AN ABSTRACT OF THE THESIS OF

Shreenivasarao Prabhakararao for the degree of Master of Science in Computer Science presented on December 3, 2003.

Title: Strategies and Behaviors of End-User Programmers with Interactive Fault Localization

Abstract approved Redacted for Privacy

Curtis R Cook

End-user programmers are writing an unprecedented number of programs, due in large part to the significant effort put forth to bring programming power to end users. Unfortunately, this effort has not been supplemented by a comparable effort to increase the correctness of these often faulty programs. To address this need, we have been working towards bringing fault localization techniques to end users. In order to understand how end users are affected by and interact with such techniques, we conducted a think-aloud study, examining the interactive, human-centric ties between end-user debugging and a fault localization technique for the spreadsheet paradigm. Our results provide insights into the contributions such techniques can make to an interactive end-user debugging process.

Strategies and Behaviors of End-User Programmers with Interactive Fault Localization

by
Shreenivasarao Prabhakararao

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented December 3, 2003
Commencement June 2004

Master of Science thesis of Shreenivasarao Prabhakararao presented on December 3, 2003.

APPROVED:

Redacted for Privacy

Major Professor, representing Electrical Engineering and Computer Science

Redacted for Privacy

Director of the School of Electrical Engineering and Computer Science

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Shreenivasarao Prabhakararao, Author

ACKNOWLEDGEMENTS

I would first and foremost like to express my gratitude towards my advisor, Dr. Curtis Cook, for his support, encouragement, and guidance. I would also like to thank him for giving me the opportunity to work with him.

I would like to thank Dr. Margaret Burnett for her encouragement and constant guidance.

This work has been done in collaboration with J.Ruthruff, E. Creswick, M. Main, M.Durham, C.Cook and M.Burnett. I would like to thank all of them. I would also like to thank a former graduate student and Forms/3 member James Reichwein, who had developed the Fault localization technique used in this study.

Thanks also, to all the members of Forms/3 team. In specific, I would like to thank J.Ruthruff for his generous help.

Lastly, I'd like to mention a few of the many others in my life who have been extremely supportive as I completed my masters: my parents, Akka, Kishore, Sonali and Soujanya.

This work was supported in part by the National Science Foundation under Awards ITR-0082265.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

Table Page

Strategies and Behaviors of End-User Programmers with
Interactive Fault Localization

1. Introduction

1.1 End-User Programming

Recent years have seen the explosive growth of end-user programming. In fact, by the year 2005, it is estimated that there will be approximately 55 million end-user programmers in the U.S. alone, as compared to an estimated 2.75 million professional programmers [Bohem et al 2000]. Real-world examples of end-user programming environments include educational simulation builders, web authoring systems, multimedia authoring systems, e-mail filtering rule systems, CAD systems, and spreadsheets.

But, how reliable are the programs end users write using such systems? One of the most widely used real-world end-user programming paradigms is the spreadsheet. Despite its perceived simplicity, evidence from this paradigm reveals that end-user programs often contain an alarming number of faults [Panko 1998]. (Following standard terminology, in this thesis we use the term *failure* to mean an incorrect output value given the inputs, and the term *fault* to mean the incorrect part of the program (formula) that caused the failure). To help solve this reliability problem, we have been working on how to improve the reliability of end-user programs in general and of spreadsheets in particular. One of the most widely used real world end-user programming paradigms is the spreadsheet. Spreadsheet tasks range from simple scratch pad calculations to more complex and important personal or business related tasks, such as calculating income tax, financial forecasting and making government and business policy decisions. Managers often use spreadsheets for modeling and decision support and for many of them it is their only decision support tool [Cragg and King 1993].

A major reason for the popularity of the spreadsheet paradigm is that spreadsheets present a simple highly visual environment for organizing and formatting data and for automating a variety of computational tasks. The spreadsheets give end users more direct control over computational resources. In spreadsheet languages the required primitives are already at hand and the end users need not know everything to do their task [Nardi and Miller 1991].One other reason for spreadsheet popularity is because the dependencies between elements of the spreadsheet are managed by the language and the effects are local and easy to trace [Nardi and Miller 1991].

1.2 Problems relating to errors in spreadsheets

Many spreadsheets are created by end users, people with little or no programming experience. Therefore, despite the perceived simplicity of the spreadsheet paradigm it is not surprising that spreadsheets have an alarming number of faults [Panko 1998]. Surveys about spreadsheets audits and experiments provide evidence about this alarming frequency of errors in the spreadsheets. For example, two large auditing firms reported finding errors in 90% of the spreadsheet models they audited; in 4 field audits of operational spreadsheets, errors were found in 20.6% of the spreadsheets audited; in 11 experiments in which the participants created spreadsheets, errors were found in 60.8% of the spreadsheets; in 4 experiments in which the participants inspected for errors, the participants missed an average of 55.8% of the errors. Further more the effects of these errors in the real world may be costly. A Dallas oil and gas company lost million dollars in an acquisition deal because of errors in their spreadsheet financial model [Panko 1998].

Compounding this problem of errors in spreadsheets is the unwarranted confidence expressed by the spreadsheet developers in the correctness of their spreadsheets. An experiment conducted by Brown and Gould [Brown and Gould 1987], reported that experienced spreadsheet developers were quite confident that

their spreadsheets did not contain errors, yet 44% of the spreadsheets they created contained errors.

1.3 End-User Software Engineering

End users differ from professional programmers with respect to their motivation, background, interests and programming experience. They view software applications as a tool to help them solve their problems and regard computers "as a means to an end rather than objects of intrinsic interest "[Nardi and Miller 1991]. End-user programmers are writing an unprecedented number of programs, due in large part to the significant effort put forth to bring programming power to end users. Unfortunately, this effort has not been supplemented by a comparable effort to help them increase the correctness of these often-faulty programs. There has been considerable software engineering research to help reduce errors made by professional programmers but the end users have been ignored. To remedy this situation we have been working on a vision we call "End-User Software Engineering". Our goal is to bring the benefits of software engineering research and methodologies to end users without requiring them to learn the underlying software engineering theory and techniques.

As a part of this work, previously a testing methodology was devised known as "What You See Is What You Test" [Rothermel et al 1998]. The WYSIWYT methodology provides the "testedness" information of each spreadsheet cell and the entire spreadsheet via incremental visualization devices such as cell border colors and a testedness indicator. In This methodology as the spreadsheet is used incrementally, users apply test inputs and validate outputs. This information is used to provide visual feedback about the effectiveness of their testing.

Given this visualization-based support for testing, it is natural to consider providing help to end users with fault localization once their test reveals a failure. We are working to integrate WYSIWYT with visual fault localization techniques

in an effort to explicitly support debugging by end users. As a part of this work we have developed fault localization techniques, which highlight in varying shades of red, the cells that might have contributed to an incorrect value, the goal being that the most faulty cell will be colored the darkest. These techniques do not require the entire WYSIWYT methodology. The minimum requirement for these techniques is any spreadsheet language that allows user to validate outputs of test inputs.

1.4 The Problem Addressed by this Thesis

In this thesis, we consider how visual fault localization techniques affect and interact with end-user programmers' debugging efforts. To explore this issue, we conducted a think-aloud study to investigate the following research questions:

RQ1: How much perceived value do end users see in the interactive fault localization feedback over time?

RQ2: How thoroughly do end users understand the interactive fault localization feedback?

RQ3: What debugging strategies do end users use to find faults?

RQ4: How does fault localization feedback influence an end user's interactive debugging strategy?

RQ5: How do wrong testing decisions affect fault localization feedback?

This thesis reports the results of our study.

2. Related Work

There has been a variety of research work in software visualization for software engineering purposes. In this chapter we focus on the research work related to debugging and fault localization.

Most fault localization research has been mostly based on slicing and dicing techniques for imperative programs. We briefly review those techniques here in turn.

2.1 Slicing and Dicing

What is the motivation for program slicing? There are times when only a portion of a program is of interest. For example, during debugging when the programmer identifies a failure and wishes to track failure to a fault in the code, the programmer starts from the failure and proceeds to find the corresponding portions of program code which might be faulty. Starting from a subset of program's behavior, slicing reduces that program to a minimal form that still produces the same behavior. Program slicing was introduced by Weiser [Weiser 1984] as a technique for analyzing program dependencies. He defined it as follows:

"Program slicing is a method for automatically decomposing programs by analyzing their dataflow and control flow. Starting from a subset of program's behavior, slicing reduces that program to a minimal form which still produces that behavior. The reduced program, called a 'slice', is an independent program guaranteed to represent faithfully the original program within the domain of the specified subset of behavior."

A program slice is defined with respect to a slicing criterion (s, v) in which s is a program point and v is a subset of program variables. A slice consists of a subset of program statements that affect, or are affected by, the values of variables in v at s [Weiser 1984]. Backward slicing finds all the statements that affect a given variable at a given statement, Weiser's slicing algorithm calculates static

slices, based solely on information contained in source code, by iteratively solving dataflow equations. These slices can be calculated using entirely static information or can be more precisely calculated using dynamic information. Korel and Laski [Korel and Laski 1990] introduced dynamic slicing, in which information gathered during program execution is also used to compute slices. Whereas static slices find statements that may affect (or may be affected by) a given variable at a given point, dynamic slices find statements that may affect (or may be affected by) a given variable at a given point under a given execution. Dynamic slicing usually produces smaller slices than static slicing. Dynamic slices are calculated iteratively in [Korel and Laski 1990]. An extensive survey of fault localization techniques based on slicing is given in [Tip 1995].

Program dicing was introduced by Lyle and Weiser [Lyle and Weiser 1987] as a fault localization technique for further reducing the number of statements that need to be examined to find faults. Whereas a slice makes use only of information on incorrect variables at failure points, a dice also makes use of information on correct variables, by subtracting the slices on correct variables away from the slice on the incorrect variable. The result is smaller than the slice on the incorrect variable; however, unlike slice, a dice may not always contain the fault that led to a failure.

Lyle and Weiser describe the cases in which a dice on an incorrect variable not caused by an omitted statement is guaranteed to contain the fault responsible for the incorrect value in the following theorem [Lyle and Weiser 1987]:

Dicing Theorem: A dice on an incorrect variable contains a fault (except for cases where the incorrect value is caused by omission of a statement) if all of the following assumptions hold:

1. Testing has been reliable and all incorrectly computed variables have been identified.
2. If the computation of a variable v depends on the computation of another variable w, then whenever w has an incorrect value then v does also.

3. There is exactly one fault in the program.

In this theorem, the first assumption eliminates the case where an incorrect variable is misidentified as a correct variable. The second assumption removes the case where a variable is correct despite depending on an incorrect variable (e.g. when a subsequent computation happens to compensate for an earlier incorrect computation, for certain inputs.) The third assumption removes the case where two faults counteract each other and result in an accidentally correct value. Given the assumptions required for the Dicing Theorem to hold, it is clear that dicing is an imperfect technique in practice. Thus, Chen and Cheung [Chen and Cheung 1997] explore strategies for minimizing the chance that dicing will fail to expose a fault that could have produced a particular failure, including the use of dynamic rather than static slicing.

2.2 Fault Localization Techniques for Professional programmers

Agarwal et al presented a technique for locating faults in traditional programming languages using execution traces from tests. This technique is based on displaying dices of program relative to one failing test and a set of passing tests. This technique was implemented in the xSlice tool [Telcordia Technologies 1998]. Jones et al have developed a similar system called Tarantula [Jones et al 2002]. Tarantula differs from xSlice in the sense that, it uses information from all passing and failing tests when highlighting possible location of faults. It colors the likelihood that statements are faulty according to the ratio of failing tests to passing tests that the statement was executed. The primary focus of both these techniques is aiding professional programmers find faults in programs developed in tradition programming languages. Our work differs in that our primary focus is to help end users in finding faults in the spreadsheets they develop. Our methods are interactive and incremental.

Zstep [Lieberman and Fry 1998], a program debugging environment designed to help the programmer understand the correspondence between static

program code and dynamic program execution provides visualization of the correspondences between static program code and dynamic program execution.

2.3 Work aimed at aiding end-user programmers

There are several interactive visual approaches to aid spreadsheet comprehension for debugging purposes. S2 visualization provides a visual auditing feature in Excel 7.0: similar groups of cells are recognized and shaded based upon formula similarity, and are then connected with arrows to show dataflow [Sajanieme 2000]. This technique builds upon the Arrow Tool, a dataflow visualization device proposed by Davis in [Davis 1996].

Work aimed particularly at aiding end-user programmers with debugging and other software engineering tasks is beginning to emerge. Myers and Ko recently proposed research in assisting users in the construction and debugging of code for event-based languages [Myers and Ko 2003]. Carr proposes reMIND+ [Carr 2003], a visual end-user programming language with support for reusable code and type checking. reMIND+ also provides a hierarchical flow diagram for increased program understanding. Finally, the assertions approach in Forms/3 has been shown empirically to help end-user programmer's correct errors in spreadsheets [M. Burnett et al 2003].

3. Background

In this chapter we discuss Forms/3 (a continuously evolving research prototype for end-user programming research), and the WYSIWYT methodology and the testedness feedback it provides. We also describe the fault localization techniques devised to help end users debug their spreadsheets.

3.1 Forms/3

Forms/3 is declarative, spreadsheet based visual programming language. Forms/3 not only supports features found in commercial spreadsheets but also advanced language features found in research spreadsheet languages. A continually evolving prototype of end-user software engineering concepts exists for Forms/3. We choose Forms/3 for our think aloud study because we have access to the source code and can tailor it to different experimental conditions.

In Forms/3, as in other spreadsheet languages, spreadsheets are a collection of cells and each cell's value is defined by the cell's formula. A user receives feedback about a cell's value immediately after the cell formula is entered. The user can create cells, delete cells, display, enter, and edit values and formulas in the cells. Underlying the user interface, there is an evaluation engine that propagates values and evaluates formulas. Some noticeable differences between Forms/3 and commercially available spreadsheets are: cells in a Forms/3 spreadsheet are free floating as opposed to a fixed grid pattern, cells in Forms/3 spreadsheet can be given meaningful names and several cell formulas can be displayed at the same time. Figure 1 shows a Forms/3 spreadsheet that computes the total score of a student based on 3 quizzes, a midterm, a final and an extra credit score. Input cells are cells that do not contain a formula while output cells contain formulas that reference other input and output cells. In Figure 1 quiz1, quiz2, quiz3, Midterm, Final and ExtraCredit are input cells and the remaining cells are output cells.

