to many domains including some HCI related applications, such as web access pattern mining for finding effective logical structure for web spaces [19] and automatic web personalization [18], mining Windows processes data to detect unauthorized computer users [12], and mining user-computer interaction patterns for finding functional usage scenarios of legacy software [13].

Most work on applying data mining techniques to HCI data has focused on finding characteristic patterns of individual users [12, 18] or finding patterns that are common to the entire population [13, 19]. In contrast, in this study we are not interested in these two types of patterns. Instead, our research goal requires us to find patterns that are linked to subgroups of users, i.e., female users and male users. Another somewhat unusual aspect of our work is that we use the found patterns to generate gender-related hypotheses. To our knowledge, data mining has not been used before to generate hypotheses from HCI data. In particular, it has not previously been used to find gender differences in females' and males' interactions with software.

3. The Data and Environment

For this type of purpose, it is acceptable to mine from previous data, so we used data from one of our earlier studies [6]. The data was collected from 39 participants


Gradebook

30% Tested

26% 15%

Quiz1: 81.25	Quiz2: 100	Min_Quiz1_Quiz2: 81.25	Quiz3: 96	Quiz4: 100	Quiz5: 82
Midterm1: 45	Midterm1_Perc: 90	Midterm2: 90	Min_Midterm1_Midterm2: 90	Midterm3: 88	Curved_Midterm3: 90
Final: 120	Final_Percentage: 82.1918		Quiz_Avg: 94.5	Midterm_Avg: 90	Exam_Avg: 87.3973
				Course_Avg: 75.4438356164	Course_Grade: C

BUG LIKELIHOOD: LOW -- Tinted cells show bug likelihood. Check for formula bugs, OR look at other tinted cells, OR check and X other cells to get more feedback.

Figure 1. The user notices an incorrect value in `Course_Avg`—the value is obviously too low—and places an X-mark in the cell. As a result of this X and the checkmark in `Exam_Avg`, eight cells are highlighted as being possible sources of the incorrect value, with the darker shaded cells deemed more likely than others.

who used the "High-Support Environment" in our research prototype. The environment provided a number of features to participants who might feel in need of extra help. These features are detailed in [6]. The participants' task was to find and fix errors in two spreadsheets. Prior to the task, the participants were introduced through a tutorial to the environment features designed to aid in debugging.

The debugging features present were part of WYSIWYT ("What You See Is What You Test"). WYSIWYT is a collection of testing and debugging features that allow users to incrementally "check off" (Checkmark) or "X out" (X-mark) values that are correct or incorrect, respectively [9]. Cells initially have red borders, indicating that they are untested. The more a cell formula's subexpressions are covered by tests (checked-off values), the more blue its border becomes.

The environment also includes arrows, which participants can toggle on and off on a per-cell (or even per-arrow) basis. Arrows serve two purposes: First, they explicitly depict the dataflow dependencies between the cells and, when cells' formulas are open, even between subexpressions in the related cells. Second, arrows' coloring reflect WYSIWYT "testedness" status at a finer level of detail, following the same color scheme as the borders. A user can thus look for red arrows to find cell dependencies that still need to be tested. Figure 1 shows an example of these WYSIWYT features that were available to the participants.

Also present in the environment was the "Help Me Test" (HMT) feature. Sometimes it can be difficult to find test values that will cover the untested logic in a collection of related formulas, and HMT tries to find inputs that will lead to coverage of untested logic in the spreadsheet, upon which users can then make testing decisions.

Each of these features is supported through the Surprise-Explain-Reward strategy [24]. This strategy relies on a user's curiosity about features in the environment. If a user becomes curious about a feature, the user can seek out explanations of the feature via tool tips. The aim of the strategy is that, if the user follows up as advised in the explanation, rewards will ensue.

