Tinkering and Gender in End-User Programmers' Debugging

Laura Beckwith[1], Cory Kissinger[1], Margaret Burnett[1], Susan Wiedenbeck[2], Joseph Lawrance[1], Alan Blackwell[3], and Curtis Cook[1]

[1]Oregon State University
Corvallis, Oregon, USA
{beckwith, ckissin, burnett, lawrance, cook}@eecs.oregonstate.edu

[2]Drexel University
Philadelphia, Pennsylvania
susan.wiedenbeck@ischool.drexel.edu

[3]University of Cambridge
Cambridge, United Kingdom
Alan.Blackwell@cl.cam.ac.uk

ABSTRACT

Earlier research on gender effects with software features intended to help problem-solvers in end-user debugging environments has shown that females are less likely to use unfamiliar software features. This poses a serious problem because these features may be key to helping them with debugging problems. Contrasting this with research documenting males' inclination for tinkering in unfamiliar environments, the question arises as to whether encouraging tinkering with new features would help females overcome the factors, such as low self-efficacy, that led to the earlier results. In this paper, we present an experiment with males and females in an end-user debugging setting, and investigate how tinkering behavior impacts several measures of their debugging success. Our results show that the factors of tinkering, reflection, and self-efficacy, can combine in multiple ways to impact debugging effectiveness differently for males than for females.

Author Keywords

Gender, debugging, end-user programming, end-user software engineering, tinkering, self-efficacy, Surprise-Explain-Reward.

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; H.5.2 [Information Systems]: Information Interfaces and Presentation—User Interfaces (D.2.2, H.1.2, I.3.6)

CHI 2006, April 22-28, 2006, Montréal, Québec, Canada.

INTRODUCTION

In educational settings, tinkering, or playful experimentation, is often encouraged because of its expected educational benefits [19]. In our research into how to support end users who perform programming-like tasks, such as creating and debugging spreadsheet formulas, we began to suspect that tinkering may be a critical factor. For example, tinkering may yield positive benefits by making the user feel in control of the system. Also, users may incidentally gain knowledge of the system through tinkering, such as about what features are available, how to invoke them, and what kind of feedback is available after actions.

However, education literature reports that tinkering is a strategy more often adopted by males than females [10, 16, 23], possibly leaving females without the advantages that can be gained from this strategy. If these same gender-related findings apply to the end-user debugging domain (e.g., [2, 6, 13, 24]), they may interact with other factors e.g., self-efficacy.

In a previous experiment, we found that females had lower self-efficacy, expressed as confidence in their ability to debug spreadsheets. They also showed lower acceptance of new features that assist in correcting spreadsheet errors [3]. These differences are troubling, because use of the new features was a significant predictor of debugging success.

In order to investigate whether gender, tinkering, and self-efficacy interact to predict effective use and understanding in end-user debugging, we conducted an empirical investigation using two end-user debugging environments. One of the environments was optimized for low cost in terms of user effort and the other optimized for high user support in terms of both the features and explanations available to the users. Our overall research question for this investigation was: *How do gender, tinkering, self-efficacy, and effectiveness interact in end-user debugging?*

BACKGROUND

Research over two decades indicates that a playful approach to learning increases motivation to learn and the corresponding ability to perform tasks effectively [17, 26].

Similarly, learning can be enhanced through arousing curiosity, by providing change, complexity, or attention-attracting features that motivate exploration of an environment [14, 15, 28]. Tinkering and curiosity are related because tinkering, as an informal, unguided exploration initiated by features visible in the environment, is one way to satisfy one's curiosity.

Curiosity-based exploration is a familiar phenomenon in science and technology education. Research emphasizes the potential value of open-ended exploration in learning [19]. Other educational research has identified gender differences in exploratory behaviors. Among primary school students, studies in mathematics, geography, and gaming indicate that boys tend to tinker and to use tools in exploratory, innovative ways. Girls are less likely to tinker, preferring to follow instructions step-by-step [10, 16, 23]. Similar tinkering findings are also true of males majoring in computer science [8, 21], but not of the female computer science majors. The consistency of these reports led us to believe that the propensity to tinker might play an important role in end-user debugging effectiveness.

To investigate whether there were any indications of tinkering in our environment, we began with a qualitative follow-up analysis of data from our previous study [3]. A clear pattern was found in the use of a debugging feature. In the environment, users were able to place an X-mark in a cell if they thought that the value might be wrong, and we noticed some participants placing an X-mark and then removing it again before taking any other action. Occasionally this corrected a slip, but usually there was no obvious goal-oriented explanation for placing and removing the X-mark. Having discovered this seemingly non-goal-oriented behavior, we further observed that the participants who did this tinkering were consistently males. A quantitative analysis of these data confirmed that males did significantly more X-mark tinkering than females.

Given this evidence of tinkering behaviors in our previous study's data, along with that study's results tying low self-efficacy in females to their lack of acceptance of important debugging features, we began to wonder whether there is a tie between tinkering and self-efficacy in the sort of problem-solving software environment used by end-user programmers. Bandura [1] defines self-efficacy as a task-specific assessment of one's belief in their own ability to accomplish a particular task. Self-efficacy is important to learning and effective performance because an individual with high self-efficacy will be more willing to take on hard tasks, expend significant effort, develop coping strategies, and persist in the face of obstacles [1]. Numerous studies show that high self-efficacy before carrying out a task is predictive of effective task performance, e.g. [1]. Our own research has shown that in the domain of end-user debugging, self-efficacy is a key predictor of females' effectiveness: females have lower self-efficacy than males and their lower self-efficacy is related to less effective performance.

