AN ABSTRACT OF THE THESIS OF

Vijay B. Krishna for the degree of Master of Science in Computer Science presented on April 6, 2001.

Title: Empirical Studies of a Spreadsheet Maintenance Experiment.

Abstract Approved: Redacted for Privacy

ook

Spreadsheet language programs, which include commercial spreadsheets, are among the most common form of software in use today. Unlike more "traditional" forms of software however, spreadsheet language programs are created and maintained by end-users with little or no programming experience. As a result, a high percentage of these programs contain errors. Unfortunately, software engineering research has for the most part ignored this problem. We have developed a methodology that is designed to aid end-users in developing, testing, and maintaining spreadsheet language programs. The methodology communicates testing information and information about the impact of cell changes to users in a manner that does not require an understanding of formal testing theory or the behind the scenes mechanisms. In this thesis, we present empirical data about the methodology's effectiveness resulting during a spreadsheet maintenance experiment. The results show that, during maintenance end-users using our methodology were more accurate in making changes and did a significantly better job of validating their spreadsheets than end-users without the methodology.

Empirical Studies of a Spreadsheet Maintenance Experiment

by

Vijay B. Krishna

A Thesis

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented April 6, 2001
Commencement June 2001

Master of Science thesis of Vijay B. Krishna presented on April 6, 2001

APPROVED:

Redacted for Privacy

Major Professor, representing Computer Science

Redacted for Privacy

Chair of the Department of Computer Science

Redacted for Privacy

Dean of the 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

Vijay B. Krishna, Author

ACKNOWLEDGEMENTS

I would like to thank my major professor Dr. Curtis Cook, minor professor Dr. Prasad Tadepalli, and committee members Dr. Greg Rothermel and Dr. Robert Higdon.

My thanks also goes to all the committee members of the Forms/3 research group, especially Joshua Cantrell and Daniel Keller, my research associates during the experiment and Chris Wallace, Bing Ren, Jay Summet and Andrew Ko for helping us in fixing any problems we managed to create.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

EMPIRICAL STUDIES OF A SPREADSHEET MAINTENANCE EXPERIMENT

Chapter 1
INTRODUCTION

Spreadsheet languages are among the most common form of software in use today. They are used for computational tasks ranging from simple "scratchpad" applications developed by single users to large-scale, complex systems developed by multiple users [25]. The relative simplicity of the spreadsheet paradigm lets end-users with little or no formal programming background quickly automate a wide variety of computational tasks. These spreadsheet programmers create spreadsheets which play an influential role in decisions about investments, budgets, taxes and a lot of other important issues. Considering the fact that spreadsheets play such an important role in our daily lives, it is rather surprising that almost no work has been done to help in software engineering tasks that arise in the creation and maintenance of spreadsheets.

Spreadsheets are not just mechanisms for organizing and displaying data: rather, they are *programs* that use formulas to transform inputs into outputs. Moreover, like programs in imperative languages, spreadsheets often contain errors. A survey of the literature [25] provides several examples: in four field audits of operational spreadsheets, errors were found in an average of 20.6% of the spreadsheets audited; in eleven experiments in which participants created spreadsheets, errors were found in an average of 60.8% of those spreadsheets; in four experiments in which the participants inspected spreadsheets for errors, the participants missed an average of 55.8% of those errors. Two large auditing firms reported finding errors in 90% of the spreadsheet financial models they reviewed [25]. Such errors can have serious

consequences; for example, a Dallas oil and gas company lost millions of dollars in an acquisition deal because of spreadsheet errors [24].

Compounding these problems, spreadsheet programmers exhibit unwarranted confidence in the correctness of their spreadsheets and their modifications [5, 26, 40]. For instance, Brown and Gould [5] studied 9 experienced spreadsheet users with an average of 2.7 years and eight hours per week of experience. Although these subjects were quite confident that their spreadsheets were accurate, 44% of the spreadsheets they created contained errors and every subject made at least one error.

Spreadsheet errors can often be traced to problems in spreadsheet "development"; however, it is widely acknowledged that spreadsheet users also report difficulties with "maintenance" tasks such as understanding, debugging, modifying, and enhancing spreadsheets [9, 26, 28]. Given the ubiquity of spreadsheets and the importance of the tasks they perform, we would like to help spreadsheet users avoid such difficulties. One natural place to seek such help is the software engineering community. Most previous research in software engineering, however, has addressed problems faced by professional software engineers using imperative languages; comparatively little research has addressed problems involving the "engineering" of spreadsheets. Moreover, in our search of the literature, with the exception of a few papers addressing problems in spreadsheet debugging and auditing [9, 30], we find no research directly addressing problems related to spreadsheet maintenance and evolution. This is particularly distressing because, like successful imperative programs, successful spreadsheets undergo evolution, adaptation, perfection, and correction, often in the hands of users other than their initial developers.

Therefore, we have been investigating the possibility of bringing some of the benefits of formal software engineering techniques to the creators and maintainers

of spreadsheets. Our initial focus has been on the testing and debugging of spreadsheets, and one outcome of this work has been our "What You See Is What You Test" (WYSIWYT) spreadsheet testing methodology [33, 34]. The WYSIWYT methodology provides feedback about the "testedness" of a spreadsheet so that its creators will be motivated to test their spreadsheets more thoroughly. The methodology has been designed, however, to function incrementally, as formulas and data are added to, deleted from, and modified in a spreadsheet, using impact analysis to determine where revalidation is needed. Thus, the methodology is expected to support not just the initial development of spreadsheets, but also their evolution and maintenance. Also, most of the spreadsheet programmers do not have any pre-requisite software engineering background. The methodology takes this fact into consideration and does not require the programmers to have a formal software engineering background to use the methodology.