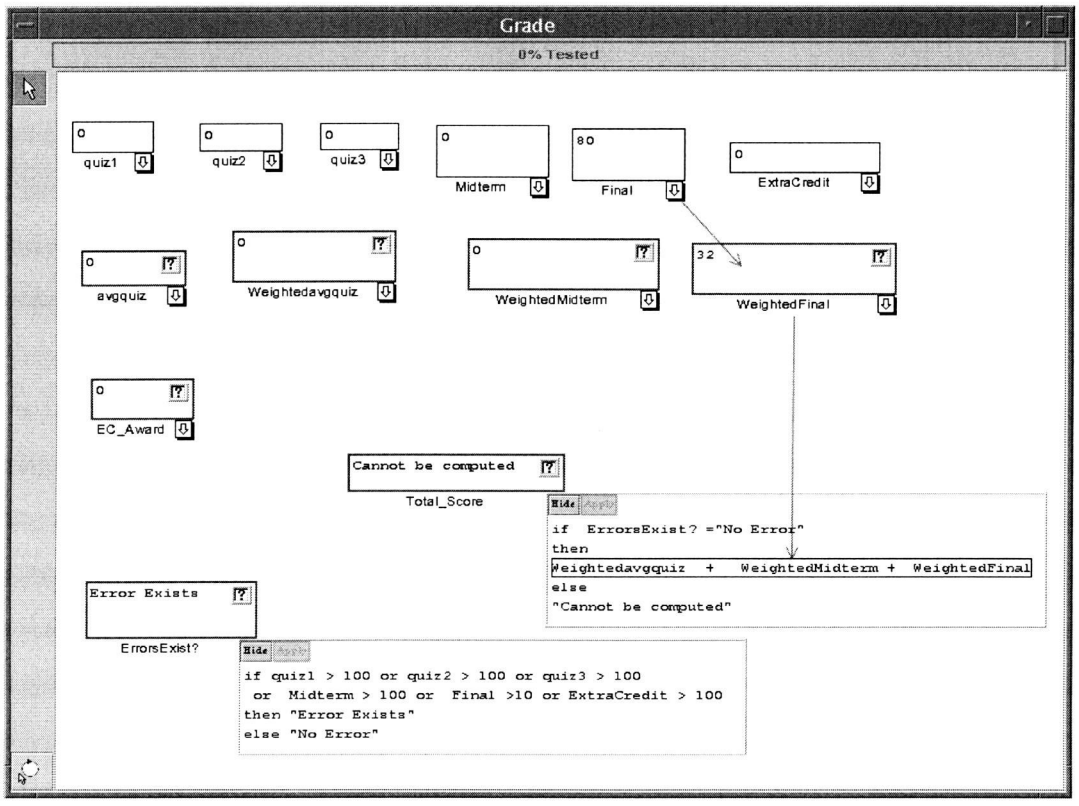


Figure 1: A grades spreadsheet. Computes a student's score based on quizzes, midterm, final and extra credit. Note that the formulas for several output cells are displayed.

Notice that in figure 1 the output cells have colored borders (black and white borders in this paper), there is a testedness indicator at the top that reads 0% and certain cells have question marks at the upper right corner of the cells. The arrows in the figure show the dependency relationship between the cells. All these elements are a part of the visual feedback of the WYSIWYT testing methodology which will be discussed in the next section.

3.2 WYSIWYT

The WYSIWYT methodology that is integrated with the Forms/3 environment provides detailed visual feedback about the testedness of the individual cells and the entire spreadsheet. The WYSIWYT methodology is

designed to accommodate the declarative evaluation model of the spreadsheet paradigm, the incremental way of development of spreadsheets, and the immediate visual feedback of spreadsheet languages.

The basic WYSIWYT methodology is comprised of four visual feedback mechanisms: cell border colors, dataflow arrows to indicate cell dependencies, a decision box for each cell for users to record testing decisions, and a percent tested indicator. An output cell's border color indicates the testedness of the cell, red indicates untested, blue indicates completely tested, and purple indicates partially tested. The dataflow arrows show the interrelationships between different cells of the spreadsheet. These arrows also indicate the degree of testedness of the cell dependencies (interrelationships between different cells) and they follow the same color scheme as that of the cell borders. The users can choose to display the arrows at the granularity of the cells or the subexpressions within the cells. The decision box in the upper right corner of each cell gives the information about the validation state of the cell based on the current input. The user records his testing decision about the cell value based on the current input by either clicking the left button (indicating that the cell value is correct) or the right button of the mouse (indicating the cell value is incorrect) in the cell's decision box. A question mark in the cell's decision box indicates to the user that validating (accomplished by left clicking) the cell value will increase the degree of testedness of the spreadsheet, a blank indicates that validating will not increase the degree of testedness, a check mark(after the user clicks the left mouse button) indicates that the user has decided that the current cell value is correct for the current input value, and a X-mark (after the user clicks the right mouse button) indicates that the user has decided that the current cell value is incorrect. Finally the percent tested indicator gives the percentage of total number of cell dependencies in the program that have been covered by the user's validations.

We now present a brief scenario illustrating how the visual feedback of the WYSIWYT methodology helps the user in testing their spreadsheet.

A Forms/3 spreadsheet is tested by trying different test cases (input values) and validating or invalidating (placing check marks or X-marks) the output cells for these test cases. Testing is pronouncing that a cell value is correct or incorrect for the current set of inputs, based on spreadsheet descriptions.

In the grades spreadsheet, there are two testing situations to be tested in the TotalScore cell:

S1: when the TotalScore cell displays "cannot be computed" and

S2: when the TotalScore cell displays the sum of Weightedavgquiz, Weightedmidterm, weightedfinal and EC_Award.

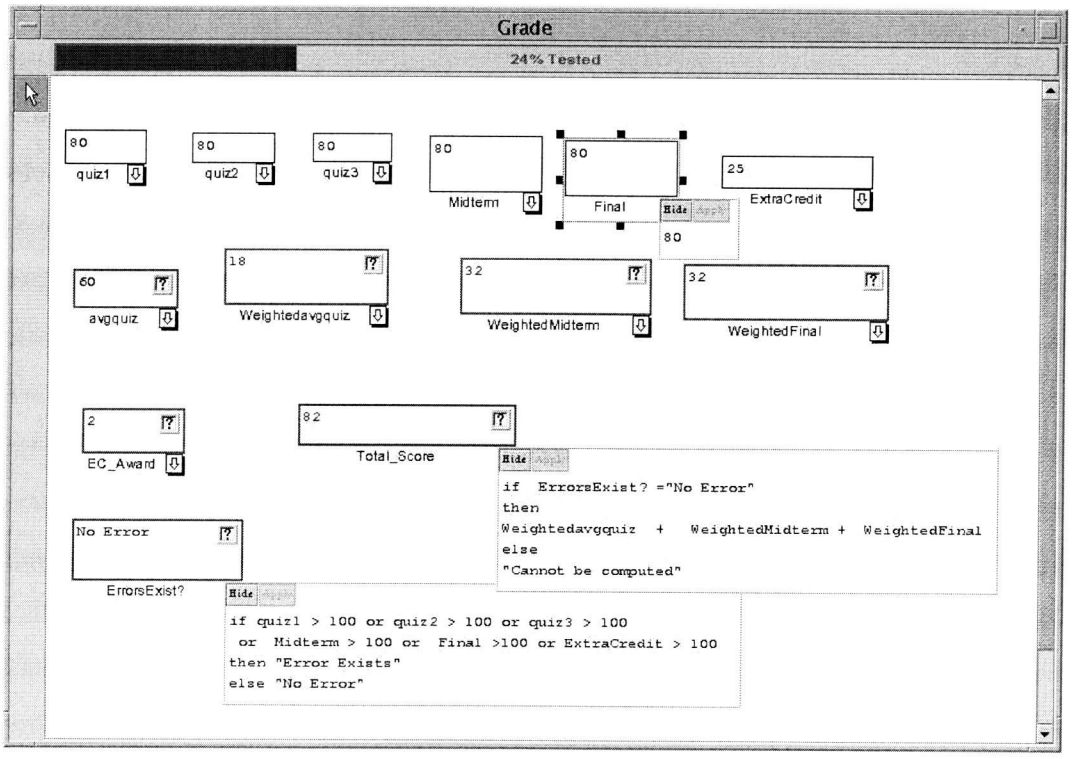


Figure 2: Grades spreadsheet after a testing decision has been made.

Figure 2 shows the grades spreadsheet in Figure1 after a decision has already been made for the situation S1 and inputs have been changed to cover the situation S2. The new testedness information of each cell and the entire spreadsheet is now displayed as in Figure 2. Notice that the borders of the

TotalScore and ErrorExists cells turn purple (shades of gray in this paper) indicating that these cells are partially tested. The testedness indicator has risen to 24%. No testing decision has yet been made for the new situation (because of changed inputs) as can be seen from the question mark in the for TotalScore's decision box.

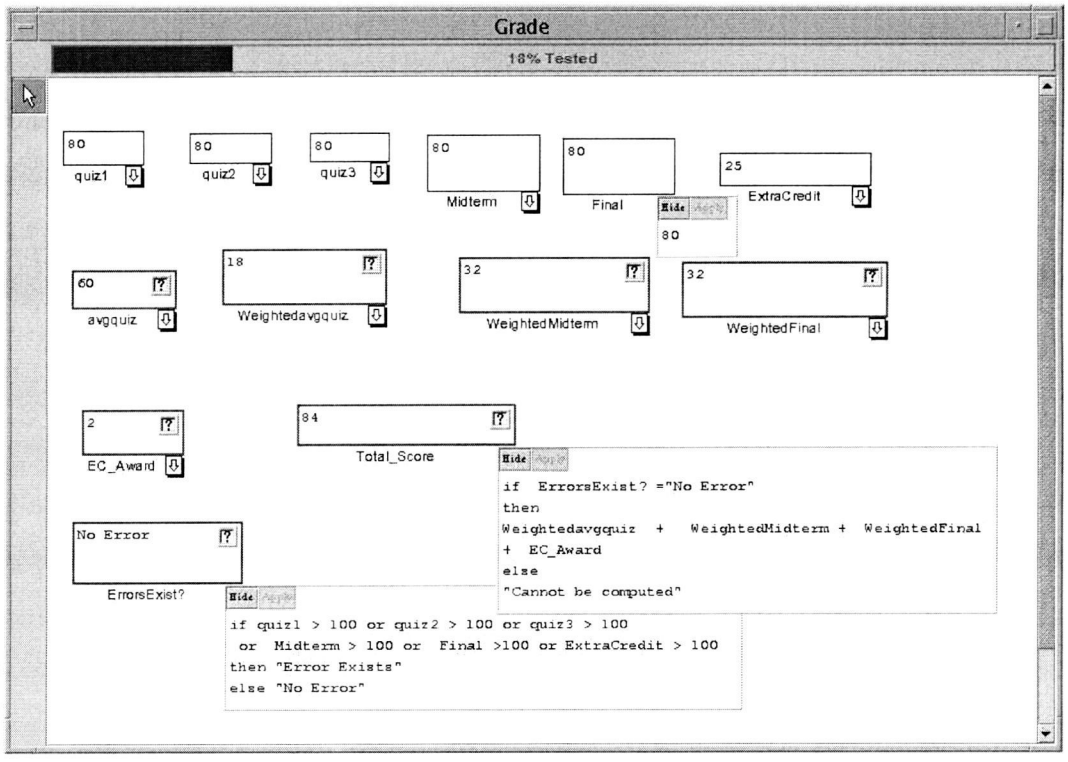


Figure 3: Grades spreadsheet after the error in the cell TotalScore is corrected.

Figure 3 shows the grades spreadsheet after it has been modified to correct the formula in the TotalScore cell by adding EC_Award to the formula in the TotalScore cell. When the formula in the cell is changed, the cell loses its own testedness information as well as of all the cells that are downstream from that cell in dataflow and depend in its value. This is because WYSIWYT assumes that cells that were pronounced as correct were only valid for those formulas in the cells at the time the cell was validated. Thus when a formula is changed in a cell, the cell

along with the cells downstream in the dataflow turn red (lighter in this paper) and the percent tested indicator drops (now 18%) to reflect this.

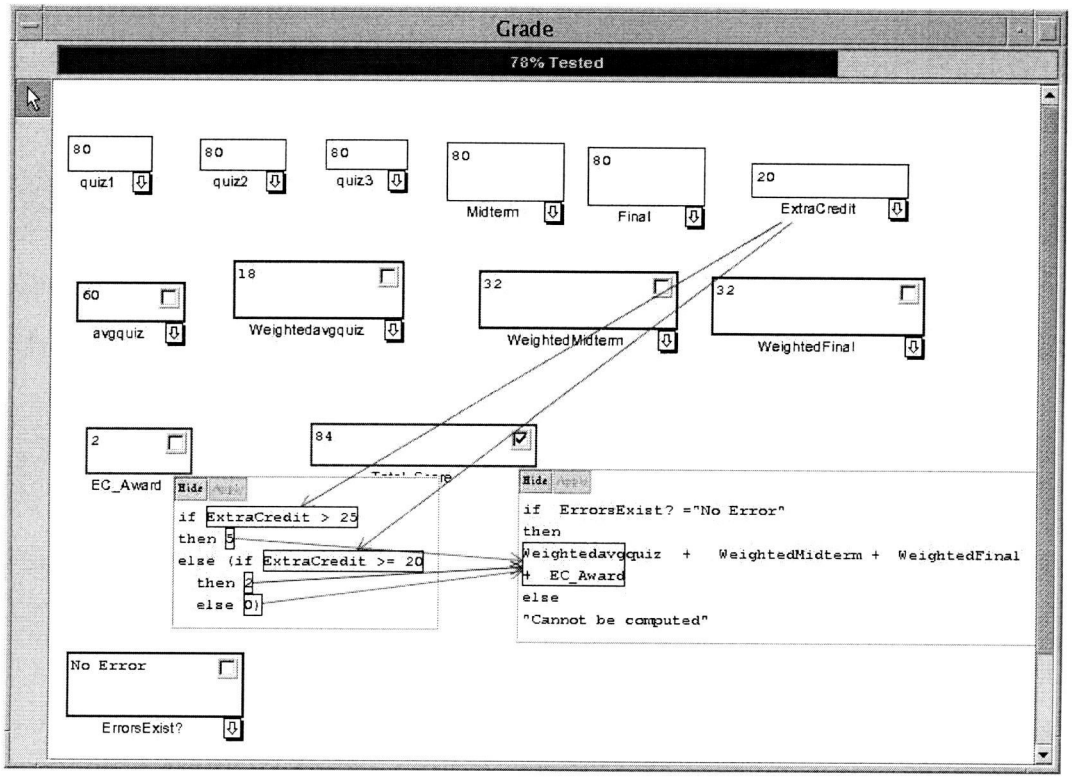


Figure 4: The grades spreadsheet tested furthermore and also depicting some untested partially tested relationships

Figure 4 shows the spreadsheet after the user validates the TotalScore cell for the current set of inputs, which cover situation S2. Notice that the question mark in the TotalScore cell now becomes a checkmark, most of the cells in the spreadsheet are now blue (more dark) and the %testedness indicator has risen to 78%. The cells EC_Award and TotalScore still have purple borders (shades of gray in this paper) and the purple and red arrows tell the user which relationships are still to be tested.

Although the user may not realize it, behind the scenes "testedness" is computed using a dataflow test adequacy criterion [Rothermal et al 1998]. A test adequacy criterion is necessary in a testing methodology to define a point when the

user has done "enough" testing. WYSIWYT's adequacy criterion is du-adequacy. This criterion focuses on the links between a cell that defines a value and a cell that use that value. Such a link is called a definition-use association or a du-association. Data flow analysis identifies these interactions and classifies them as definitions or uses depending on whether the cell defines a value or uses the values defined in other cells.

To test the interaction between cells in Forms/3, the flow of data between definition and use is traced. Two types of definition-use associations are identified: definition-c use association, where an already defined cell is used in the computation of the value of a cell and definition-p use association, where an already defined cell is used in evaluation of a predicate expression in a cell [Rothermel et al 1998]. A definition-use-association (du-association) links a definition of a cell with a use of that cell which the definition can reach. To define to measure the testedness information, we measure the number of du-associations that have been covered or executed by at least one test. A red cell border indicates that none of the du-associations have been tested for that cell, blue indicates that all the du-associations have been tested and purple indicates that some of the du-associations have been tested and others require testing. The shade of purple represents the proportion of du-associations that have to be covered.

3.3 Fault Localization Technique

In the related work, we briefly reviewed techniques of program slicing and dicing. We should mention here that our fault localization is based on techniques of program slicing and dicing for imperative programs.