The combined evidence on both tinkering and self-efficacy in our own research and in the literature, led to the empirical study we report in this paper, investigating the effect of tinkering and its relationship to gender, self-efficacy and debugging effectiveness.

EXPERIMENT

Our experiment was designed to consider the effects of two treatments, *Low-Cost* and *High-Support*, on males' and females' tinkering, self-efficacy, and debugging in a spreadsheet environment. In the Low-Cost treatment, tinkering was easy to do, since the cost in terms of user action was low. The High-Support treatment, as first suggested in [4], was designed to provide greater support for the debugging features, but had the side effect of increased tinkering cost.

The underlying spreadsheet environment for the two treatments was the same; this is presented first, followed by the specific interaction devices for each treatment.

Environment

The debugging features that were present in this experiment are 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" or "X out" values that are correct or incorrect, respectively [6]. In addition, arrows that allow users to see the dataflow relationships between cells also reflect WYSIWYT "testedness" status at a finer level of detail.

Using WYSIWYT, once a user makes a decision about a cell's value the system provides visual feedback reflecting the updated "testedness" status. For example, if a user places a checkmark (√) – as can be seen in Figure 1 on the Exam_Avg cell – the system updates the progress bar at the top of the spreadsheet, the cell border colors, and the color of any visible dataflow arrow(s) to reflect the new testedness. Untested elements are colored red (light gray in this paper), partially tested elements are a shade of purple (intermediate shades of gray), and fully tested elements are blue (black). The optional dataflow arrows (seen in Figure 2) are colored to reflect testedness of specific relationships between cells and sub-expressions.

Feedback of the kind seen on the interior of some cells in Figure 2 is provided when a user decides a cell's value is wrong and places an X-mark (cell Course_Avg in Figure 2)

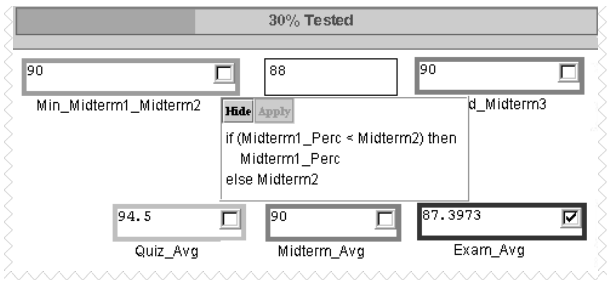

Figure 1. An example of WYSIWYT.

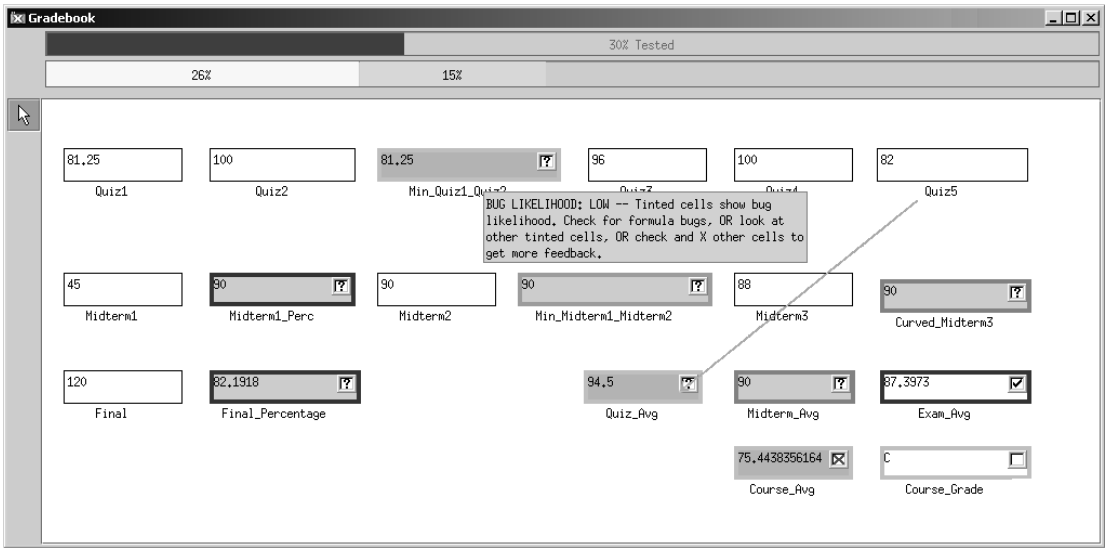

Figure 2. The user notices an incorrect value in `Course_Avg`—the value is obviously too low—and places an X-mark in the cell. As a result of this X and the checkmark in `Exam_Avg`, eight cells are highlighted as being possible sources of the incorrect value, with some deemed more likely than others.

instead of a checkmark. Behind the scenes, X-marks trigger fault likelihood calculations, which cause cells suspected of containing faults to be highlighted in shades along a yellow-orange continuum (shades of gray in this paper), with darker orange shades given to cells with increased fault likelihood [6].

Each of these features is supported through the mechanisms of the Surprise-Explain-Reward strategy. This strategy relies on a user's curiosity about features in the environment. The user can then seek out explanations (Figure 2) of the features via tool tips [28]. The aim of the strategy is that, if the user follows up as advised in the explanation, rewards will ensue [20].

Two Treatments

The Low-Cost and High-Support treatments varied in three ways: WYSIWYT input devices, explanation content and interaction, and the number of task-supporting features available within the environment.

Low-Cost

In the Low-Cost treatment, WYSIWYT interaction required one click for placing or removing a testing decision (left-click for checkmark, right-click for X-mark). Explanations, provided through tool tips, were as short as possible, to keep their reading cost low [28].