Given this methodology, determining whether it can be used in a way that brings any benefit to programmers requires answers to three questions:

- Is the methodology efficient enough to coexist with the immediate visual feedback of values in a spreadsheet? Rothermel, et al. [34] showed that most of the algorithms used in the methodology are implemented in ways that adds only O(1) to the existing cost of maintaining the interactive environment.
- Will the methodology uncover faults in the programs? The methodology guides the programmers in meeting a dataflow test adequacy criterion, which will be described in chapter 2. Rothermel, et al. [34] empirically studied the fault detection characteristics of test suites that met this criterion. Their results suggest that such test suites can provide fault detection rates for spreadsheets at a significantly higher rate than equivalently sized randomly generated test suites [33, 34].

- Will the programmers who use the methodology be more effective, more efficient, and less overconfident about their maintenance and testing activities than programmers who do not use the methodology? To investigate this question, we began a series of empirical studies with human programmers. This thesis describes one of these studies that we conducted to access the usefulness of our methodology in a spreadsheet maintenance task.

To date, studies on the WYSIWYT methodology [32, 33] have focused on its application to complete spreadsheets. These studies have shown that the methodology can help spreadsheet users test their spreadsheets more effectively and more efficiently. Thus far, however, we have not empirically investigated the methodology in the context of spreadsheet maintenance. Furthermore, the studies have focused on computer science students, who represent only a specialized subset of spreadsheet users. To more generally assess the potential of the WYSIWYT methodology, we require studies of end-users engaged in maintenance tasks.

In Chapter 2, we will review the literature and provide background information on software maintenance, spreadsheet paradigm, errors and overconfidence, Forms/3, the WYSIWYT environment and the related work. Chapter 3 describes the experimental hypothesis, experiment design, subjects and the experimental task. In Chapter 4, the results of the experiment are discussed. Chapter 5 is a discussion on the usefulness of the feedback devices. In Chapter 6, we present the threats to validity and conclusions.

Chapter 2
BACKGROUND

In this chapter we will present the background material on software maintenance, spreadsheet languages, spreadsheet errors and the overconfidence of spreadsheet programmers. We then review the WYSIWYT methodology developed to help reduce errors and the associated overconfidence. Finally, we describe the Forms/3 spreadsheet language in which the methodology was prototyped for our experiments.

2.1 Software Maintenance

Software maintenance can be viewed as a continuation of the software development process. It is expected that after a program is released, changes will be made to the program to correct errors or to add new features requested by users or to make the program compatible with changes in the operating environment. For example an income tax program is expected to change each year because the tax laws change each year. Because programs do not wear out like physical things such as machinery or highways and programs typically undergo a considerable number of changes over time, the term "software evolution" is often used in place of software maintenance.

Maintenance can be defined as the process of recording and tracking field-based problems and requests, and managing the responses. A problem does not necessarily imply a software defect. Problems arise simply because the expected behavior of the system does not match the actual system behavior. This might be the result of a software defect, but it could just as easily be the result of misunderstood system capabilities, incorrect usage workflow, poor documentation, change of business directive or a thousand other reasons.

A good software maintenance infrastructure that incorporates correcting software problems in a timely and high quality fashion requires that certain practices be put into place. Some of these practices include:

- Problems can easily be reproduced - this requires high quality information about the environment in which the problem occurred.
- Problems can be communicated to the development team in a timely manner (e.g. minutes or hours).
- Code changes can be correlated with the problems.
- Regression tests can be augmented with additional tests to guarantee correct fixes.
- Fixes can be deployed at controlled intervals.

Software maintenance tasks are typically grouped into four categories: Corrective, adaptive, perfective and preventive. Corrective maintenance involves fixing program errors. Adaptive maintenance involves making changes because of changes in the environment or system, such as a change in the operating system, or hardware. Perfective maintenance is making changes in response to user requests or to improve the software such as making it more efficient. Preventive maintenance is making changes to prevent failures or make future changes easier. It is not unusual to expend over twice as many resources during program maintenance as were expended during development of the original program.

2.2 Spreadsheet Languages and Spreadsheet Paradigm

We use the term spreadsheet languages to refer to all systems that follow the spreadsheet paradigm, from commercial spreadsheets to more sophisticated systems whose computations are defined by the cells' formulas and the cell's value is defined solely by the formula explicitly given to it by the user [15]. Spreadsheet languages differ from most other commonly used programming languages in that

they provide a declarative approach to programming, characterized by a dependence-driven, direct manipulation working model [2]. Users of spreadsheet languages create cells and define formulas for those cells. These formulas reference values contained in other cells and use them in calculations. As soon as a cell's formula is defined, the underlying evaluation engine automatically calculates the cell's value and the values of affected cells (at least those that are visible), and immediately displays the new results. This feature is called the immediate feedback part of the spreadsheet paradigm, wherein the new values are computed as soon as the cell's formula is entered and the new values are displayed immediately.