During the course of a spreadsheet development, users will experience failures in their spreadsheets: cases where cell outputs are incorrect. The interactive and incremental manner in which spreadsheets are created suggests that on discovering such failures, users may immediately attempt to debug the faults that cause those failures. Given the incremental, visualization-based support for

testing it is natural to consider aiding the users with fault localization once one of their tests reveals a failure. Recall that, while testing the spreadsheet, a user can indicate an observation of a failure by marking a cell incorrect with an "X" instead of checkmark that is used to mark the cell correct. This can be done by clicking the right button of the mouse in the decision box in the upper right corner of the cell. At this point, our fault localization techniques highlight in varying shades of red the interior of the cells that might have contributed to the failure, with the goal being that the most faulty cell will be colored the darkest. How should these colors be computed? Computing exact fault likelihood values for a cell, of course, is not possible. Instead, we must combine heuristics with deductions that can be drawn from analyzing the source code (formulas) and/or from the user's tests.

Other members of our research group have developed an integrated and incremental testing and debugging methodology that uses a fault localization technique similar to dicing called the "Blocking Technique". Our strategy of computing the best value for the fault likelihood of a cell is to maintain five properties described below. We will use producer-consumer terminology to keep dataflow relationships clear; that is, a producer of C contributes to C's value, and a consumer of C uses C's value. In slicing terms, producers are all the cells in C's backward slice, and consumers are all the cells in C's forward slice.

Property 1: If cell C or any of its consumers have a failed test, then C will have non-zero fault likelihood.

This first property ensures that every cell that might have contributed to the computation of an incorrect value will be assigned some positive fault likelihood.

Property 2: The fault likelihood of C is proportional to the number of C's failed tests.

This property is based on the assumption that the more incorrect calculations a cell contributes to, the more likely it is that the cell contains a fault.

Property 3: The fault likelihood of C is inversely proportional to the number of C's successful tests.

The third property, in contrast to Property 2, assumes that the more correct calculations a cell contributes to, the less likely it is that the cell contains a fault.

Property 4: An X mark on C blocks the effects of any checkmarks on C's consumers (forward slice) from propagating to C's producers (backward slice).

This property is specifically to enhance localization. Producers that contribute only to incorrect values are darker, even if those incorrect values contribute to correct values further downstream, preventing dilution of the cells' colors that lead only to X marks.

Property 5: A checkmark on C blocks the effects of any X marks on C's consumers (forward slice) from propagating to C's producers (backward slice), with the exception of the minimal fault likelihood property required by Property 1.

Similar to Property 4, this property uses checkmarks to prune off C's producers from the highlighted area if they contribute to only correct values, even if those values eventually contribute to incorrect values.

Property 6: A correct mark on C blocks the effects of any incorrect marks on cells in Dynamic Forward- Slice(C), preventing propagation of the incorrect marks' effects to the fault likelihood of cells in Dynamic- BackwardSlice(C), except for the minimal fault likelihood required by Property 1.

This property is relevant when a value marked incorrect depends on a value marked correct.

Implementation details of the above five properties are given in [Reichwein et al. 1999, Ruthruff et al. 2003].

As a starting point, it was decided to divide the fault likelihood into four distinct ranges: "low", "medium", "high", "very high". We will term the cells to which C refers (directly or indirectly) C's producers and the cells referring to C (directly or indirectly) C's consumers. C's fault likelihood is none if none of its consumers have X-marks. C is given a Low fault likelihood if all X-marks in its consumers are blocked from C by checkmarks on the dataflow path between the X-mark and C. Otherwise, there are X-marks in C's consumers that reach C (are

not blocked by checkmarks), and C's fault likelihood is estimated using the equation below, and then mapped to a colorization using the scheme in Table 1.

$$FL(C) = \max(1, 2 * ReachingXMarks - ReachingCheckmarks)$$

Intensity of Color	fault likelihood(C)
Low	1-2
Medium	3-4
High	5-9
Very High	10+

Table 1: Mapping fault likelihood calculations to color intensities

We should mention that, two other fault localization techniques called the "Test Count" technique and the "Nearest Consumers" technique were devised. These techniques are described in [Ruthruff et al 2003]. Even though three fault localization techniques were devised, in this study we used the blocking technique since an informative evaluation of the three techniques in [Ruthruff et al 2003] revealed that the blocking technique was the most consistent in its visual feedback and was as robust as the test count technique.

We now present a brief scenario illustrating how the visual feedback of the fault localization technique help the user to locate a fault once their testing reveals a failure in the spreadsheet.

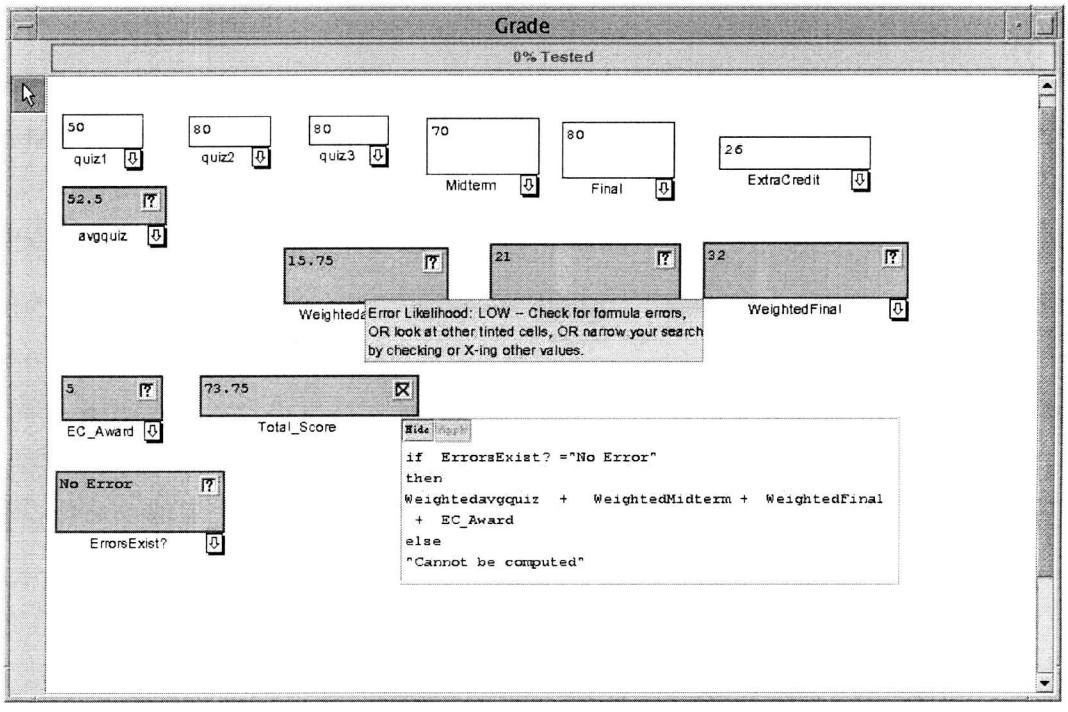


Figure 5: The grades spreadsheet after the user places an X-mark, once a failure is spotted. The tool tip over the cell weightedavgquiz informs the low fault likelihood ness of that cell.

Figure 5 shows the grades spreadsheet after a decision has been made by the user that the value in the cell TotalScore is incorrect for the current input values, by placing an X-mark in the decision box of the TotalScore cell. Using this information and the dataflow relationships, the Blocking technique has assigned an estimated fault likelihood of "low" to all those cells that were contributing to the value of the TotalScore cell. The user inspects the formula of the TotalScore cell and the formula seems correct.

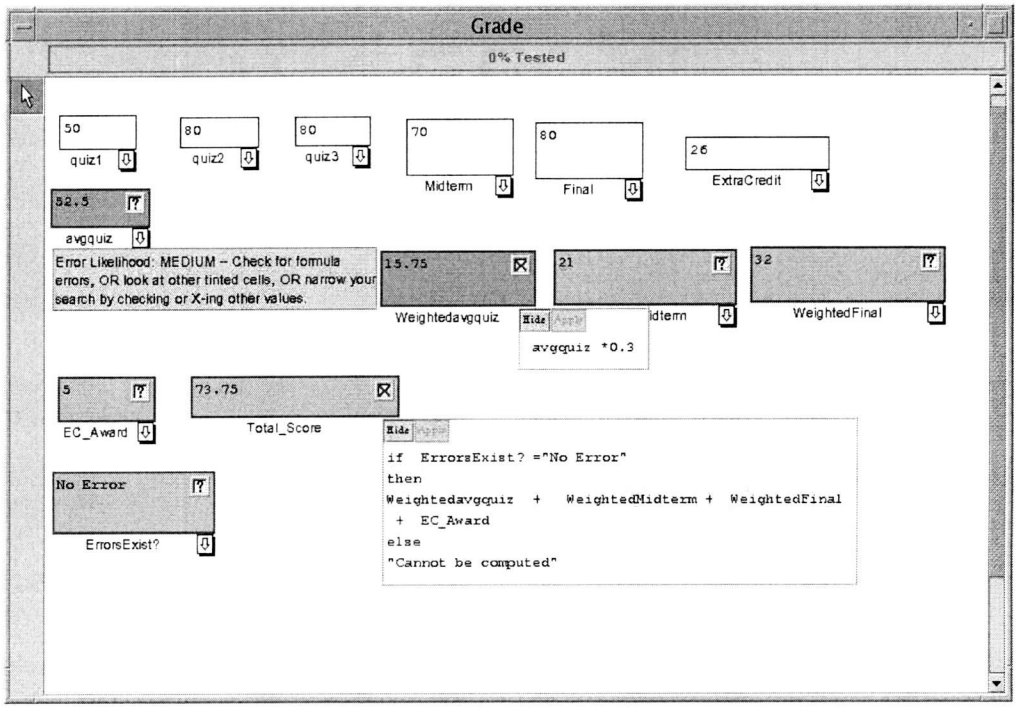


Figure 6: The grades spreadsheet after the user places a 2nd X-mark on cell weightedavgquiz.

Figure 6 shows the grades spreadsheet after the user makes another decision, by placing an X-mark in the decision box of the Weightedavgquiz cell, indicating that the cell value is also incorrect. Using this information, the Blocking technique has now assigned an estimated fault likelihood of "Medium" to the avgquiz and the Weightedavgquiz cells thus narrowing the search space. The user inspects the formula of the cell Weightedavgquiz and the formula seems correct.

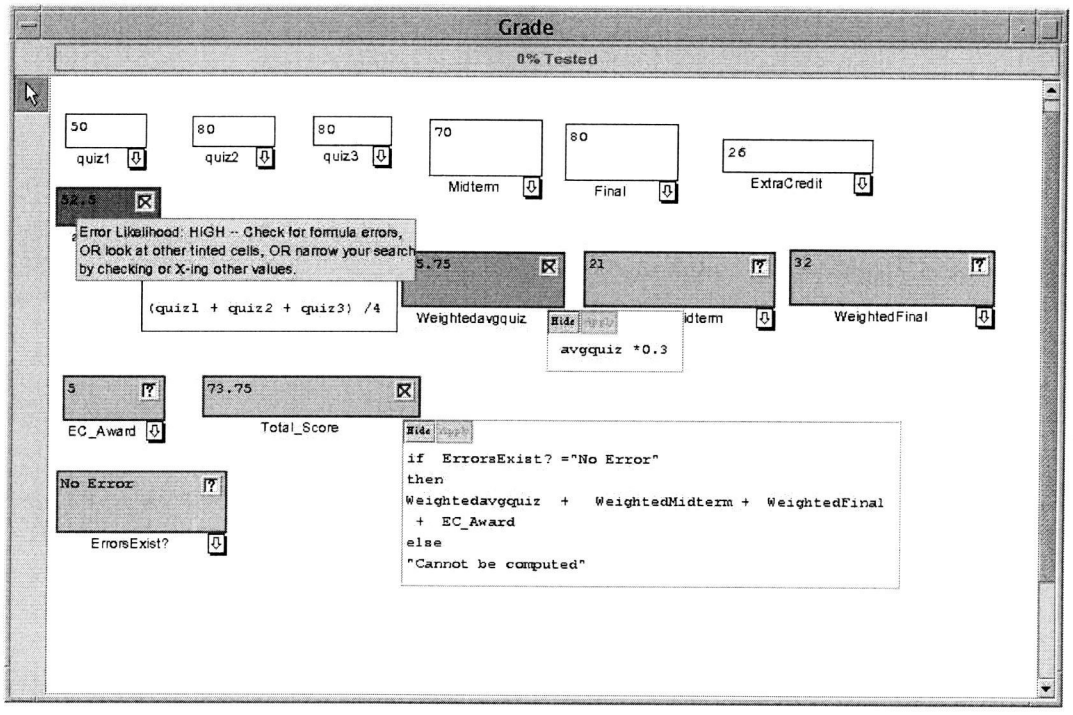


Figure 7: The grades spreadsheet after the user places his third X-mark thus isolating the fault in the avgquiz cell.

Figure 7 shows the grades spreadsheet when the user places an x-mark in the decision box of the avgquiz cell, indicating a wrong cell value. The estimated fault likelihood assigned to the avgquiz cell is now "high" thus isolating the fault to the cell avgquiz.

4. Experiment Design

We are working to tightly integrate WYSIWYT with visual fault localization techniques in an effort to explicitly support debugging by end users [Reichwein et al 1999 and Ruthruff et al 2003].

To obtain the qualitative information necessary to investigate the research questions enumerated in the introduction, we conducted a think-aloud study, using ten end users as subjects. A think-aloud study allows subjects to verbalize the reasoning for their actions. Traditional controlled experiments based on statistical methods provide quantitative information but do not provide the qualitative information we sought for this work. For example, a traditional experiment that counts the number of faults corrected cannot explain user behaviors or reactions to fault localization feedback, or provide insights into the cognitive thought process of a user; rather, such experiments provide only indirect clues about the human-centric issues we sought to investigate. Thinking-out-loud is considered as a tool for collecting systematic observations about the thinking that occurs while working on the task, that is, for collecting data about the otherwise unseen, unobservable processes. Our interactive and incremental approach to end-user debugging demands our interest into the "why" of end-user behavior just as much as the "what".

4.1 Procedure

Our subjects were divided into two groups: a control group having only the features of WYSIWYT and a treatment group having both WYSIWYT and the fault localization "Blocking" technique. A control group was needed for the debugging strategy comparisons of RQ3. Each session was conducted one-on-one between an examiner and the subject. The subject was given training on thinking aloud and a brief tutorial on the environment they would be working in. The

tutorial consisted of hands-on instruction on the basic features of WYSIWYT (and fault localization for the treatment group), followed by a practice task.

After familiarizing themselves with their environment, each subject worked on the tasks of debugging and testing a spreadsheet to ensure that the spreadsheet worked according to the description given. The subjects performed the task independently, thinking aloud as they carried out the task. After completing the task, they were asked to fill out a post-session questionnaire and answer oral questions about their actions during the experiment

The data collected for each session included audio transcripts, electronic transcripts capturing all user interactions with the system, post-session written questionnaires, and the examiner's observations.Audio transcripts captured the subjects' verbalizations as they performed the given tasks. Electronic transcripts captured user actions such as editing the values in a cell, placing a √ or X-mark in a decision box, and turning on/off arrows indicating dataflow relationships between cells. Post-session questionnaires asked about their use of the WYSIWYT features, the usefulness of these features in finding and fixing errors; treatment subjects answered questions that tested their understanding of the fault localization technique. In addition, the examiner took notes of his observations during the session.

4.2 Subjects

We selected students from the College of Business for our subjects since they seemed most representative of actual spreadsheet end users. Ten subjects were equally divided into two groups. One group (control) had only features WYSIWYT and the other group (treatment) had both WYSIWYT and the and the fault localization "Blocking" technique. We selected and distributed subjects based on their experience with spreadsheets and their GPA so that there was an even distribution of experienced and less experienced subjects in both the groups (refer to Table 2). The information about their experience and GPA was gathered

via a background questionnaire that the participants filled out prior to the experiment (see Appendix B for the background questionnaire).

	GPA	Programming Experience (yrs)	Spreadsheet Experience(yrs)
Control	3.1	none	All
Treatment	3.2	none	All

Table 2 : Background information of the subjects.