High-Support

The aim of the High-Support treatment was two-fold, first to encourage low self-efficacy users to take advantage of the features that can help users debug, and secondly to provide an environment in which learning was supported through fuller explanation content. A side effect of the additions was a higher user-action cost, requiring more clicks, reading, and choice of features to use. This higher cost applied to tinkering as well as to other actions.

In the High-Support treatment, along with the checkmark meaning that a value is correct and X-mark meaning that it is incorrect, the users also could make decisions for values that "seem right maybe," or "seem wrong maybe." The purpose of this mechanism was to encourage low self-efficacy users by reassuring them that confident decisions were not a prerequisite in using the devices. The colors reflecting these more tentative "seems" decisions were the same hues but less saturated than those of the other decisions [4]. The system's inferences about *which* cells were tested or faulty was the same as for the Low-Cost treatment, but the system also propagated the amount of tentativeness, allowing the user to discern which statuses were based on the "seems" decisions. The input device required a user first to click on the "?" in the decision box (see Figure 2), which brings up the four choices shown in Figure 3, and then to click on their choice. (In contrast, recall that placing a checkmark or X-mark in the Low-Cost treatment required only one click.)

In addition, the explanations were expanded to support users who wanted more guidance than the explanations given in the Low-Cost treatment. The mechanism was as

Figure 3. Clicking on the decision box turns it into the four choices. Each choice has a tool tip, starting with the left-most X these are "it's wrong," "seems wrong maybe," "seems right maybe," "it's right."

follows. In addition to the tool tip content of the Low-Cost treatment, additional information was available via a "Tips" expander (Figure 4), which could be expanded and dismissed on user demand. The expanded "Tips" included further information on why the object was in its current state and possible actions to take next. Once expanded, the tip would stay visible until the user dismissed it, supporting non-linear problem solving and requiring less memorization by the user.

The "Help Me Test" feature [28] was provided to the High-Support group (but not to the Low-Cost group) to help users overcome difficulties in finding additional test cases. Sometimes it can be difficult to find test values that will cover the untested logic in a collection of related formulas, and Help Me Test tries to find inputs that will lead to coverage of untested logic in the spreadsheet, about which users can then make testing decisions. Help Me Test is not automated testing but rather scaffolding: it provides new test inputs, but does not make decisions about the outputs that result, so does not actually "test" the spreadsheet.

Procedures

The participants were randomly divided into two groups: a group of 37 participants (20 males and 17 females) received the Low-Cost treatment, and a group of 39 participants (16 males and 23 females) received the High-Support treatment. We recruited participants from the university and community; we required all participants to have some spreadsheet experience, but only limited programming experience. Statistical tests on questionnaire data showed no significant differences between the groups in grade point average, spreadsheet experience, or programming experience.

A pre-experiment questionnaire collected participant background data and also contained 10 self-efficacy questions based on Compeau and Higgins' validated scale [7]. This pre-self-efficacy questionnaire was modified with task-specific end-user debugging questions (on a five-point Likert scale) such as "I could find and fix errors if there was no one around to tell me what to do as I go."

We administered a 35-minute "hands-on" tutorial to familiarize participants with their treatment. The participants were then given two tasks. We captured their actions in electronic transcripts, as well as their final spreadsheets.

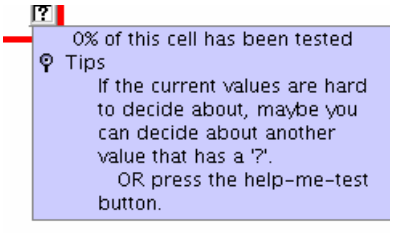

Figure 4. From the non-expanded tool tip (the first line) a user can click on the Tips expander to get the rest of the tool tip – which remains on the screen until dismissed.

At the conclusion of each task, we administered questionnaires which included questions regarding how users perceived their performance on that task. The final questionnaire included a follow-up post-self-efficacy questionnaire identical to the pre-self-efficacy questionnaire, as well as questions assessing participants' comprehension of the X-mark feature and their attitudes toward the features they had used. Taking two measures of self-efficacy (one prior to the experiment and another following the final task) is valuable information because according to self-efficacy theory people working in a new and unfamiliar environment have malleable self-efficacy much of which is based on their first experiences, and in particular early perceived failures can have especially pronounced effects on self-efficacy.

Tutorial

In the tutorial, participants performed actions on their own machines with guidance at each step. Although the Low-Cost and the High-Support tutorials both described the checkmark feature (including its associated testedness-colored arrows feature), neither tutorial included any debugging or testing strategy instruction. Furthermore, neither tutorial explained the X-mark feature beyond showing that it was possible to place X-marks (with time to explore any aspects of the feedback – through explanations – that they found interesting). At the end of the tutorial, we gave both groups time to explore the features they had just learned by working on the tutorial spreadsheet debugging task.

The High-Support tutorial explained the additional features of the treatment, allowing ample time to explore the choices in check and X-marks (Figure 3), the Help Me Test feature, and the expanded tool tips. To compensate for the extra time it took to explain the additional features in the High-Support treatment, the Low-Cost group had several extra minutes at the end of the tutorial to explore and/or work further with tutorial spreadsheet debugging task.

Half of the tutorial sessions were presented by a male and half by a female, balanced so that 50% of participants were instructed by a same-gender instructor and 50% by the opposite gender [27].

Tasks

We asked participants to test two spreadsheets, `Gradebook` (Figure 2) and `Payroll` (screenshot available in [3]). We replicated the spreadsheets and the seeded faults of [3]. (The spreadsheets were originally derived from real end-user spreadsheets and seeded with faults from end users. Each spreadsheet contained five faults representative of the fault categories in Panko's classification system [18].)