2.3 Errors and Overconfidence

Despite the end-user appeal of spreadsheet languages and the perceived simplicity of the spreadsheet paradigm, there are problems that plague spreadsheets and spreadsheet programmers: errors, overconfidence about accuracy, poor design, and lack of documentation [5, 9, 25, 31, 37].

The errors in the spreadsheet can be in the cells' formula or because of a cell referencing a wrong cell. The following are just a few of the stories of the effects of spreadsheet errors [27]:

- A Midwestern firm's estimated taxes were $5,000 off from a correct paper and pencil value. The spreadsheet had an incorrect formula for assessing salvage value [7].
- Eric Klasson, a Houston consultant with Price Waterhouse, audited 4 spreadsheet models. He found 128 errors covering 120 line items. Some formulas were applied directly to two subsidiaries. The spreadsheets had already been in use for months [10].

- In a North Carolina election, results of an election were about to be incorrectly posted. An election official, using a calculator, detected an inconsistency. Examination found an incorrect cross-tabulation in the spreadsheet being used to post the results [41].

Panko and Halverson [23] also analyzed the types of errors that subjects made while developing a spreadsheet. They found out that three types of errors were common. Mechanical errors are simple mistakes, such as mistyping a number or pointing to a wrong cell. Logic errors involve entering a wrong formula because of the mistake in reasoning. The most dangerous type of error is the omission error, in which something is left out. These types of errors are very difficult to detect [1, 3, 42].

Panko [25] investigated the associated problem of overconfidence of spreadsheet developers in his survey on spreadsheet errors. He cites the Brown and Gould experiment [5] in which 21% of the spreadsheets contained errors and yet the developers were "extremely overconfident in the accuracy of their spreadsheets", as well as findings from several other experiments in which the participants had high confidence in accuracy despite the presence of errors in their spreadsheets.

Poor design and lack of documentation are the other associated problems that plague spreadsheets. For many users, the spreadsheet program represents the first hands on experience with a computing device, programming, and documentation. In general, these users have not been trained in systems analysis and tend to overlook the concerns of the professional systems analyst in designing a system, such as reliability, auditability, and control. In fact, the typical spreadsheet user is happy to avoid systems [31].

2.4 The WYSIWYT Methodology

The literature on program testing primarily addresses the testing of imperative programs [11, 13, 14, 20, 29, 39], with a few attempts to address the testing of functional and logic programs [4, 17, 19, 22]. However, there are three major differences between the spreadsheet and the imperative language paradigms that directly affect the development of testing methodology for spreadsheets. First, evaluation of spreadsheets is driven by data dependencies between cells, and spreadsheets contain explicit control flow only within cell formulas. A methodology for testing and maintaining spreadsheets must be compatible with this dependence-driven evaluation model and not rely upon any particular evaluation order. Second, spreadsheets are developed incrementally and there is an immediate visual response after each addition of or modification of a cells' formula. A testing methodology for spreadsheets must be flexible enough to operate upon partially completed programs and efficient enough to support impact analysis. Third, and most critical, whereas most imperative programs are developed by programmers, spreadsheets are developed by a variety of users, many of whom have no training in formal software engineering principles. Rothermel et. al.'s [34] 'What You See Is What You Test' (WYSIWYT) methodology takes all these factors into account. We briefly describe the foundations (or basics) of that methodology here; a detailed presentation can be found in [34, 35].

The WYSIWYT methodology relies, behind the scenes, on code-based test adequacy criteria. Code-based test adequacy criteria provide help in selecting test data and in deciding whether a program has been tested "enough" by relating testing effort to coverage of code components. Such criteria have been well researched for imperative languages (e.g. [13, 20, 29]), and several empirical studies (e.g. [12, 14, 39]) have demonstrated their usefulness.

The WYSIWYT methodology incorporates a test adequacy criterion adapted from the *output-influencing-all-du-pairs* dataflow adequacy criterion defined originally for imperative programs [11]. This criterion, which we call *du-adequacy* for brevity, focuses on the definition-use associations (du-associations) in a spreadsheet, where a du-association links an expression (in a cell formula) that defines a cell's value with expressions in other cell formulas that reference (use) the defined cell. The criterion requires that each executable du-association in the spreadsheet be exercised by test data in such a way that the du-association contributes (directly or indirectly) to the display of a value that is subsequently pronounced correct (validated) by the programmer.

It is not always possible to exercise all du-association; those that cannot be exercised are called *nonexecutable*. Determining whether a du-association is executable is provably impossible in general and frequently infeasible in practice [14, 39]; thus, data flow test adequacy criteria typically require that test data exercise (cover) only executable du-associations. Our criterion does the same. In our experience with spreadsheets, however, most of the nonexecutable du-associations we have encountered have involved direct contradictions between conditions that are relatively easy for persons capable of creating those spreadsheets to identify.

The appropriateness of the du-adequacy criterion for spreadsheets stems from the fact that by relating test coverage to interactions between definitions and uses of cells, the criterion requires these interactions to be exercised. Since such interactions are a primary source of errors in spreadsheets, this is valuable. Moreover, by linking test coverage to cell validation, the criterion avoids problems in which du-associations influencing only values that are hidden or off-screen are considered exercised simply by applying test inputs; instead, the du-association must participate in producing a visible result judged correct by the programmer. The criterion also facilitates the incremental validation of spreadsheets, allowing a

test to involve entry of values into a small subset of the potentially large set of input cells in a spreadsheet, followed by validations of multiple cells. Finally, the criterion facilitates impact analysis: when a spreadsheet programmer changes a formula, the information used to support the criterion can be used to calculate the du-associations added or potentially affected by this modification. The system can then require their (re) validation.