4.3 Tutorial

Before beginning the experimental tasks, the subjects were given practice doing think aloud and a self-paced tutorial on the environment they would use. Since it was essential for the subjects to talk aloud, the subjects did two "thinking aloud" practice problems: adding the numbers "678" and "789" and counting the number of windows in their parent's house. The tutorial in Forms/3 and WYSIWYT was designed such that the subject received sufficient practice with the editing, testing and debugging a cell. The feedback given by WYSIWYT was explained. The tutorial also explained to the subjects how to seek explanations about various objects by seeking tool tips using mouse over. The tutorial for both the groups was same except that the feedback of the fault localization technique was explained to the treatment group by using it to debug a cell. To counterbalance this material, the control group did the same debugging activity without any fault localization feedback.

For the tutorial, the subject performed tasks on an actual spreadsheet with guidance at each step. Subjects were free to ask questions or seek clarifications during the tutorial. The tutorial ended when the subject was judged to have learnt the features of Forms/3. At the end of the tutorial all subjects were given 2 minutes to explore the spreadsheet they were working during the tutorial to help them better understand the features taught in the tutorial. As a final tutorial task, and to prepare them for the experimental tasks, all subjects were given 5 minutes to work on a debugging a different spreadsheet.

4.4 Tasks and Materials

Subjects in both the groups were given the same experimental task: test and debug two spreadsheets and ensure that the spreadsheets work according to the given description.

Allwood classified faults in spreadsheets as mechanical, logical and omission [Allwood 1984], and this scheme is also used in Panko's work [Panko 1998]. Under Allwood's categorization, mechanical faults include simple typographical errors or wrong cell references in the cell formulas. Mistakes in reasoning were classified as logical faults. Logical faults in spreadsheets are more difficult than mechanical faults to detect and correct, and omission faults are the most difficult [Allwood 1984]. An omission fault is information that has never been entered into the formula.

We drew from this research by including faults from each category in each problem. However, the precise distinctions between logical and mechanical are not clear for some types of faults in end-user programming. For example, when computing an average, does dividing by the wrong number mean the subject typed it in wrong, or that they are confused about computing averages? In our previous think-aloud studies we have collected data in which end-user subjects made exactly this error for both of these reasons. Thus, we combined the first two categories into one and then, to be sure coverage of both would be achieved, included several different subtypes under it: incorrect references (which Allwood would classify as mechanical), incorrect constants or an omitted character (could be either logical or mechanical), incorrect operators or application of operators (which Allwood would classify as logical), and extra subexpression (logical). We also included faults from the third category, omission faults.

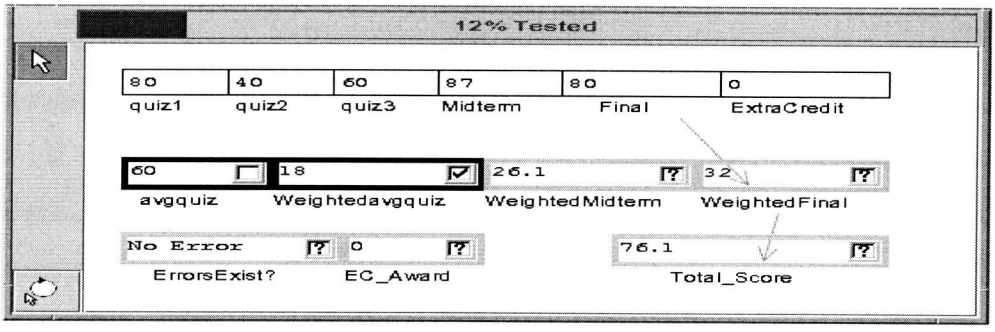


Figure 8: The Grades spreadsheet.

Drawing from this research, we seeded five faults of varying difficulty into each of two Forms/3 spreadsheets. One of these spreadsheets is the Grades spreadsheet from Figure 1, which computes the total score for a course given input for three quizzes, extra credit, a midterm, and a final exam. There is also an output cell that indicates when an input cell is outside the valid range. Grades has three mechanical faults, one logical fault, and one omission fault. This spreadsheet was deemed the "easier" of our two tasks based on its size and the complexity of its formulas and from our pilot studies.

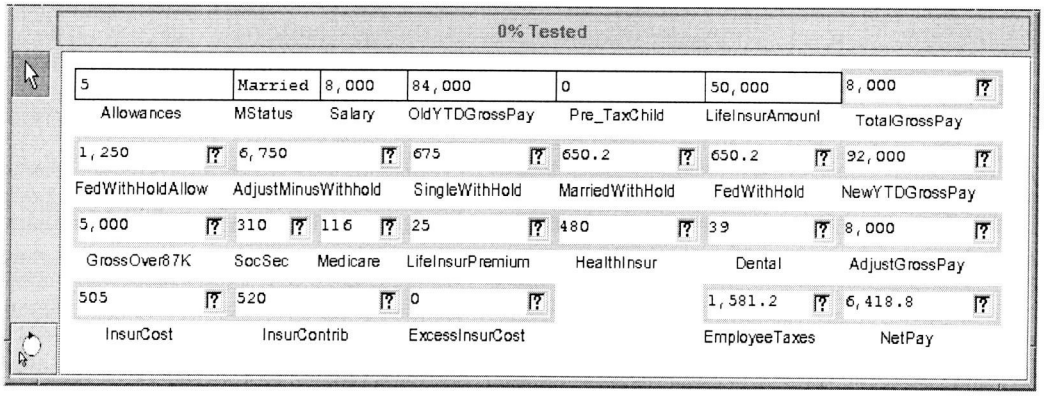


Figure 9: The Payroll spreadsheet.

The other spreadsheet, Payroll, is presented in Figure 2. In this spreadsheet, we seeded one mechanical fault, three logical faults, and one omission fault. This

spreadsheet was much larger, had more levels of data flow, and had a greater number of output cells in relation to input cells when compared to the Grades spreadsheet. In this task, the subjects were told that a spreadsheet that computes the net pay for an employee has been updated. They were given; the unverified spreadsheet, the description of all input and output cells, descriptions of how these values should be derived, and two correct sample pay stubs.

Subjects were given these two spreadsheets (tasks) in varying order, with instructions to test and correct any errors found in the spreadsheets. For each task, the subjects were provided the unverified spreadsheet and a description of the spreadsheet's functionality. Furthermore, the subjects were provided a single example of the expected output values given specified inputs for the Grades task, and two such examples for the Payroll task. Subjects had a time limit of 15 minutes to complete the Grades task and 30 minutes to complete the Payroll task. These times were determined from several pilot studies.

5. Results

As mentioned in Chapter 4, we accumulated data from the following sources: audio transcripts, electronic transcripts, observations, post session questionnaire responses. The audio transcripts captured the users verbalizations of the tasks as they performed them, electronic transcripts captured the users actions like editing values in a cell, placing a checkmark or a x-mark in a cell, turning on/off arrows and seeking explanations, and the post session questionnaire asked about their use of the WYSIWYT features, and their understanding and rating of these features. In addition, the treatment subjects were asked questions after the experiment about their understanding of the fault localization technique. Also observations made during the session were noted. This chapter presents the analysis of the data obtained from these sources for each of the research questions.

5.1 Results of Question 1:

How much perceived value do end users see in the fault localization feedback over time?

Blackwell's model of attention investment [Blackwell 2002] is one model of user problem-solving behavior. It predicts that users will not want to enter an X-mark unless the benefits of doing so are clear to them. The model considers the costs, benefits, and risks users weigh in deciding how to complete a task. For example, if the ultimate goal is to forecast a budget using a spreadsheet, then using a relatively unknown feature such as an X-mark has cost, benefit, and risk. The costs are figuring out when and where to place the X-mark and thinking about the resulting feedback. The benefit of finding faults may not be clear after only one X-mark; in fact, the user may have to expend even more effort (place more X-marks) for benefits to become clear. The risks are that going down this path will be a waste of time or worse, will mislead the user into looking for faults in the correct formulas instead of in the incorrect ones.

First, we consider whether users, having briefly seen X-marks in the tutorial, were willing to enter even one X-mark to help their debugging efforts. The control group had no fault localization feedback and hence they had no benefits of placing an X-mark. Table 3 enumerates the number of times the subjects in both the groups placed an X-mark.

Control group	**Grades**	**Payroll**
CS1	3	0
CS2	0	0
CS3	0	0
CS4	0	1
CS5	0	0
Treatment group	**Grades**	**Payroll**
TS1	3	2
TS2	3	1
TS3	0	0
TS4	0	5
TS5	1	0

Table 3: Number of times X-mark was used in each spreadsheet.

The fact that only two control group subjects used X-marks and they used it 4 times in total, suggested that the subjects were not willing to place the X-mark without any benefits from placing them. In the treatment group, four of the five treatment subjects placed at least one X-mark, especially when they needed assistance debugging a failure (discussed further in Section 5.4). The subject who did not place an X-mark (treatment subject TS3) explained during a post-session interview that she had forgotten about them, and wished she had used them:

TS3: I wish I could redo the problem with the X-mark. If I would have done that, it would have been lot more easier.

In our interactive fault localization system, the first interaction about a failure (X-mark) leads to feedback, and this feedback may or may not provide enough benefits to lead to a second X-mark. In general, a difference in any *interactive* fault localization approach from traditional approaches is that the accuracy of feedback about fault locations must be considered at every step of the

way, especially in early steps, not just at the end of some long batch of tests. As the attention investment model explains, if the early feedback is not seen as providing information that is truly of practical help, there may never be any more interactions with the system! This was exactly the case for subject TS5, who placed only one X-mark in the Grades task, and explained after the session:

TS5: *If it is wrong I am going to make it right...um.. I mean , if I know it is wrong , I am going to work to make it correct ,to me, putting an X-mark just means I have to go back to do more work.*

In his case, the X-mark he placed was in a cell whose only contributors were input cells; consequently, because our technique does not tint input cells (which do not have formulas), the only cell tinted was the cell with the X-mark. Since this feedback did not add to the information he already had, there was no benefit for him from placing an X-mark on that cell. This indicates the importance of the visible feedback (reward), even in the early stages of use; if the reward is not deemed sufficient for further attention, a user may not pursue further use.

However, the other three treatment subjects who placed an X-mark went on to place a second and a third X-mark later in the session. Some of the comments made by the subjects while using the X-mark are:

TS1(thinking aloud while working on the payroll problem):

Um, look on the sheet says social security tax is $372, but this one says $3,348. So we're wrong somewhere... and that looks right. So we right click on that one and see what boxes, show the boxes that could be wrong.

TS4 (thinking aloud while working on the payroll problem):

Moving down the line, federal withholding it says is 610, whereas actual federal withholding is 647, that number's incorrect. So married withholding is incorrect. Hide that for a minute. Right click that cause I know it's incorrect, highlights everything that's possible errors.

It is clear from the usage of the X-marks and verbalization of the treatment subjects that the subjects were willing to place the X-mark when the benefits of

placing X-marks were clear to them. We will see in section 5.3 that the rewards gained by these subjects outweighed their perceived costs of testing and marking failures with X-marks.

5.2 Results of Question 2:

How thoroughly do end users understand the interactive feedback?

To what extent did the subjects understand the information the interactive feedback was trying to communicate? We investigated two levels of understanding: the deeper level being the ability to predict feedback under various circumstances, and the more shallow level of being able to interpret feedback received.

To investigate these two levels of understanding, the post-session questionnaire for our treatment subjects had 11 questions, 6 (questions 4-9), measuring ability to predict behavior and 5 (questions 10-14), measuring ability to interpret feedback about the effects of X-marks on the interior colorings of a cell. Post session questionnaire for both the groups appear in appendix B. Table 4 shows the categorizations of the comprehension questions in the questionnaire. The subjects' ability to *predict* behavior, as measured by 6 questions (questions 4-9), was mixed. Again using producer-consumer terminology, all subjects were able to correctly predict the impacts on producer cells of placing a single X-mark (questions 4 and 5). About half the subjects were able to predict the impacts on consumer cells of placing a single X-mark (question q6) and to predict the impacts when multiple X- and checkmarks were involved (questions 7-9). However, the ability to *interpret* behaviors was uniformly good: all four of the subjects who actually used X-marks during the session were able to explain the meanings of the colorings and marks, and to say what those meanings implied about faults (questions 10-14). For example some responses to questions about what it means when the interior of cells get darker or get lighter were:

TS1: If the color becomes lighter, the cells have less of a chance of to be wrong.

TS2: The chance of an error in the darker cells is greater than in the lighter cells.

TS3: "The likelihood of errors does back to lower than darker cells"

These post-session questionnaire results are corroborated by the actions of the users themselves, as we will discuss in the next two sections.

Question number	Question content	Percent correct
Questions measuring ability to predict behavior		
q4,q5	Ability to predict the impacts on producer cells of placing a single X-mark	100%
q6	Ability to predict the impacts on consumer cells of placing a single X-mark.	60%
q7, q8, q9	Ability to predict the impacts when multiple X- and checkmarks were involved	53%
Questions measuring the ability to interpret feedback		
q10, q11, q12, q13, q14	Ability to interpret feedback about the effects of X-marks on the interior colorings of a cell.	85%

Table 4: Categorizations of the post session questions

5.3 Results of Question 3:

What debugging strategies do end users use to find faults?

Recall that for each task the subject was given a spreadsheet with instructions to test and to correct any errors found. After reading the problem description, the subjects usually entered input values from the provided examples and compared the example output with the output from their spreadsheet. This comparison was done in a top to bottom fashion starting with the first part of the description. At some point, the seeded faults in each spreadsheet resulted in an observed failure in the program. It is at this point, when the subjects switched from a "browsing" to a "debugging" mode, that debugging strategies began to differ.

Because this work is about fault *localization*, we focus on users' abilities to *identify* the location of faults, as defined by either an explicit verbal statement or by the fact that they edited the cell's formula. Once identified, corrections usually followed; 60 of the 65 faults were corrected once identified. Table 5 enumerates the number of faults identified and corrected by each subject for each problem.

	Grades		**Payroll**	
	Identified	**Corrected**	**Identified**	**Corrected**
Control				
Pn3	4	4	3	2
Pn4	2	2	2	1
Pn5	4	3	0	0
Pn6	4	4	2	1
Pn7	5	5	2	2
Treatment				
Pb1	3	3	2	2
Pb2	4	4	1	2
Pb3	5	4	4	3
Pb4	5	5	5	5
Pb5	5	5	3	3

Table 5: The number of faults identified and corrected for each problem.

Once a failure was spotted, users exhibited two kinds of strategies to find the fault causing the failure: an ad hoc strategy, in which they examined cell formulas randomly in no particular order, and a dataflow strategy, in which they followed the failure's dependencies back through cell references until they found the fault. Some subjects first started with an adhoc strategy and later switched to a dataflow strategy. A dataflow strategy can be accomplished through mental effort alone, but subjects rarely did this: mostly they used either arrows, the fault localization feedback, or a combination of both.

Strategies				
	Ad hoc	**Dataflow**		**Total**
		dataflow total	**using X-marks**	
Grades:				
Control	13 / 20 (65%)	6 / 6 (100%)	n/a	19 / 26 (73%)
Treatment	13 / 16 (81%)	9 / 10 (90%)	5 / 5 (100%)	22 / 26 (85%)
Total	26 / 36 (72%)	15 / 16 (94%)		41 / 52 (79%)
Payroll:				
Control	6 / 17 (35%)	3 / 6 (50%)	n/a	9 / 23 (39%)
Treatment	9 / 21 (43%)	6 / 12 (50%)	3 / 5 (60%)	15 / 33 (45%)
Total	15 / 38 (39%)	9 / 18 (50%)		24 / 56 (43%)

Table 6: The success rates of identifying a fault contributing to an observed failure (faults identified/failures observed), for each debugging strategy.