Participants had time limits of 22 and 35 minutes for `Gradebook` and `Payroll` respectively. These simulated the sort of time constraints that often govern real world computing tasks and prevented potential confounds, such as participants spending too much time on the first task or not enough time on the second task, participants leaving early, and so on. The use of two spreadsheets reduced the chance of the results being due to any one spreadsheet's particular characteristics. The experiment was counterbalanced with respect to task order in order to distribute learning effects evenly. The participants were instructed, "Test the … spreadsheet to see if it works correctly and correct any errors you find."

RESULTS

We have already pointed out that ties have been found between tinkering and educational goals, and within that context tinkering seems to be a male characteristic. However, in the domain of end-user debugging, the goal is not education per se, but rather productivity or effectiveness in fixing the bugs. Still, educating oneself about features that seem useful to the task could be a necessary subgoal.

Thus, we consider whether there were gender differences in tinkering as a way to master new features, and how such differences might tie to debugging effectiveness.

Tinkering by Gender: How Much

Our measures were tinkering frequency, tinkering episodes, and tinkering rate within episodes (a measure of commitment to tinkering within an episode). We operationally define a *tinkering instance* as turning a feature "on" immediately followed by turning the feature "off," such as placing a checkmark or turning on an arrow and then removing it as the next action. Although a tinkering instance is simple to perform in this environment, the complex feedback users receive when tinkering is where the constructive experience begins to occur: through users' tinkering actions they can construct concrete, visual paths backwards ("breadcrumbs" of where they've been) and forwards (where they need to go to achieve 100% testedness). *Tinkering frequency* is simply a count of the number of tinkering instances. A tinkering *episode* is defined to be a sequence of one or more tinkering instances, terminated by a cell edit or the end of the task. The episode count for each participant serves as a measure of consistent use of tinkering. Finally, *tinkering rate*, computed as tinkering frequency per episode, measures "vestedness" in tinkering within an episode: once a participant starts to tinker, how committed does he or she stay to tinkering before moving on (indicated by editing a cell)?

Our expectations, given previous literature, were that males would make greater use of tinkering than females by all of these measures, regardless of treatment.

Our expectations were wrong. The analysis of the tinkering frequency measure (illustrated in Figure 5) revealed that the Low-Cost males stood apart from the others. A 2 (gender) by 2 (treatment) ANOVA revealed a significant main effect of treatment ($p<0.01$, $F[1,72]=7.15$) and a significant interaction effect ($p<0.05$, $F[1,72]=4.42$), although the main effect of gender alone fell short of significance at the .05 level ($p<0.10$, $F[1,72]=2.82$). Thus, the treatment affected the genders differently. In particular, treatment made almost no difference for the amount of tinkering females did. However, for males a follow-up analysis (using the Tukey method) revealed a significant effect of treatment ($p<0.05$) on their tinkering frequency.

Table 1 shows the means of each gender for all three of our tinkering measures (for completeness each of the measures is reported in this table, however, discussion of tinkering rate is in later sections). On the tinkering consistency measure (number of episodes) ANOVA analysis still showed a significant effect of treatment ($p<0.01$, $F[1,72]=9.64$), but did not approach significance for gender or for gender x treatment. Thus, as the episodes rows in Table 1 show, the High-Support group had less consistent emphasis on tinkering as a problem-solving device.

Discussion:

Even though males and females had equal tinkering opportunities, these results show trends towards opposite effects of treatment on gender in the Low-Cost treatment. In particular, we found a surprisingly large effect of treatment

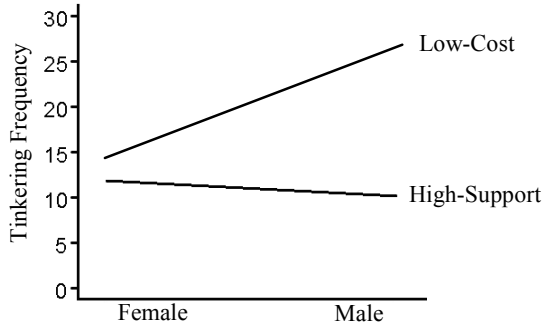

Figure 5. This interaction plot of 4 means depicts the gender x treatment interaction in tinkering frequency.

	Low-Cost	High-Support	*p*-value
Frequency			Gender: <0.10
Males	27.3	10.1	**Treatment: <0.05**
Females	13.7	11.7	**Interaction: <0.05**
Episodes			Gender: 0.769
Males	9.5	5.1	**Treatment: <0.05**
Females	7.9	6.0	Interaction: 0.230
Rate			Gender: 0.142
Males	2.6	1.8	Treatment: 0.251
Females	1.7	1.9	Interaction: <0.10

Table 1. Means of tinkering measures. Results for each measure are shown as a group at the right, with significant results (<0.05) bold faced, marginally significant results (between 0.05 and 0.10) in regular font, and insignificant results in grey font.

on males' tinkering, which we will consider further in later sections. We now turn our attention to whether tinkering actually helped either gender in their debugging efforts.

Does Tinkering Matter to Effectiveness?

In educational settings exploration has been encouraged for improved performance [19]. Although the education setting is different from the domain of end-user debugging, our expectations were that high-tinkering males would be more effective than the others since their tinkering is higher than the other participants.

To investigate this possibility, we used the following dependent measures. The first was bugs fixed, because fixing bugs was an explicit goal assigned to the participants. The second was percent testedness of the spreadsheet (as seen at the top of Figure 1), since in previous experiments this has been significantly tied with success in debugging [3, 6]. This relationship of percent testedness to bugs fixed was found again in this experiment for both genders (linear regression, males: $p<0.01$, $F[1,34]=27.16$, $R^2=0.44$; females: $p<0.01$, $F[1,38]=10.51$, $R^2=0.22$). The third was the participants' understanding of the debugging feature (X-mark) as measured in the post-session questionnaire.