Participants' actions were recorded in user action log files. A *user action* is defined as a participant's use of a debugging feature. The log files contained detailed information about every user action, including a time stamp for when the action was taken, on which cell it operated, and various related parameters. Here is an excerpt of a log file:

```
15:43:47, Tooltip Showing, CELL31567926-2332 …
15:44:12, Checkmark, CELL31567926-2342 …
15:44:57, Checkmark, CELL31567926-2332 …
```

In addition to the log files, we also collected information on participants' task success, background information, and pre-task self-efficacy. Only the log files were used in pattern mining. The additional information was used to help analyze the patterns we found.

4. The Pattern Mining Process

In this study, we applied sequential pattern mining to our log files to search for potentially interesting patterns.

To formulate the sequential pattern mining problem, we considered each user action as an *event*. Since we are most interested in *how* participants used features, we abstracted away detailed contextual information that distracted from this goal (such as the specific cell on which the features were being used, or the exact time of the actions). This abstraction transformed the data into *sequences* of events. For example, the log excerpt in the previous section translated into the sequence (Tooltip Showing, Checkmark, Checkmark).

4.1 Preprocessing into Debugging Sessions

Following the procedure of [20], we used the notion of *debugging sessions* to break the sequence of events into subsequences. As with Ruthruff et al.'s definition, a debugging session ends with a formula edit (or at the end of the experiment), which presumably represents an attempt to fix a bug. However, unlike Ruthruff et al.'s definition, in which a debugging session began with the placement of an X-mark, our debugging sessions begin as soon as the previous one ended (or at the beginning of the experiment), so that all actions could be considered—not just the subset following an X-mark. In some cases participants edited the same formula multiple times consecutively. Since such edits were obviously a continuation of fixing the same bug, we included them in the preceding debugging session.

Based on this definition, we broke each log file into debugging sessions. The total number of debugging sessions for all 39 participants was 641. Thus the mean per participant was 16.47 debugging sessions. The mean number of events per debugging session was 24.45 events.

4.2 Sequential Pattern Mining

We used the SLPMiner program [21] to search for patterns of the form (A, B, C), where A, B, and C are events that happened in the specified order. A debugging session was considered to contain the pattern (A, B, C) if it had at least one occurrence of events A, B, and C in that order, but the events did not need to be consecutive. We refer to the percentage of all debugging sessions that contained a pattern as the *support* of the pattern.

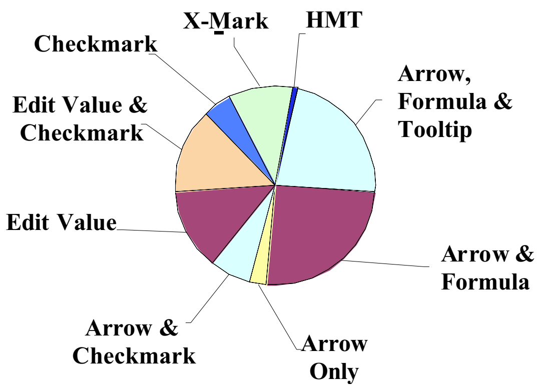


Figure 2: We grouped the 107 patterns into these 9 categories. The categories, based on the patterns' content, are focused on the debugging and other features available in the environment.

SLPMiner searches for all sequential patterns whose support exceeds a pre-specified threshold, and these patterns are referred to as *frequent patterns*. To avoid redundancy due to the fact that any subsequence of a frequent pattern will also be a frequent pattern, the software output the maximal patterns, i.e., patterns that are not subsequences of other frequent patterns. We chose the support threshold to be 10%, i.e., a pattern had to be contained in more than 10% of the 641 debugging sessions to be output by SLPMiner. This threshold was chosen because it allowed us to find patterns that were common to multiple users while still containing some of the interesting but less frequently used features such as X-marks and Arrow operations. We further focused our attention on patterns of limited size, in particular of length between one and four, because without limitations there would simply be too many patterns to process, and longer patterns often contained cyclic behavior and were difficult to interpret.

4.3 Output and Post-processing

From the 641 debugging sessions, SLPMiner found 107 patterns of length one through four. Note that SLPMiner (and other sequential pattern mining algorithms) can only find "frequent" patterns, i.e., those satisfying a minimum support criterion, which was 10% in our case. It was up to us to determine which of the found patterns were really interesting to our research goal.