Table 6 enumerates the subject's strategy choices and corresponding success rates. Comparing the first two column's percentages column-wise shows that, for both subject groups, dataflow debugging tended to be more successful than ad hoc. Within dataflow, the treatment subjects' success rates with X-marks exceeded the dataflow total success rates. A row-wise comparison of the denominators in the table also shows that treatment subjects tended to move to dataflow strategies nearly twice as frequently as the control subjects.

These differences in strategy choices lead to the following question: In what situations did the strategies matter?

Easy faults: The subject's strategy choices did not matter with the easiest faults: The easiest are mechanical faults, according to Allwood [Allwood 1984], and were usually found regardless of strategy used. Over all tasks and all subjects, 35 of the 40 mechanical faults were identified and 31 of these 35 faults identified were in the same cell in which a failure was spotted.

Difficult faults: The subject's choice of strategy did not matter with the difficult faults either: The difficult faults are logical faults, omission faults, according to Allwood [Allwood 1984], were equally found with both the ad hoc and dataflow strategy. Over all tasks and all subjects, 16 of the 31 difficult faults were identified with the adhoc strategy and the remaining 15 were identified using the dataflow strategy.

	Local		**Non-Local**	
Grades	Ad-hoc	Dataflow	Ad-hoc	Dataflow
Control	13	2	0	4
Treatment	13	3	0	6
Payroll				
Control	6	1	0	2
Treatment	7	2	0	4

Table 7: The success rates of subjects on local versus non-local faults for each debugging strategy.

Local faults: Local faults are those in which the failed value spotted by the subject was in the same cell as the faulty formula. Strategy also did not matter much with the "local" faults. This is often the case in smaller spreadsheets, where there are fewer cells to reference and the likelihood of a local fault is greater, and probably contributed to both groups' greater success in the Grades task. Table 7 enumerates the success rates of the subjects on local versus non-local faults for each debugging strategy.

Non-local faults: Non-local faults are those in which the failed value spotted by the subject was in a different cell in the forward slice of the cell containing the faulty formula. Strategy mattered a great deal for the non-local faults. Over all of the subjects and tasks, 16 non-local faults were identified—all using dataflow. *Not a single non-local fault was identified using the ad hoc strategy.* In fact, for 7 of these non-local fault identifications (by 6 different subjects), the subjects began their search for the fault using an ad hoc strategy and,

when unable to succeed, switched to a dataflow strategy, with which they succeeded in finding the fault.

The fault localization technique augments the dataflow strategy, which is illustrated by treatment subjects TS4 and TS5. Both subjects found all faults in the smaller Grades task. Both subjects also found the mechanical fault and one of the logical faults in the large Payroll task in short order. But then, they both got stuck on where to go next. At this critical juncture, TS4 decided to place an X-mark on a failure. Once he saw the feedback, he rapidly progressed through the rest of the task, placing 5 X's and correcting the final 3 faults in only 7 minutes. The transcripts show that the initial X-mark, which initiated the (dataflow-oriented) fault localization feedback, was a key turning point for him:

TS4: Right click that 'cause I know it's incorrect, highlights everything that's possible error... employee taxes is also incorrect. My net pay is incorrect. Adjusted gross pay is incorrect, so click those wrong.

Whereas TS4 made the defining decision to use the X-mark, TS5 did not. TS5's pattern of looking at cells gradually became ad hoc. He started randomly looking at formulas. He made decisions about the values in various cells and eventually happened upon a local fault, bringing his total to 3. He said "*I'm getting confused here*" numerous times, but did not change his approach.

To summarize, control group subjects resorting to an ad-hoc strategy occasionally stumbled upon and identified a fault. However, they were only able to find the "easier" and "local" spreadsheet faults, and never the "non-local" faults. Some times the subjects made use of the WYSIWYT dataflow arrows to trace the dataflow relationships from the observed failure. Such behavior mimics the information a fault localization technique would provide had such a resource been available to these subjects. Regardless, the data-flow strategy was often more successful and efficient than an ad-hoc strategy. When the treatment subjects were not employing a fault localization technique to localize a fault, they behaved much the same as the control subjects as described above. However, as soon as they

placed an X-mark, the visual debugging feedback often had an immediate impact on their debugging strategy: regardless of their previous strategy, they shifted to a data-flow strategy and were clearly more successful in finding more errors.

5.4 Results of Question 4:

How does this feedback influence an interactive debugging strategy?

We had initially expected that treatment subjects would always place X-marks whenever they observed a failure and use the subsequent visual feedback to guide their debugging, but this was not the case. Instead, they seemed to view the X-marks as a device to be called upon when they were in need of assistance. For example, only late in the session, when treatment subject TS1 got stuck debugging the failures, did he turn to the fault localization technique:

TS1 (thinking aloud): I don't know how to check the kind of error it is. I'll mark it wrong and see what happens.

When subjects did place an X-mark, the visual feedback often had an immediate impact: regardless of what their previous strategy had been, as soon as the feedback appeared, the subjects switched to a dataflow strategy by limiting their search to those cells with estimated fault likelihood and ignoring cells with no assigned fault likelihood.

TS1 (thinking aloud): I'm going to right-click on the total score. See that the weighted average, the weighted quiz, the weighted midterm, and the weighted final, and the error box all turn pink.

The fault localization device beckons the user toward a dataflow strategy, but it has attributes dataflow arrows do not have. First, it produces a smaller search space than the dataflow arrows, because it highlights only the producers that actually *did* contribute to a failure (the dynamic slice), rather than including the producers that could contribute to failures in other circumstances (the static slice). Second, it prioritizes the order in which the users should consider the cells, so that the ones most likely to be faulty are considered earliest. The above shows that

TS1's actions resulted in the reduction of search space brought about by the tinting of the producers of a cell with a failure, thus leaving TS1 in a advantageous situation than before. But did the subjects take advantage of the prioritization, indicated by some cells being darker than others?

Our electronic transcripts indicate that the answer to this question is yes. When the subjects searched cell formulas for a fault after placing an X-mark, 77% of these searches initially began at the cell with the darkest interior shading. As an example, here is a continuation of the above quote from TS1 after placing an X-mark:

TS1 (thinking aloud): See that the weighted average, the weighted quiz, the weighted midterm, and the weighted final, and the error box all turn pink. The total score box is darker though.

When the fault was not in the darkest cell, subjects' searches would gradually progress to the next darkest cell and so on. Some subjects realized that the coloring differentiations could be enhanced if they made further testing decisions by placing √ and X-marks, carried out by left- or right-clicking a cell's decision box.

TS4 (thinking aloud): Right click that 'cause I know it's incorrect, highlights everything that's possible errors. Now, I know my total gross pay is correct. I'll left click that one and simplify it.

From the results of this and the previous sections, it is evident that fault localization's ability to draw the user into a suitable strategy (dataflow) was important, particularly when subjects had not figured out a strategy that would help them succeed better than ad hoc approaches. Further, it is clear that subjects were influenced by the feedback's prioritization information when more than one color was present in that they looked first to the darkest cells, and then to the next darkest, and so on and that their doing so increased success.

5.5 Results of Question 5:

How do wrong testing decisions affect fault localization feedback?

Being human, the end-user subjects in our study made some mistakes in their testing decisions. Here we consider the types of mistakes they made, and the impact of these mistakes on the users' successful use of the fault localization feedback. Because the control subjects did not have fault localization feedback, we consider only the treatment subjects.

In total, the five treatment subjects placed 241 checkmarks, of which 11 (4.56%) were wrong—that is, the user pronounced a value correct when in fact it was incorrect. Surprisingly, however, no subjects made incorrect X-marks.

A possible reason for this difference may be a perceived seriousness of contradicting a computer's calculations, meaning subjects were only willing to place X-marks when they were really sure their decision was correct. For example, at one point, subject TS1 placed an X-mark in a cell, then reconsidered the mark because he was unsure the X-mark was really warranted.

TS1 (thinking aloud): So, I'll right click on that one. I'm not sure if this is right. Eh, I'll leave it as a question mark.

In contrast, checkmarks were often placed even if the user was unsure they were warranted. Our verbal transcripts include 10 different statements by treatment subjects with this sentiment. For example, consider the following from the same subject as quoted above:

TS1 (thinking aloud): I'll go ahead and left click the LifeInsurPrem box because I think that one's right for now.

TS3 (thinking aloud): I think these are right, (so) check that.

What impact did the wrong checkmarks have on fault localization? Four of the 11 wrong checkmarks were placed with a combination of X-marks, resulting in incorrect fault localization feedback. All four of these particular checkmarks, placed by three different subjects, adversely affected the subjects' debugging efforts.

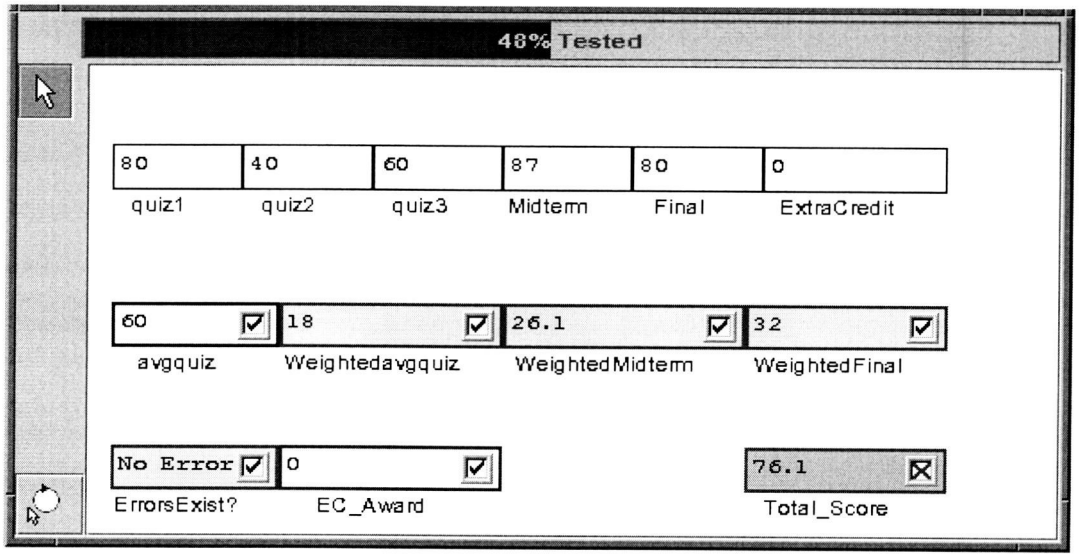


Figure 10a: The Grades task, with an incorrect checkmark in WeightedMidterm, as seen by subject TS1. TotalScore is the darkest, and the other 6 all are the same shade.

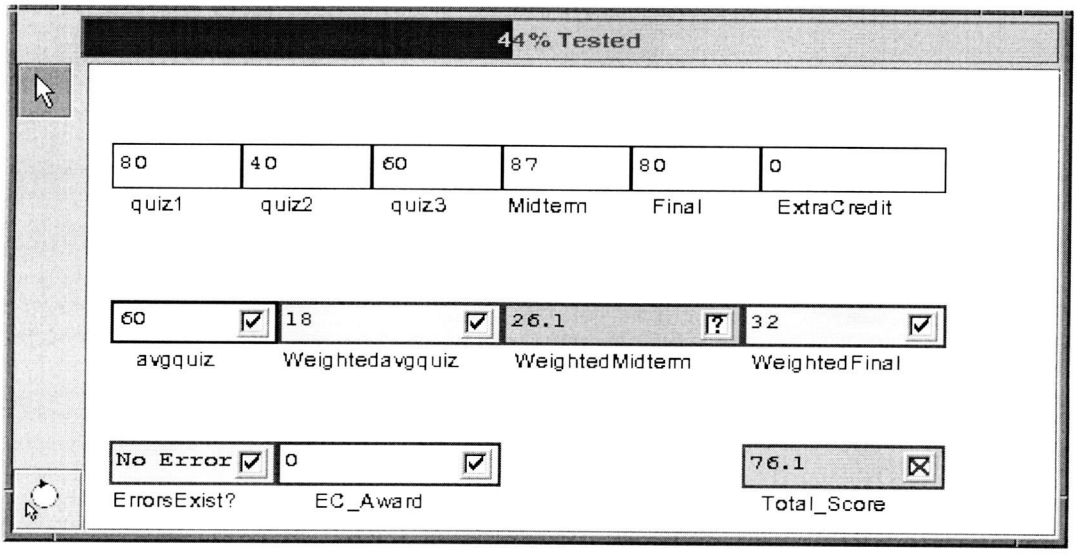


Figure 10b: What TS1 would have seen without the wrong checkmark: Weighted-Midterm would be as dark as TotalScore.

For example, during the Grades task, TS1 placed an incorrect checkmark in the (faulty) WeightedMidterm cell. He later noticed that the Total_Score cell,

although its formula was correct, had an incorrect value (due to the fault in WeightedMidterm). Unable to detect the source of this failure, he turned to the fault localization technique and placed an X-mark in the Total_Score cell:

TS1 (thinking aloud): The total score box is darker though. And it says the error likelihood is low, while these other boxes that are a little lighter say the error likelihood is very low. Ok, so, I'm not sure if that tells me anything.

The subject had checked the formula for the Total_Score a couple of times and hence he knew that Total_Score was correct. Figures 10a and 10b illustrates that had it not been for the wrong checkmark, the faulty cell WeightedMidterm cell would have been one of the two darkest cells in the spreadsheet. Instead, the wrongly placed checkmark caused WeightedMidterm to be colored the same as its correct siblings, thus providing the subject with no insightful fault localization feedback. (The subject eventually corrected the fault after a search of over six minutes.)

Subject TS2, faced with a similar scenario as in Figure 3, was overcome with confusion:

TS2 (thinking aloud): All right... so, I'm doing something wrong here. (long pause) I can't figure out what I'm doing wrong.

TS2's confusion resulted in nearly seven minutes of inactivity. He eventually located and corrected the fault, but remained flustered for the duration of the session.

As this evidence makes clear, it would not be realistic to ignore the fact that end users will provide some wrong information. In our study, even though fewer than 5% of the checkmarks placed by the subjects were wrong, these marks affected 60% (3 out of 5) of the treatment subjects' success rates! Given the presence of mistakes, robustness features are necessary to allow success even in the presence of mistakes. Toward this end, recall from Section 3.3 that the fault localization technique used in this study colors every cell in the dataflow chain contributing to a failure—even the cells the user may have previously checked off.

Clearly, however, this attempt at robustness was not enough to completely counteract the impacts of mistakes. Alternative techniques whose build-up of historical information can overcome some number of errors [Ruthruff et al 2003] are another possibility.

6. Discussion and Conclusion

6.1 Usage of X-marks

We had expected to see more usage of X-marks by the treatment group subjects. One possible reason is the method of placing an X-mark, right clicking a cell's decision box, seemed to be rather un-intuitive to our subjects. Right clicking to place an X-mark did not seem to come naturally to the subjects. Even though the subjects were taught how to place an X-mark in the tutorial, during the course of the experiment a couple of subjects asked the examiner how they could place an X-mark. For example a subject TS1 said:

TS1: How do I say that this value is wrong, I do not remember

One other subject TS3, who did not place a single X-mark over both the problems, remarked after the session that she completely forgot about the X-marks.

All this suggests that given a more intuitive way of placing X-marks the treatment group subjects would have placed more X-marks thereby taking advantage of the fault localization feedback and thus finding more faults with less effort and time.

6.2 Debugging Strategies

To make sure that the tutorial did not teach the subjects any particular strategy to debug the spreadsheet, only the fault localization feedback was explained to the subjects and they were not led through any strategy to help them debug.