Key to the usefulness of the WYSIWYT methodology, however, are devices for communicating testing and impact information to the spreadsheet user, in a manner that does not require understanding of formal testing theory. These visual devices are easy to understand and use and do not require any software engineering background in order for their use. These visual feedback devices are described in detail in the next section.

2.5 Using WYSIWYT Methodology in Forms/3

We have prototyped our WYSIWYT methodology in the research language Forms/3 [6], one of many spreadsheet language research systems (e.g. [6, 8, 18, 21, 36, 38]). This choice is motivated partly by the fact that we have access to the implementation of Forms/3, and thus, we can implement and experiment with various testing technologies within that environment. More important, however, is that by working with Forms/3, we can investigate language features common to commercial spreadsheet languages as well as advanced language features found in research spreadsheet languages.

As in other spreadsheet languages, a Forms/3 spreadsheet is a collection of cells; each cell's value is defined by the cell's formula. But, unlike in traditional spreadsheet languages, Forms/3 cells need not be elements of grids. A Forms/3 user can place the individual cells in the form anywhere on the form.

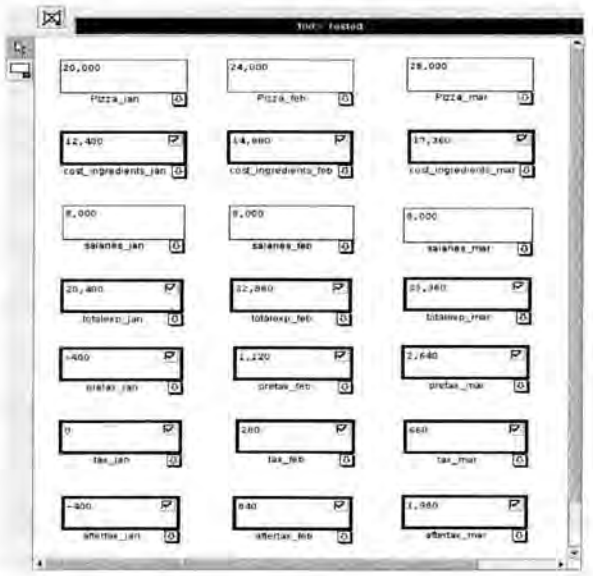


Figure 1: Pizza sales spreadsheet

Each cell has a formula as well as some visual attributes controlling its appearance, and the program's outputs are entirely determined by the combination of these formulas and attributes. A cell's value is the result of the execution of the formula. The value is well defined prior to computation (since it is simply the result of the formula); however, Forms/3 is a lazy evaluation language, and hence each value is actually computed only as needed, and may be saved or discarded according to any arbitrary caching strategy.

The programmer enters the cell's formula and receives an immediate feedback about the cell's value. Figure 1 shows an example of a Forms/3 spreadsheet. The figure depicts a financial spreadsheet that represents the cashflow projection model for a "Pizza Parlor" consisting of information about sales, expenses and final cashflow for three months. Forms/3 spreadsheets are not restricted to fixed grid of cells and we can give the cells meaningful names. For example, the cell representing the cost of ingredients for January is called **cost_ingredients_jan**.

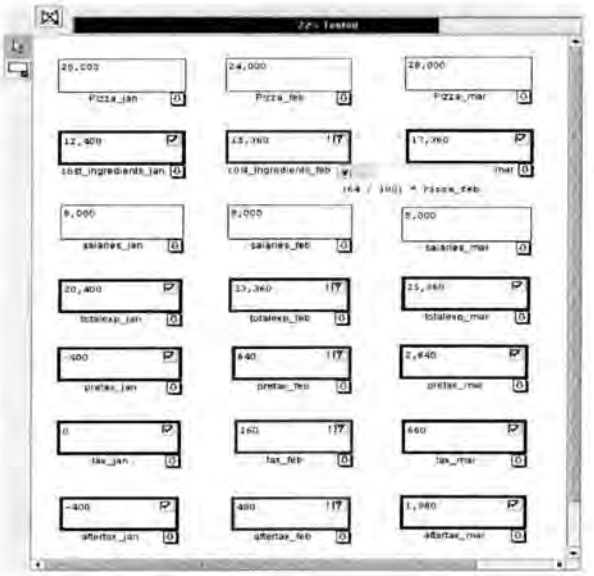


Figure 2: Pizza spreadsheet with the new formula for cost_ingredients_feb

The cells in the spreadsheet can be categorized as input cells or output cells, depending on whether they take inputs, or calculate values based on other cell values. In the figure, **Pizza_jan** is an input cell (thin border) and **cost_ingredients_jan** is an output cell (thick border). Cell formulas may be hidden or displayed.

In figure 2, the formula for cell **cost_ingredients_feb** is displayed. The underlying validation algorithm is given in [33]; the overall notion is that it recurses back through the du-associations that affect, directly or indirectly, the currently computed value of **totalexp_feb**, and marks them tested[1].