Toward this aim, for each pattern we computed its occurrence frequency for each user as the percentage of that user's debugging sessions that contained the pattern. For example, if user A had 20 debugging sessions and 10 of them contained pattern p, the occurrence frequency of pattern p for user A was 50%. As a result, we obtained a pattern occurrence frequency table, which provided a comprehensive description of the distribution of the pattern occurrence among all users. We then analyzed these pattern occurrence frequencies in relation to the gender, task performance, and self-efficacy of the participants who used them.

To help analyze the pattern occurrence frequencies in an organized manner and gain high-level understanding of the patterns, we categorized the found patterns such that each category contained patterns centered on a certain set of features. Figure 2 shows how the 107 patterns were distributed into nine non-overlapping categories. See Table 1 for examples of patterns and their categories. Our analysis described in the following sections will be presented based on these categories.

5. Results: How Each Gender Pursued Success

How did the successful versus unsuccessful females and males go about debugging? To investigate this question, we divided the 39 participants (16 males and 23 females) into four groups by gender and number of bugs fixed. We considered a participant "successful" if they fixed at least 7 of the 10 bugs, and "unsuccessful" otherwise. The groups and number of participants are displayed in Table 2.

5.1 Just Like Males: A Female Success Strategy?

Strikingly, in Figure 3 the unsuccessful females and successful males showed a similar frequency profile for

Table 1: Representative patterns output by SLPMiner. Categories were based on the pattern's content.

Category	Example Pattern
Help Me Test (HMT)	(HMT)
Arrow, Formula & Tooltip	(Tooltip Showing, Arrow On, Arrow Off, Edit Formula)
Arrow & Formula	(Arrow Off, Post Formula, Hide Formula, Post Formula)
Arrow Only	(Arrow On, Arrow On)
Arrow & Checkmark	(Hide Formula, Checkmark, Arrow On)
Edit Value	(Edit Value, Edit Value)
Edit Value & Checkmark	(Post Formula, Edit Value, Checkmark, Hide Formula)
Checkmark	(Checkmark, Tooltip Showing, Tooltip Showing, Checkmark)
X-Mark	(Hide Formula, X-Mark, Post Formula, Edit Formula)

Table 2: What: The median number of arrows turned on and off during the experiment by gender and debugging success. Note especially the difference between the successful and unsuccessful males.

Group	Number of participants	Arrows
Successful Females	8	17.5
Unsuccessful Females	15	24
Successful Males	10	12
Unsuccessful Males	6	25.5

each of the five categories on the left half of the graph (from Edit Value to HMT)—all of which are testing-oriented activities. We follow the software engineering definition of "testing" here: judging the correctness of the *values* produced by the program's *execution*.

This suggests that the ways males successfully went about their debugging task are the very ways that did not work out well for the females, leading to the following hypothesis:

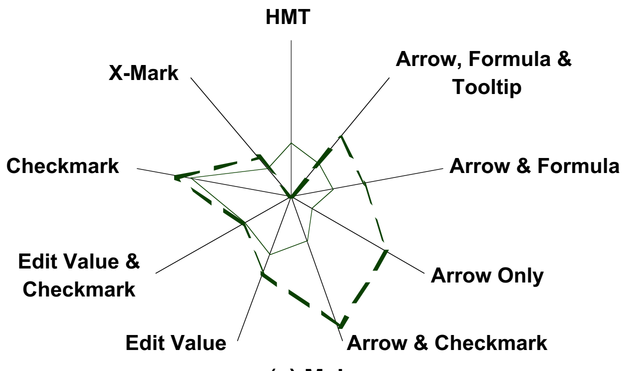
Hypothesis: The debugging and testing strategies that help with males' success are not the right ones for females' success.