We had expected the treatment subjects to use the fault localization support provided by the system early in their task. We expected them to compare the output values to the sample values given to them in the spreadsheet description and then as they spot failures place X-marks and take advantage of the reduced search space to find faults. But the subjects saw the fault localization technique as a tool

to be called upon only when they needed assistance. Most of the subjects started their search for faults in an adhoc fashion by randomly examining cell formulas and then when they made no further progress in their search for faults they shifted to a dataflow strategy by either placing X-marks or using arrows to find a fault. One possible reason is that the benefit of placing an X-mark was not clearly communicated to the subjects by our explanation system. We did not find a single situation where the users switched back from a dataflow strategy to an adhoc strategy. Considering that users had other ways of finding a fault, it might also be the case that for the users the benefit of placing an X-mark is not sufficient to get the users place X-marks early in their task.

6.3 Wrong Testing Decisions Affecting Fault Localization Feedback

Recall from the results that though fewer than 5% of the checkmarks placed by the subjects were wrong, they affected 60% of the treatment subjects' success rates. Moreover the data indicates that users rarely make mistakes in placing X-marks. In fact none of the X-marks placed by the users were wrong.

Given the presence of mistakes, robustness features are necessary to allow success even in the presence of mistakes. Further more, since we can say that users tend make mistakes while placing check marks there is a need to determine if checkmarks placed by the user should be considered while calculating the fault likelihood of a cell.

Recall from Section 3.3 that the fault localization technique used in this study colors every cell in the dataflow chain contributing to a failure—even the cells the user may have previously checked off. Clearly, however, this attempt at robustness was not enough to completely counteract the impacts of mistakes. Alternative techniques whose build-up of historical information can overcome some number of errors [Ruthruff et al 2003] are another possibility.

6.4 Threats to Validity

We attempted to address threats to internal validity by balancing the two groups of subjects in the experiment according to spreadsheet experience and programming background, by counterbalancing with respect to problem type, by equalizing training time and by selecting problems from familiar domain.

As in most experiments, however, threats to external validity are more difficult to address given the need to control all other factors. For Example, the spreadsheets used in the experiments may not be representative of the population of spreadsheets. However, although the spreadsheets may seem rather simple, given the limited time for the debugging task, most subjects could not find 100 % faults, indicating that the spreadsheets were not too simple for the amount of time given. The fact that our experiment included explicit time limits is a threat to external validity.

6.5 Conclusions

Previous fault localization research has focused primarily on techniques to aid *professional programmers* performing *batch* testing. In contrast, our study focuses on supporting *end-user programmers* with an *interactive* fault localization technique. Some revealing results were:

The subjects did not use fault localization from the beginning. Rather, they treated fault localization as a resource to be called upon only when they had exhausted their own debugging abilities. This supports the notion that end users won't necessarily use a programming resource just because it exists. When they did turn to fault localization, it often helped. In fact, end users generally saw the fault localization feedback as helpful to their interactive debugging, as evidenced by the continued use of the technique by the majority of our treatment subjects.

The subjects' understanding of the fault localization feedback was shallow. While all the subjects were able to interpret the feedback correctly, only a few could predict the feedback correctly under various circumstances.

Some subjects realized without help that a dataflow strategy was needed, but some did not. While dataflow-based debugging strategies may seem natural and intuitively superior in the eyes of traditional software engineers, our study indicates that such strategies may not come naturally to end-user programmers. Thus, any tool that seamlessly brings a more effective debugging strategy to an end user is valuable. One key way the fault localization technique helped was to lead them into a suitable strategy. Once subjects were engaged in a suitable strategy, fault localization helped further by prioritizing the order they should follow the strategy. Our study indicates that dataflow is a suitable strategy that would help find non-local errors. The question of bringing effective *interactive* debugging strategies to end users is one unanswered by previous research. Our study indicates that such effective, interactive strategies may be feasible for end users. However, more evidence is needed to support this possibility.

Previous fault localization research has focused primarily on techniques to aid professional programmers performing batch testing of test suites. Our study brings a new and unique approach to fault localization by considering *interactive* debugging with respect to end-user programmers. In exploring this issue, we found that end users make mistakes, and because even a few mistakes can have a big impact on fault localization's helpfulness, the importance of these mistakes should not be ignored. Thus, fault localization techniques should include features to enhance robustness in the face of a few mistakes.

Perhaps the most challenging result was the important role of early interactive feedback. Our study found that if a fault localization technique's *early* feedback is not seen to be useful, users may not give the technique a chance to produce better feedback later. The early feedback of a fault localization technique may be of little consequence to professional programmers performing batch testing of test suites; yet this issue may be paramount to the success of an interactive technique in an end-user programming environment.

7. References

[Allwood 1984] C. Allwood , "Error Detection Processes in Statistical Problem Solving", *Cognitive Science* 8, 4, 1984, 413-437.

[Blackwell 2002] A. Blackwell, "First steps in programming: a rationale for attention investment models," *Proc. IEEE Human-Centric Computing Langauges and Environment*s, Arlington, VA, Sept. 2002,2-10.

[Boehm 2000] Boehm, B., E. Horowitz, R. Madachy, D. Riefer, B. Clark, B. Steece, A. W. Brown, S. Chulani, C. Abts, Software Cost Estimation With Cocomo II. Prentice-Hall, Englewood Cliffs, NJ, 2000.

[Brown and Gould 1987] Brown, P. and J. Gould, "Experimental study of people creating spreadsheets," ACM Transactions on Office Information Systems, 5(3), July 1987, 258-272.

[Burnett et al 2001] M.Burnett, J. Atwood, R.Djang, H.Gittfried, J. Reechwein, and S. Yang, "Forms/3: A First- Order Visual Langauge to Explore the Boundaries of the Spreadsheet Paradigm, " *J. Func. Prog.* 11,2,Mar. 2001, 155-206.

[Burnett et al 2003] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace, "End-User Software Engineering with Assertions in the Spreadsheet Paradigm", Int. Conf. Soft. Eng., Portland, OR, May 2003 , 93-103.

[Carr 2003] D.A. Carr, "End-User Programmers Need Improved Development Support", CHI 2003 Workshop on Perspectives in End User Development, April 2003, 16-18.

[Chen and Cheung 1997] T.Y. Chen and Y.Y. Cheung, "On Program Dicing", Soft. Maintenance: Research and Practice 9, 1, 1997, 33-46.

[Cragg and King 1993] Cragg, P.G. & King, M.,"Spreadsheet Modeling Abuse: An opportunity for OR?", Journal of the operational Research Society(44:8) August 1993, 743-753.

[Davis 1996] J.S. Davis, "Tools for Spreadsheet Auditing", Int. J. Human-Computer Studies 45, 1996, 429-442.

[Jones et al 2002] J.A. Jones, M.J. Harrold, and J. Stasko, "Visualization of Test Information to Assist Fault Localization", Int. Conf. Soft. Eng., May 2002, 467-477.

[Korel and Laski 1990] B. Korel and J. Laski. , "Dynamic slicing of computer programs",.The Journal of Systems and Software, 13(3):187–195, November 1990.

[Lieberman and Fry 1998] H. Lieberman and C. Fry, "ZStep 95: A Reversible, Animated Source Code Stepper", Soft. Visualization: Programming as a Multimedia Experience (J. Stasko, J. Domingue, M. Brown, and B. Price, eds.), MIT Press, Cambridge, MA, 1998, 277-292.

[Lyle and Weiser 1987] J.R. Lyle and M. Weiser.,"Automatic program bug locationby program slicing"., In Proceedings of the 2nd International Conference, Computers and Applications, pages877–883, 1987.

[Myers and Ko 2003] B. Myers and A. Ko, "Studying Development and Debugging to Help Create a Better Programming Environment", CHI 2003 Workshop on Perspectives in End User Development, April 2003, 65-68.

[Nardi and Miller 1991] Nardi, B. and J. Miller, "Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development," Int. J. Man-Machine Studies, 34 (1991), pp.161-184.

[Panko 1998] Panko, R., "What we know about spreadsheet errors," Journal of End User Computing, Spring 1998, 15-21.

[Reichwein et al 1999] J. Reichwein, G. Rothermel, and M. Burnett, "Slicing Spreadsheets: An Integrated Methodology for Spreadsheet Testing and Debugging", 2nd Conf. Domain Specific Langs., Oct. 1999, 25-38.

[Rothermel et al 1998] G. Rothermel, L. Li., C. Dupuis, and M. Burnett, "What You See Is What You Test: A Methodology for Testing Form-based Visual Programs", Int. Conf. Soft. Eng., Apr. 1998, 198-207.

[Ruthruff et al 2003] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Prabhakararao, M. Fisher II, and M. Main, "End-User Software Visualizations for Fault Localization", ACM Symp. Soft. Visualization, Jun. 2003, 123-132.

[Sajanieme 2000] J. Sajanieme, "Modeling Spreadsheet Audit: A Rigorous Approach to Automatic Visualization", J. Visual Langs. and Computing 11, 1, 2000, 49-82.

[Telcordia Technologies 1998] Telcordia Technologies, "xSlice: A Tool for Program Debugging", xsuds.argreenhouse.com/html-man/coverpage.html, July 1998.

[Tip 1995] F. Tip, "A Survey of Program Slicing Techniques", J. Programming Langs. 3, 3, 1995, 121-189.

[Weiser 1984] M. Weiser., "Program slicing". IEEE Transactions on Software Engineering, SE-10(4):352–357, July 1984.

Appendices

Appendix A: Tutorial Materials

Think-aloud introduction read to subjects before the tutorial:

"In this experiment we are interested in what you say to yourself as you perform some tasks that we give you. In order to do this we will ask you to TALK ALOUD CONSTANTLY as you work on the problems. What I mean by talk aloud is that I want you to say aloud EVERYTHING that you say to yourself silently. Just act as if you are alone in this room speaking to yourself. If you are silent for any length of time, I will remind you to keep talking aloud. It is most important that you keep talking. Do you understand what I want you to do ?

Good. Before we turn to the real experiment and the tutorial, we will start with a couple of practice questions to get you used to with talking aloud. I want you to talk aloud as you do these problems. First I will ask you to add two numbers in your head.

So talk aloud while you add 234 and 456

Good. Now I will ask you one more question before we proceed with the main experiment. I want you to do the same thing as you did for the addition problem. I want you to talk aloud while you answer the question.

How many windows are there in your parent's house?"

Tutorial given to treatment group subjects

In this experiment you will be working with the spreadsheet language Forms/3. To get you familiarized with the features of Forms/3, we're going to start with a short tutorial in which we'll work through a couple sample spreadsheet problems. After the tutorial you will be given different spreadsheets and will be asked to test the spreadsheets and correct any errors you find in them.

As we go through this tutorial, I want you to actually DO the steps I'm describing. When I say, "click", I'll always mean click the left mouse button once unless I specify otherwise. Pay attention to your computer screen while you do the steps .If you have any questions, please don't hesitate to ask me to explain.

For each spreadsheet that we will be working, you will have a sheet of paper describing what the spreadsheet is supposed to do

Read the first page of the description of the PurchaseBudget spreadsheet now. Now open the Purchase Budget spreadsheet by Double clicking on the icon labeled Purchase Budget at the bottom of the screen

This is a Forms/3 spreadsheet.

There are some obvious differences between Forms/3 and other spreadsheets. Forms/3 spreadsheets don't have cells in a grid layout. We can put cells anywhere. We can give the cells useful names like PenTotalCost. (Point to the cell on the Spreadsheet) However, just like with any other spreadsheet you can see a value associated with each cell. You can select a cell by clicking on it. Click on the Pens cell.

The Pens cell now has a selection outline indicating you have selected it. You can move the cell by holding the left mouse button down over the cell and dragging. You can also resize the cell by dragging one of the black boxes on the selection outline. Try moving and resizing the Pens cell. For undoing the selection just click outside the cell. You can also see that some cells have colored borders. I'll explain their meaning shortly.

Let's find out what the red color around the border means. Rest your mouse on top of the border, a message will pop up and tell us what this color means. Read it. Yes, it means that the cell has not been tested.

You're probably wondering, what does testing have to do with spreadsheets? Well, it's possible for errors to exist in spreadsheets but what usually happens is that they tend to go unnoticed. It's in our best interest to find and correct the bugs or errors in our spreadsheets so that we can be confident that they are correct.

So, the red border around the cells is just telling us that the computer does not know if the cell's value is correct. It's up to us to make a decision about the correctness of the cells based on how we know the spreadsheet should work. In our case we have the spreadsheet description that tells us how it should work.

Observe that Pens & Paper cells don't have any special border color. Such cells without colored borders are called input cells. Cells with colored borders are output cells.

Now, Lets test the BudgetOk cell by making a decision whether or not the value is correct for the current inputs. Read the description of the PurchaseBudget Spreadsheet for the BudgetOk cell. (Pause). According to spreadsheet description this cell tells you if you have spent more than your budget and you cannot spend more than $2000. For the Current set of inputs the Total cost should be 1600, which is less than 2000. That means the value associated with the Budget Ok cell is "Wrong".

Now let's make a decision about the correctness of the BudgetOk cell. Move your mouse to the box with the question mark in it and hold it there until a message pops up. What does it say? The message tells us that **if** the cell's value is correct to go ahead and left - click and if it is Wrong, Right-Click. Go ahead and right click.

Notice what happened. The cell BudgetOK and the cells upstream have been tinted with pink. Lets find out what does this mean. Move your mouse over the pink shade and hold it until the message appears. What does the message say? Yes, it means that there is a possibility of errors being present in one of the cells shaded pink. It also says that the Error likelihood is LOW. We could check the formula or to narrow our search make decisions for other cells which would help us find errors.

So, now let's make a decision about the correctness of the TotalCost cell. As we said before the value for total cost should be 1600 instead of 2800. So go ahead and Right click to give your decision on this cell. Observe the changes. The interior of the TotalCost cell and the cells up stream gets darker. Now mouse over to find what it means. (Pause) Yes, Error likelihood is Medium. So, to narrow the search further, lets make a decision on the cell PaperTotalCost. Read the description for this cell. According to the description the rate of paper is $4. At this rate the value in the cell should be 1600 instead of 2800. So right click to give your Decision for this cell. What changed? Yes, The interior of the cell gets darker. Mouse over to find what that means. The Error likelihood is "HIGH". Ok, now let's check the formula for this cell. To do this click on the formula tab. According to the description the formula should be Paper * 4 instead of Paper * 7. Correct the formula and click on the apply button to save your changes.

Notice what happened. Everything goes back to what it was before. The Question marks reappear. Now let's make a decision about the correctness of the TotalCost cell. Move your mouse to the box with the question mark in it and hold it there until a message pops up. What does it say? The message tells us that if the cell's value is correct to go ahead and left - click and if it is Wrong, Right-Click. Go ahead and left click .

Notice what happened. Three things changed. A checkmark replaced the question mark in the decision box. The borders for the cell TotalCost turned Purple, and the % testedness indicator changed to 28% (point to it). Forms/3 lets

us know what percent of the spreadsheet is tested through the % testedness indicator. It is telling us that we have tested 28% of this spreadsheet.

If you accidentally checked off the decision box, the value in the cell was really wrong, or you haven't seen the changes that occurred, you can "uncheck" the decision about TotalCost with another click in the same decision box. Try it. (Pause) Everything went back to how it was. The cells' borders turned back to red, the % testedness indicator dropped back to 0% and a question mark reappeared in the decision box.