The visual feedback devices in the methodology keep the user continually informed about the testedness of the spreadsheet. The testedness is computed by relating the

[1] Formally, du-adequacy does not define test adequacy at the granularity of the cells. When we refer to a cell being tested, it is shorthand for saying that all the du-associations whose uses are in that cell have contributed to values that the programmer has pronounced correct.

testing effort to the coverage of code components. The various visual feedback devices in the WYSIWYT methodology are cell border colors, arrows, checkboxes, and percent tested indicator. The cell border color indicates the testedness of a cell. The system depicts a fully tested cell with blue borders (black in this paper), an untested cell with red borders (light gray), and a partially tested cell with borders in various shades of purple (darker gray). The programmer can choose to display arrows showing interactions between some of the cells. The arrows follow the same color scheme as the cell borders. We provide additional testing information through the marks in the cell's checkbox: an exclamation point means that validating that cell's value will increase the spreadsheet's testedness, a question mark indicates that some previous value for this cell was validated, a blank indicates that no opinion has been recorded about this cell's value, and a check mark indicates that the user's validation was recorded. Finally, additional testedness information is provided by the "percent tested" indicator at the top of the window which shows the percent of du-associations that have been tested.

Suppose a programmer has created the Forms/3 spreadsheet of Figure 1. During this process, the underlying evaluation engine has not only been displaying cell values, but has also been calculating the du-associations that come into existence as new formulas are created, and tracking the du-associations that influence calculations. Using this information, visual feedback devices keep the programmer continually informed of testedness status, draw attention to untested sections of the evolving spreadsheet, and suggest where testing activity will provide new progress (according to our adequacy criterion).

For example, suppose the spreadsheet programmer now decides to change the formula for **cost_ingredients_feb**, so that the cost of ingredients is now 64% of the pizza sales amount for February. As soon as the programmer enters the new formula and clicks on the apply button in the formula window, the system updates

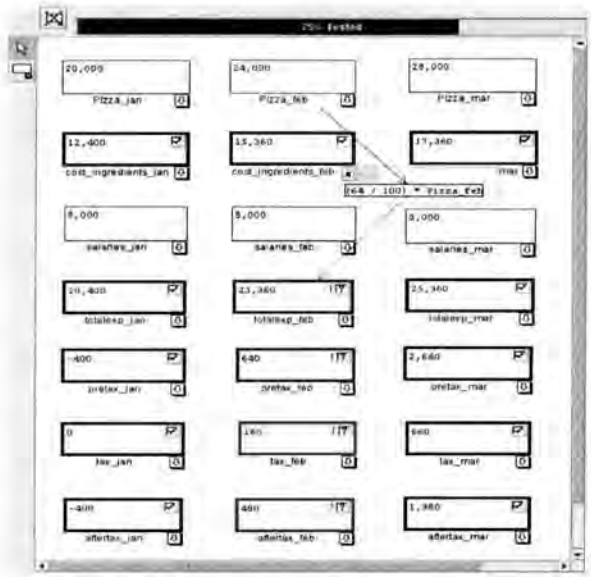


Figure 3: Pizza sales spreadsheet after validating the value for cost_ingredients_feb.

all new cell values and responds with the immediate visual feedback on the new testedness of each affected cell, as well as for the whole spreadsheet as shown in figure 2.

The spreadsheet programmer now decides to turn on the arrows for **cost_ingredients_feb** cell and decides that this cells' displayed value is correct, given the value of the cells it depends on, and clicks on the checkbox in the upper right corner of that cell to validate it. The system responds with immediate visual feedback as to the new testedness of each visible cell and arrow, as well as the percent testedness for the whole spreadsheet, as shown in Figure 3.

Suppose the programmer has validated a large percentage of the du-associations in the spreadsheet, turning most borders and arrows blue, and suppose that the programmer decides to alter a formula - either to correct an error or to enhance the spreadsheet. When the programmer completes the alteration, the underlying

evaluation engine uses static information about du-associations to walk forward through cells that use, directly or indirectly, the modified cell, and mark those du-associations "not validated". Having completed this impact analysis task, the engine updates the border and arrow colors and other indicators to reflect the new testedness status of the spreadsheet. This act draws the programmer's attention to the need to re-validate interactions newly created, or potentially impacted, by the modification.

2.6 Related Work

Unfortunately, of the large amount of work that has been done in the area of software maintenance, little of it is in the area of spreadsheet program maintenance. Our research [30, 33, 34] has so far focused on how the WYSIWYT methodology benefits spreadsheet programmers in their testing and debugging activities, but we have not investigated whether it might benefit a programmer in a spreadsheet maintenance activity. All our empirical studies have focused on determining how useful the WYSISYT methodology is, in testing spreadsheets. These experiments [20] showed that WYSIWYT methodology helps users test their spreadsheets more effectively and more efficiently and aids in reducing their overconfidence. The subjects in all of these previous experiments were students with some computer science background. Two questions that we had in our mind were, whether this methodology would be useful in a spreadsheet maintenance task and would the end-users with little or no computer science background, be able to use the methodology easily and effectively. To seek an answer to these questions, we conducted a controlled laboratory experiment to see if the WYSIWYT methodology is helpful in maintaining spreadsheets. In the experiment, which is described in the next chapter, the subjects were business students and their task was to make changes to a financial spreadsheet model.

Chapter 3
THE EXPERIMENT

In this chapter we describe the design of the spreadsheet maintenance experiment: the participants, the tutorial, the tasks and materials. Much of the information in this chapter can also be found in [10].