5.2 Unsuccessful Males Like Arrows

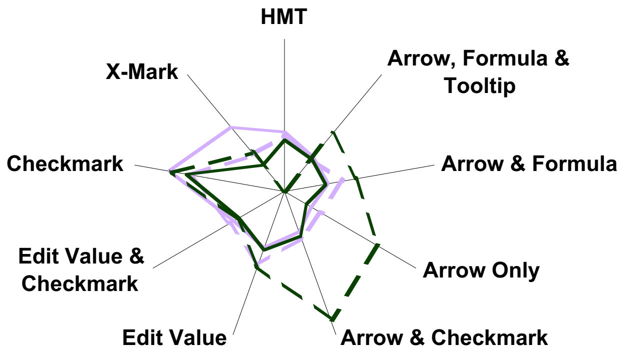
Turning to the right half of the graph, which represents arrow-oriented patterns, the successful and unsuccessful females converge with the successful males. Interestingly, regarding this "how" aspect of arrows, there was a striking difference between successful and unsuccessful males. This difference is further illustrated by Figure 4, which shows that, with all arrow patterns combined, the unsuccessful males used arrow patterns far more frequently than the successful males.

The higher frequency of arrow patterns for unsuccessful males coincides with a higher raw count of arrows used. As Table 2 shows, successful males used a median of 12 arrows, whereas unsuccessful males used more than twice as many, 25.5.

One of the most distinctive differences between the successful and unsuccessful males' arrow patterns occurred in the category Arrow & Checkmark. Within this category were two subcategories using the arrow and checkmark features: "Formula Checkmark Arrow", and "Arrow Checkmark Only." In the first, "Formula Checkmark Arrow", the patterns contain a formula-related action then a checkmark, followed by an arrow. For example: (Hide Formula, Checkmark, Arrow On) and (Hide Formula, Checkmark, Arrow Off). Both unsuccessful and successful males used these patterns frequently, in over one third of their debugging sessions.



(a) Male



(b) Everyone

Figure 3: How by success group. Successful: solid line, unsuccessful: dashed line, females: light, males: dark. (Each category is represented by an axis line radiating from the center. Where the polygon crosses an axis represents the frequency of that pattern.) Note the differences in use of arrow-related patterns between successful and unsuccessful males.

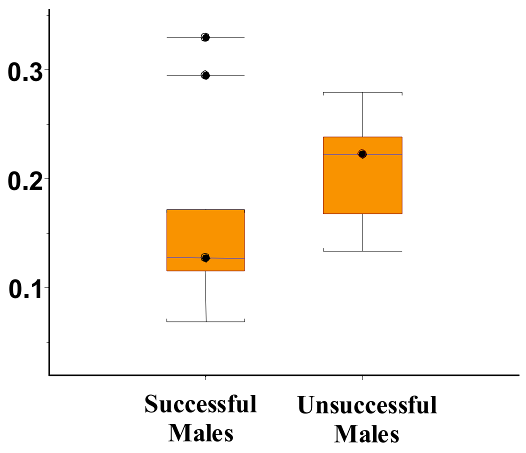


Figure 4: How: Percentage of debugging session that contained arrows by successful versus unsuccessful males.

On the other hand, "Arrow Checkmark Only", while frequently used by the unsuccessful males (in one of every four debugging sessions), was rarely used by successful males (one of every ten debugging sessions). Examples of patterns in this subcategory included: (Arrow On, Checkmark) and (Arrow Off, Checkmark).

Although subtle, these two different strategies could have a large influence on task success. By basing testing decisions solely on the information provided by arrows, as may be the case in the "Arrow Checkmark Only" subcategory, participants may have neglected to take into account the current state of the program execution. In contrast, the "Formula Checkmark Arrow" subcategory is about making a testing decision and then using the arrows, perhaps to direct their next actions.

Hypothesis: Unsuccessful males overdo use of arrows—unlike successful males, successful females, or unsuccessful females.

5.3 Unsuccessful Males: Tinkering Addicts?

We suspected that gender differences in tinkering behavior may be a factor in observed pattern differences. In particular, the unsuccessful males' more frequent use of arrows and their greater variety of arrow-related patterns is suggestive of a larger picture of unsuccessful males tinkering with arrows, to their detriment.