A 2 (gender) by 2 (treatment) ANOVA showed no significant differences of the outcomes of these measures by gender, treatment, or gender x treatment, as Table 2 suggests. However, there were surprising gender differences in the ways tinkering predicted these results.

Figure 6 summarizes the differences in the way tinkering related to effectiveness for the males versus the females. As it shows, males' and females' tinkering affected their debugging effectiveness, but in essentially opposite ways. Females' tinkering was a significant predictor of their final percent testedness (linear regression, episodes: $p<0.05$, $F[1,38]=4.63$, $R^2=0.11$). Recall from above that final percent testedness was in turn highly predictive bugs fixed. (That is, tinkering did not directly predict bugs fixed; rather testedness was a mediating factor for the relationship.)

For the males, however, no measure of tinkering was predictive of their percent testedness. Thus no mediated relationship to bugs fixed existed and, in fact, tinkering rate was found to be a *negative* predictor of bugs fixed (linear regression, $p<0.01$, $F[1,34]=8.04$, $R^2=0.19$).

	Low-Cost	High-Support
Bugs fixed		
Males	5.9	6.5
Females	6.3	5.3
% testedness		
Males	61.9	63.2
Females	67.9	65.3
Understanding		
Males	6.8	7.7
Females	8.1	7.1

Table 2. Means of effectiveness measures.

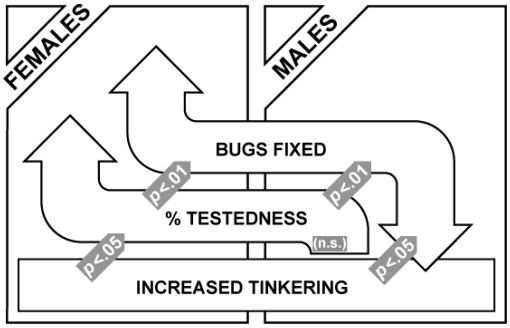

Figure 6. Left: females, Right: males. Direction of stylized arrows depicts increase/decrease in a measure, and shaded arrows show significance of the regression relationships between measures.

Discussion:

Although there is previous research, including our own, showing that software environments are often designed in ways better aligned with males' need rather than females' needs [3, 9], in the realm of tinkering, this section's results point to a disadvantage for the males. In particular, in contrast to our expectations, tinkering was tied to negative outcomes for the males. The females, however, had positive outcomes tied with tinkering.

Tinkering for females showed similar benefits to understanding as it did for effectiveness: Their tinkering significantly predicted their understanding (linear regression, frequency: $p<0.05$, $F[1,38]=4.56$, $R^2=0.11$; episodes: $p<0.05$, $F[1,38]=4.44$, $R^2=0.10$). For the males no measure of tinkering was predictive of their understanding.

Tinkering and Self-Efficacy

As referred to earlier, in our previous work we established pre-self-efficacy as an important factor in female end-users' debugging. The current study was motivated by that work, and has confirmed some of the results we found then (see Table 3). With respect to tinkering and pre-self-efficacy, we expected that increased tinkering would increase females' post-self-efficacy. We also expected the set of features in the High-Support treatment would increase females' post-self-efficacy; however, adding additional features carries the risk of reducing self-efficacy—even if the features provide more support—by making the environment more complex.

	p-value	F [1,74]	R^2
Previous experience → SE	**<0.05**	4.46	0.06
SE → % testedness	**<0.01**	7.10	0.09
SE → Understanding	**<0.01**	11.38	0.13

Table 3. Self-efficacy (SE) results of current experiment replicated those from [3]. Previous experience was predictive of pre-self-efficacy, and pre-self-efficacy was predictive of the two other factors.

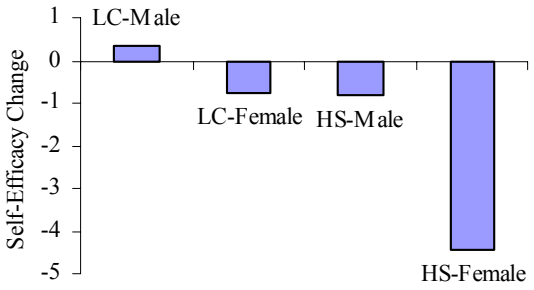

Figure 7. The change from pre-self-efficacy to post-self-efficacy. The High-Support females' drop was significant.

In fact, we found a dramatic fall in post-self-efficacy for females using the High-Support treatment (Figure 7). Most groups had little to no difference in their self-efficacy change between pre-self-efficacy and post-self-efficacy. However, the High-Support females' self-efficacy dropped significantly over the course of the study (paired t-test, $p<0.01$, DF=22, t=3.19).

The relationships between tinkering and post-self-efficacy were also surprising. For High-Support females, the rate of tinkering per episode was predictive of the *drop* in self-efficacy reported above (linear regression, $p<0.05$, $F[1,21]=6.32$, $R^2=0.23$). Also, we found suggestive evidence linking increased tinkering frequency to increased post-self-efficacy for the females in the Low-Cost condition (linear regression, $p<0.10$, $F[1,15]=3.51$, $R^2=0.19$). Males, however, did not exhibit any significant relationships between their tinkering and post-self-efficacy.

Discussion:

What conclusions can be drawn from the extreme drop in self-efficacy by High-Support females and the relationships between tinkering and post-self-efficacy? One possible conclusion is that the High-Support females did not perceive tinkering as helpful for understanding how their debugging environment worked. Therefore, the more they tinkered, the more they reinforced their perception of their inability to understand what was happening in the environment. This relates to a finding in our previous experiment [3], in which females believed more than males that it would take them too long to learn the X-mark feature—even though in actuality they understood it as well as the males.