Since we've already decided the value in the TotalCost cell is correct, we want to retell Forms/3 that this value is correct for the inputs. So click in the decision box. You may have noticed that the border color of the PenTotalCost and PaperTotalCost cells is blue Now find out what the blue border indicates by holding the mouse over the cell's border in the same way as before. What does the message say? It tells us that the cell is fully tested. Also notice the blank decision box in the PenTotalCost and PaperTotalCost cells .What does that mean? Position your mouse on top of the box to find out why it is blank. A message pops up that says we have already made a decision about this cell, but I don't remember us making any decisions about PenTotalCost. How did that happen? Let's find out. Position your mouse to the TotalCost cell and click the middle mouse button. Notice that colored arrows appear,

Click the middle mouse button again on one of the arrows -- it disappears. Now, click the middle mouse button again on TotalCost cell -- all the other arrows disappear. Now bring the arrows back again by re-clicking the middle mouse button on TotalCost. Move your mouse over to the blue arrow and hold it there until a message appears. It explains the arrow is showing a relationship that exists between TotalCost and PentotalCost The answer for PenTotalCost goes into or contributes to the answer for TotalCost.

Oh, ok, so this explains why the arrow is pointed in the direction of TotalCost? Yes it is, and it also explains why the cell borders of PenTotalCost and PaperTotalCost turned blue. Again, if you mark one cell as being correct and there were other cells contributing to it, then those cells will also be marked correct.

Now let us test the cell BudgetOk, ok now refer back to the description for the cell on the page of the spreadsheet description. (Pause). According to spreadsheet description this cell tells you if you spent more than your budget.

The current set of values covers a situation when TotalCost is less than 2000 dollors. The BudgetOk Cell displays "Budget Ok" Is this the correct answer for this situation? (Look at the cell description and then make a decision—Pause). Yes it is the correct answer. go ahead and click the decision box for BudgetOk.

Notice the BudgetOK cell's border is now a shade of purple. Now find out what the purple border indicates by holding the mouse over the cell's border in the same way as before. What does the message say? It tells us that the cell is 66% tested. It is partially tested. This means there are more situations for this cell that we haven't tried to test. Let's try to find out what those situations could be. Open the formula for BudgetOk cell.

Remember that the description said that the Budgetok cell tells you if you spent more than your budget .We tested the part for 2000 dollors or less, what should we do now? We tested the then portion of the nested if statement, so we should now test the else part of the same if statement. Ok now lets do that. Let's change the value in the Pens cells to 200.

To do this:

1) Click on the formula tab for the Pens cell and delete the old value and enter the new value.
2) After you finish entering the new value, click on the apply button.
3) Now close the formula window .

You should notice that the new value appears in the Pens cell as soon as you click the apply button . and that the value for BudgetOk immediately changed. Notice that the checkmark in the TotalCost decisionbox and the BudgetOK decisionbox were replaced with a question mark. A question mark means that you need to make a decision about the correctness of the value in the cell. Look at the value in the BudgetOK cell. Do you think it is Correct? Yes. Go ahead and Make your Decision.

Remember that the ultimate goal is to make sure the spreadsheet works like it says in the description. Refer back to it as many times as you need it. Take two minutes now to explore the spreadsheet. See if you can find any more bugs and if you do, fix them. If you need help or can't remember how something works, use the mouse over feature to get more information about it.
(pause 2 minutes)

Exploratory Task #2:

Ok, time is up. To continue developing the skills you'll need in a few minutes, I'm going to ask you first minimize the spreadsheet you were working with. Then, open the PurchaseBudget_Decision spreadsheet and read the spreadsheet description for it. (read it with them--so we now how long to pause)

Now read the tutorial task. (pause) . Some spreadsheet developer has already created the spreadsheet according to the description given.

Now, you'll have about 5 minutes to explore the spreadsheet. If you encounter any errors, fix them. Remember that the ultimate goal is to make sure the spreadsheet works like it says in the description. Refer back to it as many times as you need it. If you need help or can't remember how something works, use the mouse over feature to get more information about it.
(pause 5 minutes)

End of Tutorial.

Tutorial given to control group subjects

In this experiment you will be working with the spreadsheet language Forms/3. To get you familiarized with the features of Forms/3, we're going to start with a short tutorial in which we'll work through a sample spreadsheet problem. As the next task in the tutorial you will be given a practice problem and the task is similar to what you will be doing in the experiment. You will be then given a different spreadsheet and will be asked to test the spreadsheet and correct any errors you find.

As we go through this tutorial, I want you to actually DO the steps I'm describing. When I say, "click", I'll always mean click the left mouse button once unless I specify otherwise. Pay attention to your computer screen while you do the steps .If you have any questions, please don't hesitate to ask me to explain.

For each spreadsheet that we will be working, you will have a sheet of paper describing what the spreadsheet is supposed to do

Read the first page of the description of the PurchaseBudget spreadsheet now. Now open the Purchase Budget spreadsheet by Double clicking on the icon labeled Purchase Budget at the bottom of the screen

This is a Forms/3 spreadsheet. There are some obvious differences between Forms/3 and other spreadsheets.Forms/3 spreadsheets don't have cells in a grid layout. We can put cells anywhere. We can give the cells useful names like PenTotalCost. (Point to the cell on the Spreadsheet). You can also see that some cells have colored borders. I'll explain their meaning shortly. However, just like with any other spreadsheet you can see a value associated with each cell. You can select a cell by clicking on it. Click on the Pens cell (pause).The Pens cell now has a selection outline indicating you have selected it. You can move the cell by holding the left mouse button down over the cell and dragging. You can also resize the cell by dragging one of the black boxes on the selection outline. Try moving and resizing the Pens cell. To unselect a cell just click outside the cell

More often than we would like spreadsheets contain bugs or errors. We would like to find and fix these bugs so that we can be confident that our spreadsheet is correct. Testing is one way of finding bugs. Testing is making decisions about the correctness of the cells based on how the spreadsheets should work .The computer does not know if the cells value is correct. It's up to you to make these decisions

First lets look at the cell border colors. What does the red color mean? To find out, rest your mouse on top of the border, a message will pop up and tell us what this color means. Read it. Yes, it means that the cell has not been tested.

Observe that Pens & Paper cells don't have any special border color. Such cells without colored borders are called input cells. Cells with colored borders are output cells.

Now let's make a decision about the correctness of the PensTotalCost cell. Move your mouse to the box with the question mark in it and hold it there until a message pops up. What does it say? The message tells us that **if** the cell's value is correct to go ahead and left - click and if it is Wrong, Right-Click. Go ahead and left-click. What happened? Three things changed. A checkmark replaced the question mark in the decision box. The borders for the cell turned blue , and the % testedness indicator changed to 7% (point to it). Forms/3 lets us know what percent of the spreadsheet is tested through the % testedness indicator. It is telling us that we have tested 7% of this spreadsheet.

Now find out what the blue border indicates by holding the mouse over the cell's border in the same way as before. What does the message say? It tells us that the cell is fully tested

You can undo your decisions too. Now uncheck the PenTotalCost cell by left –clicking the mouse button in the decision box.

Now let's see what happens if we right click in the decision box.Go ahead and Right-click. Observe that an X-mark Replaced the question mark.Now find out what that means by holding the mouse on the X-mark.What does the message say? It tells that you have decided that the value associated with this cell is wrong. The X- mark reminds you that the value of a particular cell is wrong and that there is possibility of an error in it. You can undo the X on the PenTotalCost cell by right-clicking the mouse button in the decision box. Do it now.

Now, Lets test the BudgetOk cell by making a decision whether or not the value is correct for the current inputs. Read the description of the PurchaseBudget Spreadsheet for the BudgetOk cell. (Pause). According to spreadsheet description this cell tells you if you have spent more than your budget and you cannot spend more than $2000. For the Current set of inputs the Total cost should be 1600, which is less than 2000. That means the value associated with the Budget Ok cell is "Wrong". Go ahead and right click.

As the value in this cell is wrong, there is possibility of an error in the formula for this cell. It is also possible that there could be an error in the cell's which contribute to the value for this cell. Let's now find out what are those cells which contribute for the value in this cell. Position your mouse to the BudgetOk cell and click the middle mouse button. Notice that colored arrow appears, click the middle mouse button again on the arrow -- it disappears. Now bring the arrows back again by re-clicking the middle mouse button on BugetOK. Move your mouse over to the arrow and hold it there until a message appears. It explains the arrow is showing a relationship that exists between TotalCost and BudgetOK The answer for TotalCost is used or contributes to the answer for BudgetOK. Now click the middle mouse button on the TotalCost cell.see that the PenTotalCost and the PaperTotalCost contribute for the value in the TotalCost cell. Check the value in theses cells Do you see a wrong value? Look at the PaperTotalCost Cell. According to the description the rate for paper is $4 so the PaperTotalCost should be 1600 instead of 2800. So, now let's check the formula for this cell. To do this

click on the formula tab. According to the description the formula should be Paper * 4 instead of Paper * 7. Correct the formula and click on the apply button to save your changes.

Notice what happened. Everything goes back to what it was before. The Question marks reappear. Now let's make a decision about the correctness of the BudgetOK cell. We know that the value in the BudgetOK cell is correct. So, Go ahead and left click.

You may have noticed that the border color of the TotalCost and is blue.Also notice the blank decision box in the TotalCost cell .What does that mean? Position your mouse on top of the box to find out why it is blank. A message pops up that says we have already made a decision about this cell, but I don't remember us making any decisions about TotalCost. How did that happen? We know that the answer for TotalCost goes into or contributes to the answer for BudgetOK. If you mark one cell as being correct and there were other cells contributing to it, then those cells will also be marked correct. So this explains why the cell borders of TotalCost cell turned blue.

Notice the BudgetOK cell's border is now a shade of purple. Now find out what the purple border indicates by holding the mouse over the cell's border in the same way as before. What does the message say? It tells us that the cell is 50% tested. It is partially tested. This means there are more situations for this cell that we haven't tried to test. Let's try to find out what those situations could be. Open the formula for BudgetOk cell.

Remember that the description said that the Budgetok cell tells you if you spent more than your budget .We tested the part for 2000 dollars or less, what should we do now? We tested the then portion of the if statement, so we should now test the part for more than 2000 dollars , the else part of the same if statement.

Ok now let's do that. We need to find a situation when the TotalCost exceeds the value of 200. We can do that by changing our input values. Let's change the value in the Pens cells to 200.

To do this:

1) Click on the formula tab for the Pens cell and delete the old value and enter the new value.
2) After you finish entering the new value, click on the apply button.
3) Now close the formula window.

You should notice that the new value appears in the Pens cell as soon as you click the apply button and that the value for BudgetOk immediately changed. Notice that the checkmark in the TotalCost decisionbox and the BudgetOK decisionbox were replaced with a question mark. A question mark means that you need to make a decision about the correctness of the value in the cell. Look at the value in the BudgetOK cell. Do you think it is Correct? Yes. Go ahead and Make your Decision.

Remember that the ultimate goal is to make sure the spreadsheet works like it says in the description. Refer back to it as many times as you need it. Take a minute now to explore the spreadsheet. See if you can find any more bugs and if you do, fix them. If you need help or can't remember how something works, use the mouse over feature to get more information about it.
(pause 2 minutes)

Exploratory Task #2:

Ok, time is up. To continue developing the skills you'll needfor the experiment, I'm going to ask you first minimize the spreadsheet you were working with. Then, open the PurchaseBudget2 spreadsheet and read the spreadsheet

description for it. (read it with them--so we now how long to pause) This is a Practice problem task similar to what you will be doing in the experiment.

Now read the tutorial task. (Pause).

Some spreadsheet developer has already created the spreadsheet according to the Description given. Now, you'll have about 5 minutes to explore the spreadsheet. Remember that the ultimate goal is to make sure the spreadsheet works like it says in the description. If you encounter any errors, fix them. Refer back to the description as many times as you need it. If you need help or can't remember how something works, use the mouse over feature to get more information or catch one of the assistants.

(Pause 5 minutes)

End of Tutorial.

Appendix B: Spreadsheet Descriptions and Questionnaires

Grades Spreadsheet Description

Grade for a course

The total score for a class you are taking is based on three quizzes, one midterm , one Final and one extra-credit assignment. You can determine your Total score in the class once you know your scores for the quizzes, midterm and final and the extra-credit assignment. All quizzes, midterm, final and extra credit scores range from 0 to 100.

The Total Score ranges from 0 to 100 plus extra credit is calculated in the Forms/3 spreadsheet as follows:

1. 30 % of the total score is the average of 3 quizzes, 30% for the midterm and 40% for the final.
2. Extra credit is computed as follows:
   5 points of extra credit for a score greater than 25 on the extra-credit assignment.
   3 points of extra credit for a score of 20 to 25 on the extra-credit assignment.
   0 points of extra credit for scores less than 20 on the extra-credit assignment.

Input Cells	
quiz1	Score for quiz one.
quiz2	Score for quiz two.
quiz3	Score for quiz three.
Midterm	Score for the midterm.
Final	Score for the Final.
ExtraCredit	Score on the extra-credit assignment
Output Cells	
ErrorsExist?	Lets you know if you accidentally entered a quiz score of greater than 100
TotalScore	The total score you are getting for the class

The table below gives you an example Correct Total Score of a student.

Quiz1	Quiz2	Quiz3	midterm	final	Extra credit	Total Score
50	80	80	70	80	26	79

Task

You are to thoroughly test the Forms/3 spreadsheet and correct any errors you find.

Payroll Spreadsheet Description

A spreadsheet program that computes the net pay of an employee has been updated by one of your co-workers.

Below is a description about how to compute the answers.

On the backside of this sheet are two correct examples, which you can compare with the values on screen.

Your task is to test the updated spreadsheet to see if it works correctly and to correct any errors you find

FEDERAL INCOME TAX WITHHOLDING

To determine the federal income tax withholding:

From the monthly adjusted gross pay subtract the allowance amount (number of allowances claimed multiplied by $250). Call this amount the adjusted wage. Calculate the withholding tax on adjusted wage using the formulas below:

If Single and adjusted wage is not greater than $119, the withholding tax is $0; otherwise the withholding amount is 10% of (adjusted wage – $119).

If Married and adjusted wage is not greater than $248, the withholding tax is $0; otherwise the withholding amount is 10% of (adjusted wage – $248).

SOCIAL SECURITY AND MEDICARE

Social Security and Medicare is withheld at a combined rate of 7.65% of Gross Pay. The Social Security portion (6.20%) will be withheld on the first $87,000 of Gross Pay, but there is no cap on the 1.45% withheld for Medicare.

INSURANCE COSTS

The monthly health insurance premium is $480 for Married and $390 for Single. Monthly dental insurance premium is $39 for Married and $18 for Single. Life insurance premium rate is $5 per $10,000 of insurance. The monthly employer insurance contribution is $520 for Married and $300 for Single.

ADJUSTED GROSS PAY

Pretax deductions (such as child care and employee insurance expense above the employer's insurance contribution) are subtracted from Gross Pay to obtain Adjusted Gross Pay.