In fact, in previous work, we reported results in which males were found to do more unproductive tinkering, using a different environment [6]. However, the definition of tinkering used in that paper was necessarily simple—and its simplicity prevented it from capturing the excessive exploring/playing the unsuccessful males did in the High-Support Environment. Based on patterns found via mining that data, we are now able to identify more complex tinkering behavior of unsuccessful males in this environment, which we failed to notice in our previous study.

For example, referring to Figure 3, notice the large differences in pattern frequencies for unsuccessful versus successful males in the Arrows Only category, which contains patterns that involve only arrow operations. Two representative patterns in this category were (Arrow Off, Arrow On) and (Arrow Off, Arrow Off). Unsuccessful males had more frequent occurrences of these patterns—one out of every four debugging sessions for the unsuccessful males versus only one out of 20 for the successful males.

Hypothesis: Unsuccessful males have a tendency to tinker excessively with the features themselves rather than using the features to accomplish their task.

6. Results: Self-Efficacy

How do high and low self-efficacy females and males go about debugging? Since self-efficacy did not give the same groupings of the participants as given by task success, it is useful to consider how self-efficacy related to pattern choices. Recall that *self-efficacy* measures a person's belief in his or her ability to perform a particular task [2].

To investigate the question of whether self-efficacy played a role in pattern usage, we divided the participants into four groups based on their self-efficacy score. In particular, we considered a participant to have high (low) self-efficacy if her or his score was higher (lower) than the median of all participants. See Table 3 for the grouping of the participants.

In previous studies [4], self-efficacy has been shown to be predictive of task success for females. That conclusion also holds for the data examined in this study. Half of the 12 high self-efficacy females were successful but only two out of 11 low self-efficacy females were successful. However, it was not true for males: seven out of 10 high self-efficacy males were successful and half of the low self-efficacy males were successful.

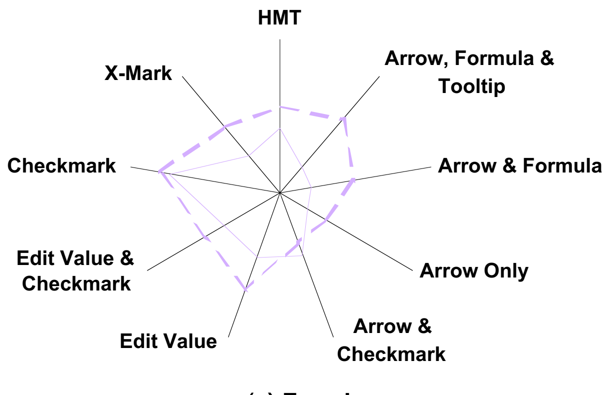
Figure 5 shows that high and low self-efficacy females had pattern frequency profiles that are very distinct from one another, suggesting that self-efficacy made a difference with females. However, the males' self-efficacy did not appear to matter much in their pattern choices.

Patterns alone tell only part of the story. We turned to median raw counts of the number of features used to better understand the reasons behind the patterns that we were seeing. Low self-efficacy female feature counts (Table 3) revealed that low self-efficacy females were the highest usage group for all of the features—except the checkmark.

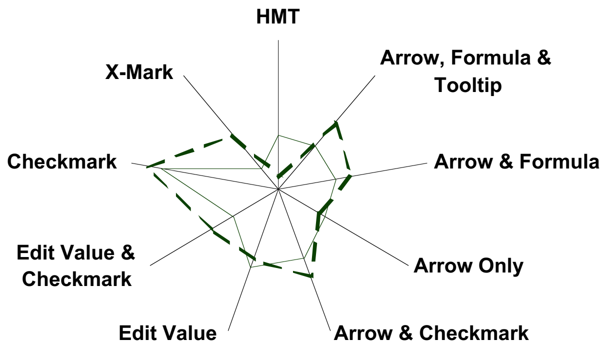
High feature usage by low self-efficacy females may at first seem to contradict our previous results, which

Table 3: What by self-efficacy group. We divided the participants into four groups based upon their gender and pre-task self-efficacy. The rest of the table shows median raw counts of the number of features used during the experiment.