Taken in combination with our current tinkering effectiveness results, it appears that the tinkering issue for females is complex. Females were better than the males at consistently extracting problem-solving benefits from tinkering, but were worse than the males at maintaining their self-efficacy levels.

TINKERING CONSIDERED HARMFUL?

Recall that male tinkering was highly dependent upon the treatment, with the males in the Low-Cost group being significantly higher tinkerers than the others. A closer look at the understanding scores among the Low-Cost males revealed that their understanding scores tended to be quite extreme: either nearly perfect or extremely low. To shed light on potential factors that may explain these results, we consider *types* of tinkering and also tinkering's relationship to *reflection*.

Tinkering: Exploratory and Repeated

First we consider two types of tinkering, which may help explain why males' tinkering was not effective, at least from a statistical standpoint, whereas females' tinkering was effective.

For example, one of the Low-Cost males was observed turning on a feature, then immediately turning off that feature, and then again turning the same feature on and off again, all actions occurring on the same cell. This participant's understanding score was very low (5.5 out of a possible 12) and he did not fix any of the 10 bugs, suggesting that this type of tinkering was not particularly useful to him.

To investigate whether different types of tinkering might have opposing effects—some increasing debugging effectiveness and some interfering with it—we partitioned the tinkering behaviors into two subsets: exploratory tinkering and repeated tinkering. *Repeated tinkering instances* are the number of tinkering instances in a sequence of two or more consecutive tinkering instances on the same feature and the same cell, as with the Low-Cost male above. For example, turning the arrows for a cell on and then immediately back off again three times in a row is three repeated tinkering instances. Hence, the repetitions can only repeat information already revealed by the previous tinkering instances. *Exploratory tinkering instances*, which we define as the difference between total tinkering instances and repeated tinkering instances, potentially can impart new information.

Oddly, the males' exploratory tinkering was *not* statistically predictive of understanding or of any of the effectiveness measures. In contrast to this, an increase in female exploratory tinkering (which accounted for 91% of their overall tinkering) was a significant predictor of increased understanding (linear regression, $p<0.05$, $F[1,38]=4.61$, $R^2=0.11$).

Interestingly, as Figure 8 shows, repeated tinkering instances accounted for a significantly greater proportion of the Low-Cost males' repeated tinkering than any other group, with a significant effect of interaction between gender and treatment (ANOVA, $p<0.05$, $F[1,72]=5.82$). In fact, nearly 17% of the Low-Cost males' tinkering instances were of this type, almost twice as many as the next highest group.

Repeated tinkering, which was done predominantly by the Low-Cost males, had a significant negative relationship to understanding (linear regression, $p<0.05$, $F[1,18]=6.0$, $R^2=0.25$). See Figure 9.

Discussion:

The High-Support treatment, because of its increase in tinkering cost over the Low-Cost treatment, not only greatly reduced males' tinkering, it *selectively* reduced primarily the ineffective type of tinkering. Since it did not affect the females' tinkering behavior, from a tinkering effectiveness perspective, this approach appears to be the better of the two treatments for both genders, albeit for different reasons. Still, the lack of relationship between males' exploratory tinkering and effectiveness suggests that the different types of tinkering do not alone explain how males' tinkering related to their debugging effectiveness.

Reflection

Research from the education field has shown that when students are given "wait-time" of three seconds or more after a classroom response, their critical thinking about that response improves [19]. To see whether such wait-times also apply to end-user debuggers, we likewise define *pauses* as three or more seconds of inactivity after a user action. Note that here we consider pauses after *any* action (not just tinkering instances), because every action provides feedback. If Rowe's research applies to our domain, then increased pauses between actions should result in increased understanding and effectiveness.

For our analysis, we counted the frequency of pauses. Overall, females had significantly more pauses than males, with a mean of 220 versus 190 occurrences (ANOVA, $p<0.05$, $F[1,74]=4.22$, $R^2=0.05$). Thus, males did not take the time that might have been used to reflect upon the feedback from their actions as often as the females did.

These pauses mattered. Participants who paused long enough to reflect on the system's responses understood the debugging devices better and used features more effectively than the other participants. Specifically, for both genders,

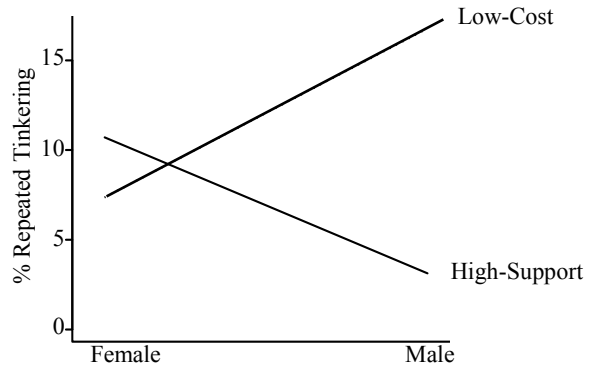

Figure 8. This interaction plot (of means only) illustrates the gender x treatment interaction in percentage of repeated tinkering.