3.1 Design of the Experiment

The objectives of our study were to investigate the following research questions:

RQ1: Are end-users who use the WYSIWYT methodology more accurate in performing spreadsheet modifications than end-users who use an ad-hoc approach?

RQ2: Are end-users who use the WYSIWYT methodology during a maintenance task more effective in testing modified spreadsheets than end-users who use an ad-hoc approach?

RQ3: Are end-users who use the WYSIWYT methodology during a maintenance task less overconfident about the accuracy of their spreadsheets than end-users who use an ad-hoc approach?

RQ4: Are end-users who use the WYSIWYT methodology during a maintenance task less overconfident about the quality of their testing of modified spreadsheets than end-users who use an ad-hoc approach?

These questions were translated directly into hypotheses. We also took care that the design of our experiment would provide insight into the following question:

Is training in the underlying test adequacy criterion and its relationship to the visual devices needed in order for the spreadsheet maintainer to perform more effectively when using our methodology?

The following sections will describe the details of the experiment and our analysis of the data collected during the experiment.

3.2 Subjects

Since we were investigating whether typical spreadsheet end-users could effectively use the WYSIWYT methodology, we decided to draw subjects from a pool of business school students, because we believed they would be representative of spreadsheet end-users and have the business domain knowledge required to understand many typical spreadsheet applications. We selected 36 students who were either currently enrolled in or had recently taken a sophomore business course in which they worked with spreadsheet financial models. The subjects were a mix of sophomore and junior business majors with varying levels of spreadsheet and programming experience. We randomly partitioned these students into two groups: a control ("Ad Hoc") group of 17 students that did not have access to our WYSIWYT methodology, and a treatment ("WYSIWYT") group of 19 students that did have access to the methodology. (The difference in the group size was due to a few subjects failing to arrive for their appointment.)

To ascertain whether the subjects had reasonably similar backgrounds, we administered a questionnaire that asked about their academic background and experience with spreadsheets and programming [Appendix A]. A summary of the analysis of the responses is given in Table 1. A subject was credited programming experience if he or she had a high school or college class or professional experience in programming.

Group	Number of subjects	Average GPA	Subjects with programming experience	Subjects with spreadsheet experience	Subjects with exposure to experimental environment
Ad Hoc	17	3.09	8	17	0
WYSIWYT	19	3.35	10	19	1

Table 1: Subject group demographics.

A subject was credited with spreadsheet experience if he or she had created a spreadsheet for a high school or college class, or for professional or personal use. As Table 1 shows, there is little difference in the collective backgrounds of the two groups. Only one subject reported experience using Forms/3. Our statistical analysis showed homogeneity between the two groups.

3.3 Procedure

To investigate our research questions, we conducted a controlled laboratory experiment, during which the subjects modified and tested a spreadsheet. Almost half of the subjects ("WYSIWYT" subjects) did so using the Forms/3 spreadsheet environment that included the WYSIWYT methodology, and the rest ("Ad Hoc" subjects) used the same environment minus the WYSIWYT methodology.

The experiment was conducted in a small lab with six workstations running Windows NT with the subjects seated one per workstation. The experiment began with a 25-minute tutorial on Forms/3 in which each subject actively participated by working with several example spreadsheets on their individual workstations (The tutorial is described further in Section 3.3). To control for amount of exposure to the environment, both groups of subjects were given identical amounts of training time.

Subject data was collected during the experiment from pre- and post-problem questionnaires [Appendix A, G, H] and from electronic transcripts that recorded all on-line modification and testing activities. The pre-problem questionnaire measured the subjects' understanding of the Forms/3 problem and the post-problem questionnaire measured the subjects' perceptions of how well they had performed the modification and testing tasks. The post-problem questions for the WYSIWYT subjects also included questions about their understanding and use of the feedback devices provided by the WYSIWYT methodology.

3.4 The Tutorial

During the 25-minute tutorial on Forms/3, each subject worked with example spreadsheets on their workstation following instructions given by the lecturer. The tutorial introduced basic language features (e.g. basic syntax of formulas) and environmental features (e.g. how to create, delete, and edit cells) that would eventually be used in the spreadsheet modification task. Throughout the tutorial the subjects had access to a hardcopy quick reference guide [Appendix B] to the features they were being taught. They could make notes on the handouts, which remained available to them throughout the experiment. An assistant was available to answer questions.

In the tutorial, testing was described as a process of trying different input values and recording decisions about the correctness of values in output cells. The subjects were told that the decision about a cell's value can be recorded by clicking on its checkbox. All subjects were asked to record decisions only for those output cells whose value seemed correct to them. The Ad Hoc subjects were told that the spreadsheet would flash whenever they clicked on a checkbox indicating that their decisions have been recorded. The tutorial for the WYSIWYT subjects included basic instructions on the use of cell border colors, arrows, percent-tested indicator,

and checkboxes. Because one of the goals of the WYSIWYT methodology is that the user need not acquire an understanding of formal testing theory, we explained only that red means "not tested", blue means "fully tested", and purple means "partially tested". For the symbols inside checkboxes, the blanks were described as meaning "a testing decision has not been recorded", check marks as "you have made a decision for this cell's value for the current set of input values", the question mark as meaning "you have made a decision for this cell's value for some previous input values", and the exclamation point as meaning "by checking this cell's checkbox you will increase the testedness of the spreadsheet". We did not mention the underlying concepts of du-associations or impact analysis, nor did we describe non-executable du-associations. Following these explanations, the subjects were given unstructured time to practice their Forms/3 skills.