Group	Number of participants	Arrow	X-mark	Check mark	HMT
High Females	12	10.5	3	65.5	5
Low Females	11	24	8	45	8
High Males	10	20	2	52	1.5
Low Males	6	20	5.5	39	3



(a) Female



(b) Male

Figure 5: How by self-efficacy group. High self-efficacy: solid line, low self-efficacy: dashed line.

showed that for females, high self-efficacy predicted more (effective) use of features, which in turn led to greater debugging success [4]. We proposed that offering greater support in the environment would encourage low self-efficacy females to use the features more. The current study used the High-Support Environment, which included features designed to fix that very problem. Our results show that they worked—the low self-efficacy females did indeed use the features in this version! But our current study suggests that quantity of feature adoption is misleading in isolation: feature adoption must be considered in conjunction with *how* the features are used.

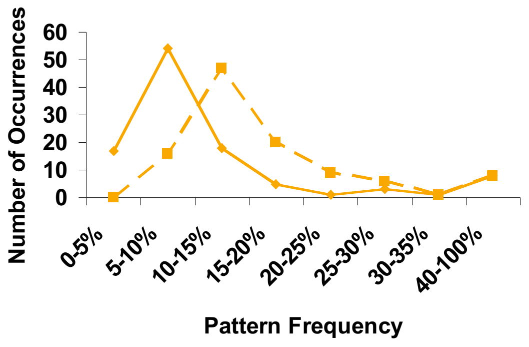


Figure 6: How: The high self-efficacy females (solid line) had more patterns fall in the frequency range of 5-10%, where as the low self-efficacy females had more of their patterns fall a bit higher, around 10-15% of debugging sessions.

High and low self-efficacy females diverged in both counts and patterns. Notice in Figure 5 how many patterns the low self-efficacy females used compared to high self-efficacy females, except for the checkmark. As suggested by self-efficacy theory, people with high self-efficacy are more likely to abandon faulty strategies faster than those with low self-efficacy [2]. Our results were consistent with this. For patterns other than the checkmark patterns, as suggested by Figure 6, the high self-efficacy females were willing to try out and quickly abandon many patterns in order to settle upon the ones they liked, whereas the low self-efficacy females were more likely to try a pattern again and again before ultimately moving on. For example, 54 patterns were used 5-10% of the time by high self-efficacy females, but only 16 were abandoned so quickly by the low self-efficacy females. This leads to the following hypothesis:

Hypothesis: Females with lower self-efficacy are likely to struggle longer to use a strategy that is not working well, before moving on to another strategy.

7. Some Lessons Learned

This research is the first exploration into using data mining in the investigation of possible gender differences in the way females and males went about problem-solving their programs. We briefly share the lessons we learned with the community so that others can gain from these lessons.

In short, these lessons were: (1) Data mining is no panacea. Many of us authors did not anticipate the amount of work required to identify useful patterns of human behavior and to interpret the results. (2) Data mining does not eliminate all bias. Human judgment can impact the results in many ways, such as in determining pattern definitions, thresholds and most importantly how to interpret patterns. Still, despite these somewhat rude awakenings, (3) it was worth it! Despite the fact that we had previously gone over the same data ourselves extensively, data mining turned up patterns for which we might never have thought to look.

8. Conclusions and Future Work

In this paper, we have reported new data-derived hypotheses about gendered patterns in the way females and males used our available spreadsheet debugging features. Because these data had already been analyzed, we have employed a conservative approach: we have used data mining solely to generate hypotheses that must be tested in later studies. These hypotheses are based on evidence in the data, as revealed by the data mining approach.

The new hypotheses related to the following results:

- Patterns used by the successful males were not a recipe for success for the females.
- Unsuccessful males used many more arrow patterns beyond those that appeared useful to the successful males.
- Self-efficacy, again, impacted females' choice of patterns, but not males'. This is the fourth study showing ties between females' self-efficacy and feature usage.

We caution that these are hypotheses and, although they are data-derived, they require future empirical study to accept or refute them.