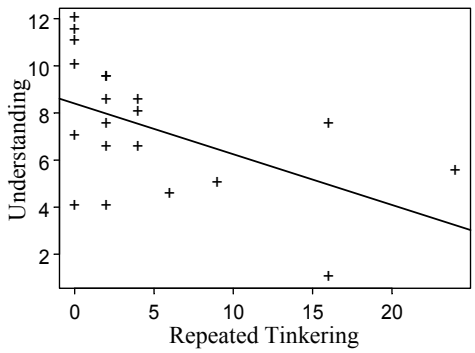

Figure 9. The negative regression relationship between repeated tinkering and understanding for the Low-Cost males.

more frequent pauses were predictive of greater effectiveness, as measured by bugs fixed (linear regression, $p<0.05$, $F=[1,74]=5.36$, $R^2=0.07$), final percent testedness (linear regression, $p<0.01$, $F=[1,74]=31.3$, $R^2=0.30$), and understanding of features (linear regression, $p<0.01$, $F=[1,74]=14.11$, $R^2=0.16$).

One question that arises is whether the longer the pause the better the effect. Our analysis suggests that this is not the case: for both genders, longer pauses (which were also tied with fewer tinkering instances; linear regression, $p<0.05$, $F[1,74]=6.13$, $R^2=0.08$) were not helpful. They were significantly predictive of a decrease in bugs fixed, percent testedness, and understanding (linear regression, bugs fixed: $p<0.01$, $F[1,74]=11.85$, $R^2=0.14$; percent testedness: $p<0.01$, $F[1,74]=37.07$, $R^2=0.33$; understanding: $p<0.01$, $F[1,74]=11.81$, $R^2=0.14$).

Discussion:

These results help to shed light on the inverse relationship between tinkering and effectiveness for males as compared to the positive relationship of tinkering and effectiveness for females. Males' tendency to tinker more appears to be useful only when they make regular use of pauses: and in our study, their tendency to pause too little may have interfered with the benefits of tinkering.

GENERAL DISCUSSION: TINKERING AND BRICOLAGE

Although there have been previous studies of gender issues in professional programming, education, and research careers, it has generally been assumed that the programming language and tools themselves are gender neutral. This presumption has been carried over into end-user programming. Some educational computing environments (e.g. Alice™ and ToonTalk™) attempt to attract girls by including animated female characters, but not by changing the language or its features [11, 12].

An exception to these assumptions is a paper by Turkle and Papert [22], who advocate *bricolage* as an approach to programming that is more appropriate to females, and should be used to inform the design of educational programming environments. They describe bricolage as a

"soft" style of working in which programmers adopt a flexible, concrete, and non-hierarchical approach, rather than a "hard" style of abstract thinking and systematic, hierarchical planning. Their account is supported with observations of female students taking programming courses at Harvard University, who performed better when they could build by experimenting, iterating, and adapting existing objects and code. A "soft" approach to programming can be supported by programming languages and environments that are flexible, for example, allowing both top-down and bottom-up program development.

Our use of the term tinkering, like bricolage, describes concrete, "soft" activity. Our experiment confirms the value of tinkering to females. Having observed the difference in tinkering in our two treatments, we see the design of the programming environment rather than the style of the language as the best way to facilitate productive tinkering.

Turkle and Papert's use of the term bricolage expresses a philosophy of learning by doing, inspired by anthropologist Levi-Strauss, who used the term to describe the intellectual style of non-Western cultures [5]. Turkle and Papert treat bricolage as gender-neutral, perhaps even a female working style. However, in modern French the word is used as a synonym for "DIY" (i.e., hobbyist "Do-It-Yourself" building or decorating). This is a masculine habit in France, as is common usage of the term tinkering in the USA. Consider this recent Washington Post article: "Sometimes, tinkerers become a consumer electronics maker's unofficial research-and-development team…TiVo's most ardent fans came up with a way to [send] commands via the Internet…Americans have always liked to fiddle with things, from building better ham radios to juicing up car engines in the driveway" [25].

These descriptions of the generic American innovator sound depressingly like descriptions of a generic American male. If tinkering is a critical aspect for successful end-user programming, is it acceptable that it be associated so much with the male domain? If we are not careful to define tinkering, or bricolage, as the "soft" approach of Turkle and Papert, it becomes too easy to alienate and exclude females from the advantages of tinkering seen in our research. In this paper, we have demonstrated that some tinkering habits are counter-productive and that these are more often exhibited by males. If we can encourage both females and males to avoid such habits, while exploring end-user programming tools in a productive way, it should be possible to provide genuine benefits for *both* genders.

CONCLUSION

An assumption in much past work on gender differences in computing is that males' behaviors are reasonably well-matched to today's problem-solving features in environments such as for end-user programming. A further assumption is that these environments need to evolve to allow females' behaviors to achieve success at the same levels. Our results show, however, that males' behaviors may sometimes be the ones that are not as well-supported as the females' behaviors.

More specifically, our findings included:

- As in previous research, males tinkered more than females but, surprisingly, males' tinkering was often counterproductive to their effectiveness in debugging.
- One factor in the above result was the fact that the Low-Cost treatment led some males to engage in unproductive, repeated tinkering, which was linked to poor understanding.
- Although they tinkered less, females' tinkering was very effective: it was significantly tied to understanding and to successfully testing and debugging, regardless of treatment. However, when tinkering in the more complex environment, females' tinkering was predictive of lower self-efficacy.
- Tinkering with pauses allows for reflection and was helpful to everyone, but females were more likely than males to pause.

These results show that tinkering can be a valuable activity in end-user debugging, but the prescriptions on how an environment should be designed to guide male and female tinkering are different. Females should be encouraged to tinker because it helps them to be effective, with the important caveat that tinkering in a complex environment carries a risk of damaging the females' self-efficacy. In contrast to this, males' self-efficacy did not seem at risk, but our results suggest that males need to be guided to tinker less repeatedly and more "pausefully."

ACKNOWLEDGMENTS

We thank the participants of our study. This work was supported in part by Microsoft Research, by NSF grant CNS-0420533, and by the EUSES Consortium via NSF grants ITR-0325273 and CCR-0324844. We also thank Sienna Hiebert (supported by Saturday Academy's Apprenticeships in Science and Engineering Program), Shraddha Sorte and Thippaya Chintakovid for their help.