Regardless of which group a subject was in, the total training time was identical; subjects not receiving explanations of the methodology's feedback were given more unstructured time to practice using Forms/3. It was important to equalize the total time, because the additional instructions that the WYSIWYT subjects received provided them with additional practice in Forms/3, and this additional practice time could have confounded the results. Since the subjects had little or no previous exposure to the Forms/3 environment, at the conclusion of the tutorial the subjects could be considered equal in their knowledge of Forms/3.

3.5 Tasks and Materials

The subjects were given the experimental task once the tutorial was completed. Choosing an experimental problem for this study was a big challenge. We did not want the experimental problem to be too easy or too hard as that would have had a significant impact on the quality of the results - we may not have been able to seek answers to our hypothetical questions; thus we wanted an experimental problem that the subjects would be able to understand easily. We chose a financial

spreadsheet model depicting the cashflow projection for a pizza restaurant, as our experimental problem (Figure 1). The model included the pizza sales for each of three months (January, February, and March) and the expenses (cost of pizza ingredients, worker's salaries). The pretax cash flow was the difference between sales and expenses. Taxes were deducted from the pretax cash flow to yield an after tax cash flow. The model was similar to a spreadsheet model the subjects had encountered in a sophomore level business class. A complete description of the problem and materials is included in [Appendix D].

We instructed the subjects to read the description of the spreadsheet they were about to work with. The description included details on the cash flow projections for the pizza restaurant. We then administered a five question comprehension quiz [Appendix E], so that we could assess the subjects' basic understanding of the problem and whether they could relate parts of the description of the model to the cells in the Forms/3 spreadsheet. (Over three-fourths of the subjects answered all five questions correctly.)

Following this quiz, we gave the subjects their experimental task [Appendix F], which involved making changes to the pizza restaurant model. In brief, the subjects were asked to (1) modify the tax rate for each of the three months covered by the model, (2) add spaghetti (sales and cost of ingredients) to the restaurant's menu for the last two of the three months, and (3) add an additional worker for the last two months. To encourage the students to test the modified model, the task description indicated that the initial unmodified spreadsheet model was correct and that it was important that the modified spreadsheet not contain errors. The subjects were given 15 minutes to make the modifications and to verify that the spreadsheet was working correctly. The subjects completed the post-problem questionnaire after the modification task.

Chapter 4
RESULTS AND ANALYSIS

In this chapter, we present results of the experiment and the analysis of the results. The results and analysis are based on the transcripts and the questionnaires that we collected from the subjects.

4.1 Accuracy

Our first research question considers whether using our methodology helped the subjects make the modifications more accurately. The modification was scored on an 18-point basis according to the following:

- Tax rate change (1 point per month for a total of three points): The subjects were required to change the tax rates for each of the three months from 25% to 28%. Changing the tax rate correctly was worth 1 point for each month.
- Adding spaghetti to the menu (4 points/month for a total of 12 points): There were many ways the subjects could add spaghetti to the menu. But at an abstract level, all these ways reflected two major changes for each month and each of these changes was worth two points. These changes included changing the pretax amount for the months of February and March to include the spaghetti sales and changing the total expenses for these two months to include the cost of spaghetti ingredients. Not making any changes to the pretax amount and total expenses for the month of January was worth 4 points.
- Adding additional worker (1 point per month for a total of 3 points): Adding the additional worker for the months of February and March was worth 1 point each and not adding the worker for the month of January was also worth 1 point.

Group	Tax Rate (3 points)	Spaghetti (12 points)	Worker (3 points)	Total Modification
Ad Hoc (n=17)	2.71 (.69)	9.94 (2.1)	2.94 (.24)	15.6 (2.1)
WYSIWYT (n=19)	2.79 (.63)	10.74 (1.6)	2.63(.83)	16.2 (2.7)

Table 2: Average modification scores for each subtask and the total modification task.

The average modification scores for the Ad Hoc and WYSIWYT subjects for each modification subtask are given in Table 2. The average total modification score was higher for the WYSIWYT subjects but not significantly higher (Mann-Whitney, $p = 0.1681$). On further examination of the data, however, we discovered that one WYSIWYT subject had scored only 7 out of 18: over 3 standard deviations below the mean and considerably below any other WYSIWYT subject. With this outlier removed, the WYSIWYT subjects' total modification scores were significantly higher than the Ad Hoc subjects' modification scores (Mann-Whitney, $p = 0.0861$).

Based on the average scores from each of the modification tasks, the spaghetti modification seemed to be the most difficult of the three tasks. 22 out of the 36 subjects made at least one mistake in the spaghetti modification and lost at least one point. Compared to this, only 5 subjects committed mistakes in each of the other two modification tasks. Table 3 shows the number of subjects from each group who made at least one mistake in their modification task.

Group	Tax Rate	Spaghetti	Worker
Ad Hoc (n = 17)	3	12	2
WYSIWYT (n = 19)	2	10	3

Table 3: Number of subjects who made an error in each of the modification tasks.