Acknowledgments

We thank Susan Wiedenbeck for ongoing words of wisdom and help related to this work. This work was supported in part by Microsoft Research, by NSF grant CNS-0420533, by an IBM Faculty Award, and by the EUSES Consortium via NSF grant ITR-0325273.

References

[1] Agrawal, R. and Srikant, R. Mining sequential patterns. *Eleventh Int. Conf. Data Engineering*, 1995, 3-14.

[2] Bandura, A. *Social Foundations of Thought and Action.* Prentice Hall, Englewood Cliffs, NJ, 1986.

[3] Beckwith, L. and Burnett M. Gender: An important factor in end-user programming environments? *IEEE Symp. Visual Languages and Human-Centric Computing*, 2004, 107-114.

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

[5] Beckwith, L., Sorte, S., Burnett, M., Wiedenbeck, S., Chintakovid, T., and Cook, C. Designing features for both genders in end-user software engineering environments. *IEEE Symp. Visual Languages and Human-Centric Computing*, 2005, 153-160.

[6] Beckwith, L., Kissenger, C., Burnett, M., Wiedenbeck, S., Lawrance, J., Blackwell, A., and Cook, C., Tinkering and gender in end-user programmers' debugging. *ACM Conf. Human-Computer Interaction*, April 2006, 231-240.

[7] Beyer, S., DeKeuster, M., Rynes, K., Kosman, A., and DeGregorio, N. Barriers to women's success in Management Information Systems courses. *American Psychological Society*, 2004.

[8] Blackwell, A. First steps in programming: A rationale for attention investment models. *IEEE Symp. Human-Centric Computing Languages and Environments*, 2002, 2-10.

[9] Burnett, M., Cook, C. and Rothermel, G. End-user software engineering. *Communications of the ACM 47*(9), 2004, 53-58.

[10] Busch, T. Gender differences in self-efficacy and attitudes toward computers. *J. Educational Computing Research 12*, 1995, 147-158.

[11] Carroll, J. *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*, MIT Press, Cambridge, MA, 1990.

[12] Cervone, G. and Michalski, R. Modeling user behavior by integrating AQ learning with a database: Initial results. *Intelligent Information Systems*, 2002, 43-56.

[13] El-Ramly, M., Stroulia E., and Sorenson, P. From run-time behavior to usage scenarios: An interaction-pattern mining approach. *ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2002, 315-323.

[14] Fisher, A., Margolis, J. and Miller, F. Undergraduate women in computer science: Experience, motivation, and culture. *ACM SIGCSE Technical Symp. Computer Science Education*, 1997, 106-110.

[15] Hatonen, K., Klemettinen, M., Ronkainen, P., and Toivonen, H. Knowledge discovery from telecommunication network alarm data bases. *12th Int. Conf. Data Engineering*, 1996, 115-122.

[16] Huff, C. Gender software design and occupational equity. *ACM SIGCSE Bulletin* 34(2), 2002, 112-115.

[17] Mannila, H., Toivonen, H., and Verkamo, A. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1997, 259-289.

[18] Mobasher, B., Cooley, R., and Srivastava, J. Automatic personalization based on web usage mining. *Communications of the ACM*, 8. 2000, 142-151.

[19] Perkowitz, M. and Etzioni, O. Adaptive web sites: Automatically synthesizing web pages. *15th Nat. Conf. Artificial Intelligence*, 1998.

[20] Ruthruff, J., Burnett, M., and Rothermel, G. An empirical study of fault localization for end-user programmers. *Int. Conf. Software Engineering*, 2005, 352-361.

[21] Seno, M. and Karypis, G. SLPMiner: An algorithm for finding frequent sequential patterns using length decreasing support constraint. *ICDM'02*, 2002, 418-425.

[22] Simon, H. The structure of ill-structured problems. *Artificial Intelligence*, 4, 1973, 181-202.

[23] Strauss, A. and Corbin, J. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, Second Edition, SAGE Publications, Thousand Oaks, CA, 1998.

[24] Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., Durham, M., Rothermel, G. Harnessing curiosity to increase correctness in end-user programming, *ACM Conf. Human Factors in Computing Systems*, 2003, 305-312.