REFERENCES

1. Bandura, A. *Social Foundations of Thought and Action*. Prentice Hall, Englewood Cliffs, NJ, 1986.
2. Beckwith, L. and Burnett M. Gender: An important factor in end-user programming environments? In *Proc. Visual Languages and Human-Centric Computing*, IEEE (2004), 107-114.
3. 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.
4. Beckwith, L., Sorte, S., Burnett, M., Wiedenbeck, S., Chintakovid, T. and Cook, C. Designing features for both genders in end-user programming environments.

In *Proc. Visual Languages and Human-Centric Computing*, IEEE (2005), 153-160.

5. Ben-Ari, M. Bricolage forever! In *Proc. of the 11th Annual Workshop of the Psychology of Programming Interest Group,* (1999), 53-57.
6. Burnett, M., Cook, C. and Rothermel G. End-user software engineering. *Comm. of the ACM 47*, 9 (2004), 53-58.
7. Compeau, D. and Higgins, C. Computer self-efficacy: Development of a measure and initial test. *MIS Quarterly 19*, 2 (1995), 189-211.
8. Fisher, A., Margolis, J. and Miller, F. Undergraduate women in computer science: Experience, motivation, and culture. In *Proc. SIGCSE Technical Symposium on Computer Science Education*, ACM Press (1997), 106-110.
9. Huff, C. Gender, software design, and occupational equity. *ACM SIGCSE Bulletin 34*, 2 (2002), 112-115.
10. Jones, M. G., Brader-Araje, L., Carboni, L. W., Carter, G., Rua, M. J., Banilower, E. and Hatch, H. Tool time: Gender and students' use of tools, control, and authority. *Journal of Research in Science Teaching 37*, 8 (2000), 760-783.
11. Kahn, K. Drawings on napkins, video-game animation, and other ways to program computers. *Comm. of the ACM 39*, 8 (1996), 49-59.
12. Kelleher, C. and Pausch, R. Stencils-based tutorials: Design and evaluation. In *Proc. CHI 2005*, ACM Press (2005), 541-550.
13. Ko, A.J. and Myers, B.A. Designing the Whyline: A debugging interface for asking questions about program failures. In *Proc. CHI 2004*, ACM Press (2004), 151–158.
14. Lepper, M.R. and Malone, T.W. Intrinsic motivation and instructional effectiveness in computer-based education. In R.E. Snow and M.J. Farr (Eds.), *Aptitude, Learning, and Instruction: Vol. 3. Conative and Affective process Analyses,* 255-286. Lawrence Erlbaum, Hillsdale, NJ, 1987.
15. Malone, T.W. and Lepper, M.R. Making learning fun: A taxonomy of intrinsic motivations for learning. In R.E. Snow and M.J. Farr (Eds.), *Aptitude, Learning and Instruction. Volume 3: Conative and Affective Process Analysis,* 223-253. Lawrence Erlbaum, Hillsdale, NJ, 1987.
16. Martinson, A.M. Playing with technology: Designing gender sensitive games to close the gender gap. *Working Paper SLISWP-03-05*, School of Library and Information Science, Indiana University, http://www.slis.indiana.edu/research/working_papers/files/SLISWP-03-05.pdf, accessed Sept., 12, 2005.
17. Martocchio, J.J. and Webster, J. Effects of feedback and playfulness on performance in microcomputer software training. *Personnel Psychology 45*, (1992), 553-578.
18. Panko, R. What we know about spreadsheet errors. *Journal of End User Computing 10*, 2 (1998), 15-21.
19. Rowe, M.B. *Teaching Science as Continuous Inquiry: A Basic (2nd ed.)*. McGraw-Hill, New York, NY 1978.
20. Ruthruff, J., Phalgune, A., Beckwith, L., Burnett, M. and Cook, C. Rewarding 'good' behavior: End-user debugging and rewards. In *Proc. Visual Languages and Human-Centric Computing,* IEEE (2004), 115-122.
21. Tillberg, H.K. and Cohoon, J.M. Attracting women to the CS major. *Frontiers: A Journal of Women Studies 26*, 1 (2005), 126-140.
22. Turkle, S. and Papert, S. Epistemological pluralism and the revaluation of the concrete. *Journal of Mathematical Behavior 11*, 1 (1992), 3-33. Available online at http://www.papert.org/articles/EpistemologicalPluralism.html
23. Van Den Heuvel-Panheizen, M. Girls' and boys' problems: Gender differences in solving problems in primary school mathematics in the Netherlands. In T. Nunes and P. Bryant (Eds.), *Learning and Teaching Mathematics: An International Perspective,* 223-253. Psychology Press, UK, 1999.
24. Wagner, E.J. and Lieberman, H. Supporting user hypotheses in problem diagnosis on the web and elsewhere. In *Proc. of the International Conference on Intelligent User Interfaces*, ACM Press (2004), 30–37.
25. Washington Post. Tapping Into Tinkering: Some Makers of Electronics Benefit From Users' Modifications. July 12, 2005.
26. Webster, J. and Martocchio, J.J. Turning work into play: implications for microcomputer software training. *Journal of Management 19,* 1 (1993), 127-146.
27. Whitworth, J.E., Price, B.A. and Randall, C.H. Factors that affect college of business student opinion of teaching and learning. *Journal of Education for Business 77*, 5 (2002), 282-289.
28. Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., Durham, M. and Rothermel, G. Harnessing curiosity to increase correctness in end-user programming. In *Proc. CHI 2003***,** ACM Press (2003), 305–312.