Example Correct Payroll Stubs

John Doe	Month	Year-To-Date
Marital Status – Single		
Allowances	1	
Gross Pay	6,000.00	54,000.00
Pre-Tax Child Care	0.00	
Life Insurance Policy Amount	10,000	
Health Insurance Premium	390.00	
Dental Insurance Premium	18.00	
Life Insurance Premium	5.00	
Employee Insurance Cost	413.00	
Employer Insurance Contribution	300.00	
Net Insurance Cost	113.00	
Adjusted Gross Pay	5,887.00	
Federal Income Tax Withheld	551.80	
Social Security Tax	372.00	
Medicare Tax	87.00	
Total Employee Taxes	1,010.80	
Net Pay	4,876.20	

Mary Smith	Month	Year-To-Date
Marital Status – Married		
Allowances	5	
Gross Pay	8,000.00	72,000.00
Pre-Tax Child Care	400.00	
Life Insurance Policy Amount	50,000	
Health Insurance Premium	480.00	
Dental Insurance Premium	39.00	
Life Insurance Premium	25.00	
Employee Insurance Cost	544.00	
Employer Insurance Contribution	520.00	
Net Insurance Cost	24.00	
Adjusted Gross Pay	7,576.00	
Federal Income Tax Withheld	607.80	
Social Security Tax	496.00	
Medicare Tax	116.00	
Total Employee Taxes	1,219.80	
Net Pay	6,356.20	

Background Questionnaire

3.Major _____

4.Year _____

5.Overall GPA _____

6.Do you have previous programming experience? (Check all that apply)

- High school course(s). How many? _____
- College course(s). How many? _____
- Professional. How long? _____

7.Have you ever created a spreadsheet for _____? (Check all that apply)

A high school course

A college course

Professional use

Personal use

8.Have you participated in any previous Forms/3 experiments? Yes / No

9.Is English your primary language? Yes / No
If not, how long have you been speaking English? _____

Post Session Questionnaire for the treatment group

1. Use this scale to answer the following questions.

   1 = Not Confident
   2 = Somewhat Confident
   3 = Confident
   4 = Quite Confident
   5 = Very Confident

How confident are you that you found all the bugs in the Total Score spreadsheet? 1 2 3 4 5

How confident are you that you fixed all the bugs in the TotalScore spreadsheet? 1 2 3 4 5

2. How much additional time would you need to complete this task?

_____ None. It only took me _____ minutes.
_____ None. I took about the whole time.
_____ I would need about _____ more minutes.
_____ I'm not sure.

1. Use this scale to answer the following questions.

   1 = Not Confident
   2 = Somewhat Confident
   3 = Confident
   4 = Quite Confident
   5 = Very Confident

How confident are you that you found all the bugs in the Payroll spreadsheet? 1 2 3 4 5

How confident are you that you fixed all the bugs in the Payroll spreadsheet? 1 2 3 4 5

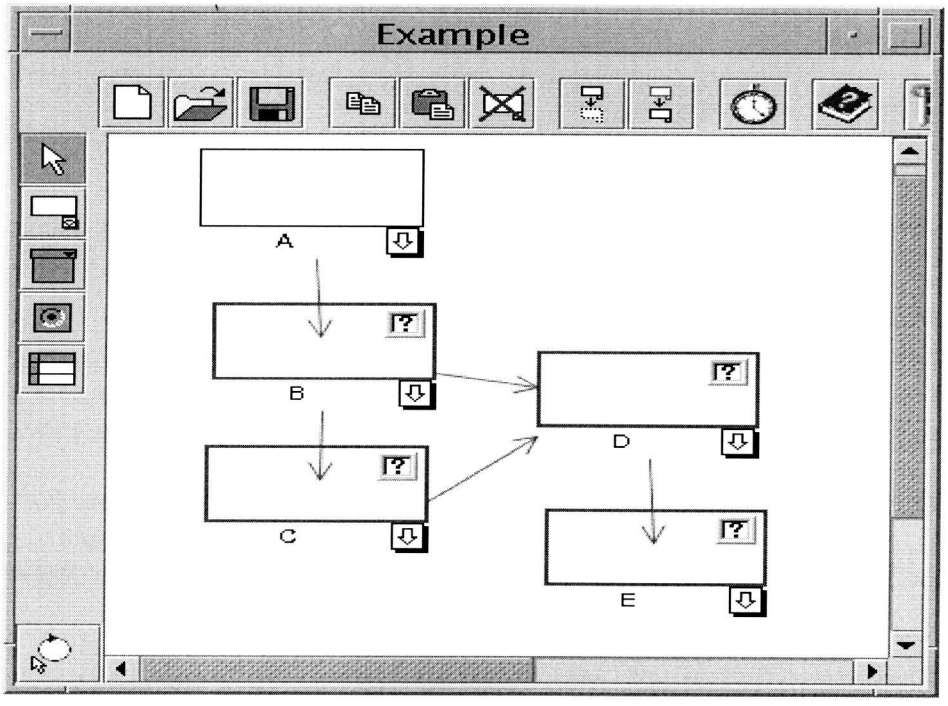
2. How much additional time would you need to complete this task?

_____ None. It only took me _____ minutes.
_____ None. I took about the whole time.
_____ I would need about _____ more minutes.
_____ I'm not sure.

3. Use this scale to answer questions regarding use various tools in finding and fixing
errors:

1 = No Opinion
2 = Not HelpFul
3 = Somewhat Helpful
4 = Helpful
5 = Quite Helpful
6 = Very Helpful

Cell border colors were	1	2	3	4	5	6
Interior Cell Coloring (pink and red)	1	2	3	4	5	6
Pop up messages were	1	2	3	4	5	6
Arrows were	1	2	3	4	5	6
Percent tested indicator was	1	2	3	4	5	6



Q4 to Q9: Refer to the Figure Above and choose your answers from the choices below.

One or more Questions can have the same answer.

Choices for answers to Q4 to Q9.

A. Remains the same.
B Gets Darker.
C. Gets Lighter
D. Don't Know.

4. If we place an X- mark in cell D the color of the cell D _____

5. If we place an X- mark in cell D the color of the cell C _____

6. If we place an X- mark in cell D the color of the cell E _____

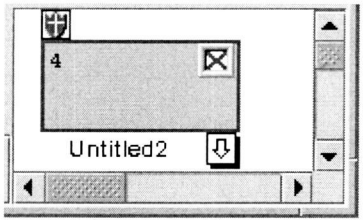
Assume for the next three Questions that an X- mark has been placed on the cell D.

7. If we place an X- mark in cell C the color of the cell C _____

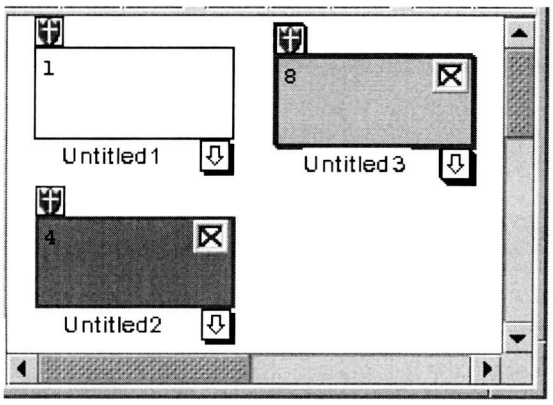
8. If we place an X- mark in cell C the color of the cell B _____

9. If we place an Check- mark in cell C the color of the cell D _____

10. What does the X- mark in the decision box mean?



11. In the above figure what does the pink color in the interior of the cell mean?



12. In the above figure what does it mean when the colors in the interior of one cell is darker the others?

Please provide any other general comments/suggestions you may have regarding the cell interior colorings :

_____

_____

_____

_____

13. Given the Situation, I described in the purchase budget problem, what does it mean

when the colors in the interior of the cells get Darker?

14. Given the Situation, I described in the purchase budget problem, what does it mean

when the colors in the interior of the cells get lighter?

Post session Questionnaire for the control group

1. Use this scale to answer the following questions.

1 = Not Confident
2 = Somewhat Confident
3 = Confident
4 = Quite Confident
5 = Very Confident

How confident are you that you found all the bugs in the Payroll spreadsheet?	1	2	3	4	5
How confident are you that you fixed all the bugs in the Payroll spreadsheet?	1	2	3	4	5

2. How much additional time would you need to complete this task?

_____ None. It only took me _____ minutes.
_____ None. I took about the whole time.
_____ I would need about _____ more minutes.
_____ I'm not sure.

➢Use this scale to answer questions regarding the following features:

1 = No Opinion
2 = Not Helpful
3 = Somewhat Helpful
4 = Helpful
5 = Quite Helpful
6 = Very Helpful

Seeing a blank in a decision box was:	1	2	3	4	5	6
Seeing a check mark in a decision box was:	1	2	3	4	5	6
Seeing a question mark in a decision box was:	1	2	3	4	5	6
Seeing different colors to indicate the testedness was:	1	2	3	4	5	6
Seeing Arrows was	1	2	3	4	5	6

Please provide any other general comments/suggestions you may have regarding the Testing tools :

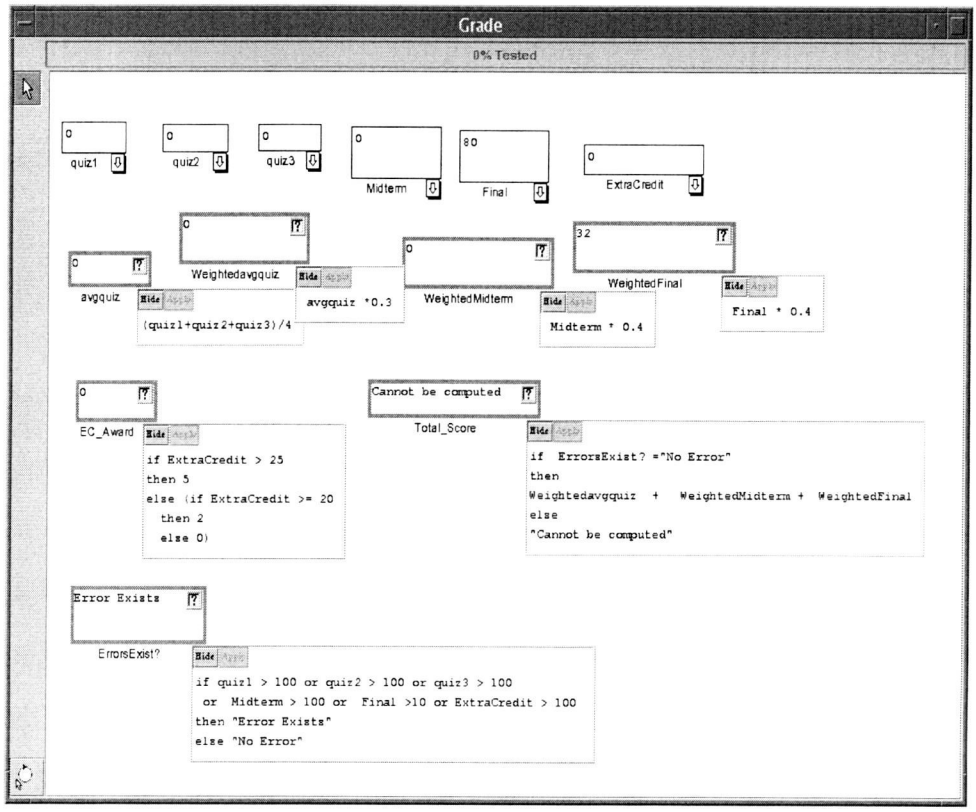
_____

_____

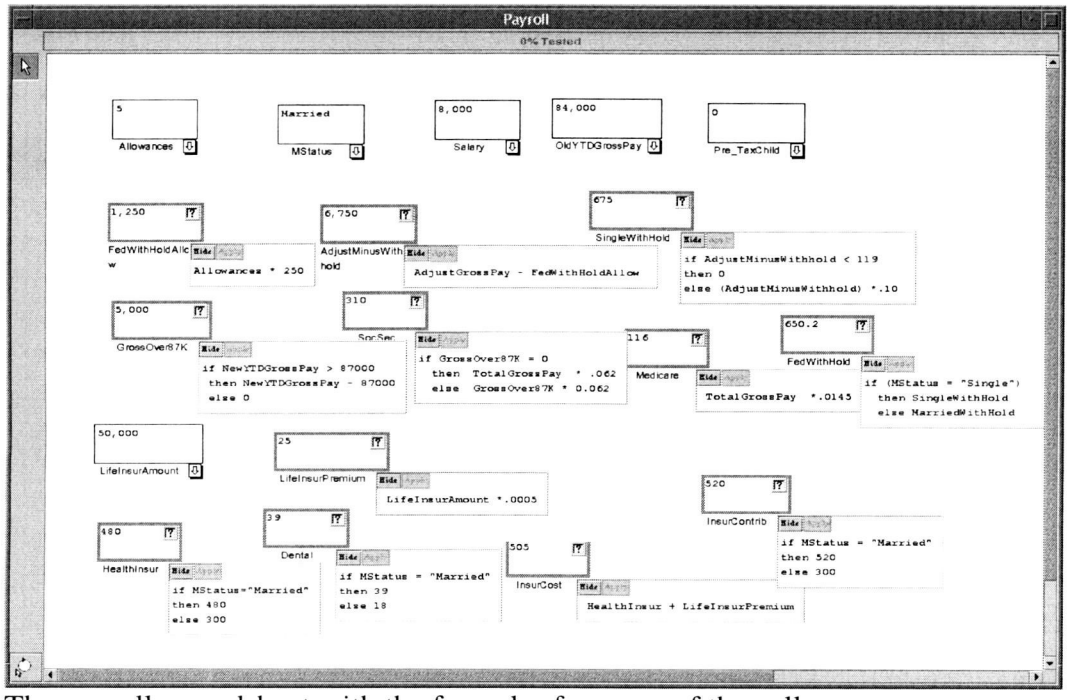
_____

_____

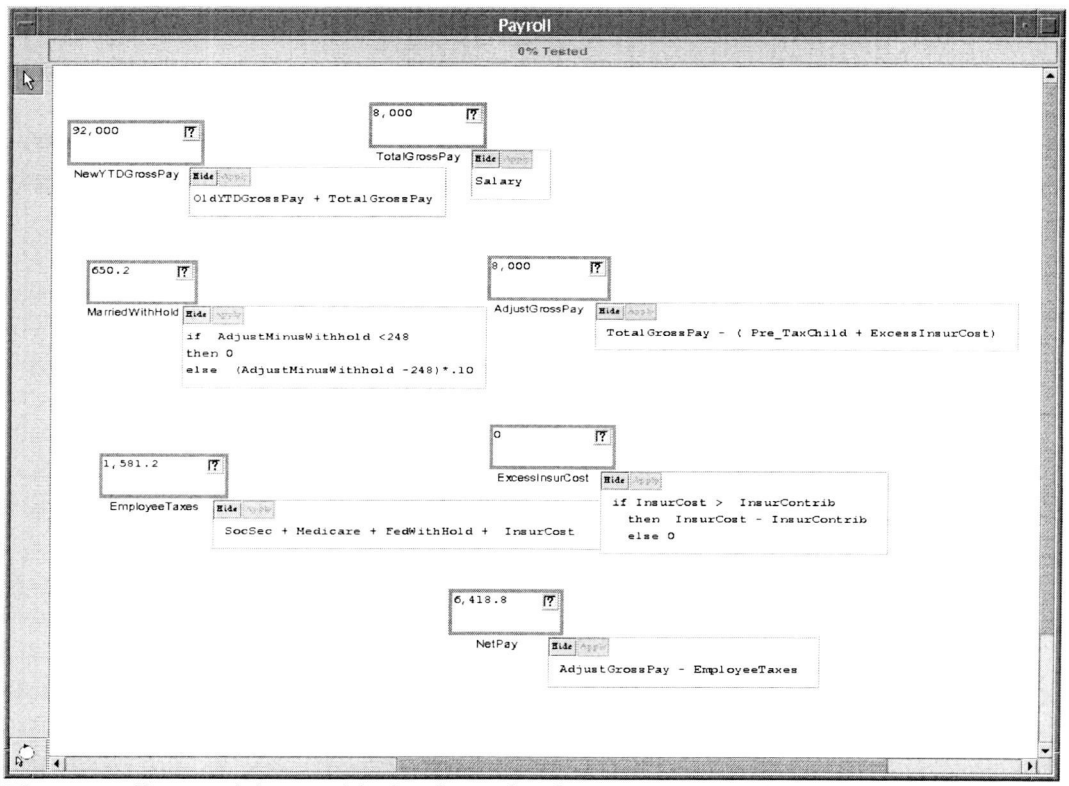
Appendix C: Experiment Spreadsheet Formulas



The grades spreadsheet with all the formulas.



The payroll spreadsheet with the formulas for some of the cells.



The payroll spreadsheet with the formulas for remaining cells.