4.2 Testing the changes made

A second research question considers whether the subjects with the WYSIWYT methodology were more effective in their testing. Recall that, to encourage all subjects to test their spreadsheets after making changes, the modification task description for both groups indicated that the original spreadsheet had been thoroughly tested and that the modified spreadsheet needed to be correct.

Each click in a checkbox to record a decision about a cell's value by the subject was considered to be a test. Table 4 shows that our methodology provides strong encouragement to test the modified spreadsheets: significantly more WYSIWYT subjects than Ad Hoc subjects executed at least one test (Fisher's Exact Test, df=1, p=0.0004).

Group	No tests	At least one test
Ad Hoc (n=17)	12	5
WYSIWYT (n=19)	2	17

Table 4: Number of subjects executing no tests and executing at least one test.

Also, the five Ad Hoc subjects who executed at least one test executed an average of 8.4 tests while the 17 WYSIWYT subjects who executed at least one test executed an average of 31.7 tests. Because the testing difference between the two groups of subjects was so dramatic, we wondered whether some of the Ad Hoc subjects might have "tested" by visually inspecting cell formulas for different input values, rather than by clicking checkboxes. Our inspection of the transcripts, however, revealed that this had not occurred: after completing the modifications,

only one of the 12 Ad Hoc subjects who did not execute a test made a change to the input cells; further, only 5 of these 12 subjects displayed a cell formula after completing their modifications.

During their testing after completing the modifications, three of the WYSIWYT subjects found and corrected errors; no Ad Hoc subject who executed tests found an error after completing the modifications. Hence our WYSIWYT methodology seems to provide strong visual suggestions to test and helps find errors during maintenance.

4.3 Modification Task Behavior

With few exceptions, the general modification behavior was very similar for subjects in all the groups. They first made the tax rate changes, then the spaghetti changes and finally added the new worker.

Recall as new cells are added to the spreadsheet or as the formulas in cells are changed, the du-pairs created or impacted by these changes are no longer tested and hence the percent of du-pairs covered decreases. Plotting changes to the du-pair by the modifications and testing, revealed three distinct patterns. We classified the subjects into three pattern groups: the "\" pattern group executed no tests so the percent of du-pair covered continually decreased. The "V" pattern group completed all of their modifications before they tested so their percent of du-pairs covered continually decreased during the modifications and then increased during the testing. The "W" pattern group intermixed modifications and testing as the percent of du-pairs covered decreased during modifications and increased during testing. Note that each group contained both Ad Hoc and WYSIWYT subjects. Table 5 shows the number of subjects in each of the pattern groups.

Group	\ group	V group	W group
Ad Hoc	12	3	2
WYSIWYT	2	9	8

Table 5: Number of subjects in each of pattern groups.

When we compared the various pattern groups for accuracy, we found out that the \ group had an average modification score of 15.5, the V group had an average modification score of 16.1 and the W group had an average modification score of 16.2.

The efficiency for the V and W groups was measured on the basis of the number of tests the subjects ran in each of the groups. We did not include the \ group in measuring efficiency because the subjects from this group did not run any tests. The subjects in the V group ran 24.25 tests on an average, while the subjects in the W group ran 29 tests on an average. The average testedness for the subjects in the V group was 76.6%, while it was 81.7% for the subjects in the W group.

We also compared the V and W groups for redundancy, by checking how many redundant tests they ran. Any test that did not increase their percent testedness was considered to be redundant. The subjects in the V group ran 6.5 redundant tests on an average, while the subjects in the W group ran 5.7 redundant tests on an average.

In order to investigate the general modification and testing behavior pattern of the subjects in the W group, we analyzed their transcripts to find out how many times the subjects switched between modification and testing. These subjects switched about 9.8 times on an average between modification and testing.

4.4 Overconfidence

In spite of the high percentage of spreadsheets containing errors, studies [5, 26, 40] have reported that end-users are very confident that their spreadsheets do not contain errors. Experience does not seem to matter as these studies have shown that both novices and experts are overconfident about the correctness of their spreadsheets. One goal of our research is to reduce overconfidence by providing feedback about testedness and about the impact of changes. Recall that participants using our methodology received feedback via changes in cell border and arrow colors, in the "percent tested" indicator, and in checkboxes, after each change to a cell or after each test was executed. To determine the impact of this feedback on overconfidence, after completing the modification task, we asked the participants to answer questions about the correctness of their modification and how well they thought they had tested the spreadsheet.

Spaghetti Modification	Regression Coefficient	Standard Error	t-value	Significance
Ad Hoc	-0.167	0.536	-0.311	0.7602
WYSIWYT	0.28	0.363	0.776	0.4482

Table 6: Regression analysis for spaghetti modification subtask.

The modification self-rating asked them to rate on a 1 ("not confident") to 5 ("very confident") scale how confident they were that each of the modification subtasks (tax rate, adding spaghetti, and new worker) had been completed correctly. Since few subjects made errors in the tax rate and new worker subtasks, we compared only the spaghetti subtask ratings with their spaghetti modification scores. Although Table 6 shows that the regression coefficients (self-rating) for the Ad Hoc and WYSIWYT groups were not significantly different from 0, it does indicate

that the self-rating is a modest predictor for the WYSIWYT group but has no predictive value for the Ad Hoc group.

Group	Tested Self-Rating
Ad Hoc	3.353 (.931)
WYSIWYT	3.000 (.667)

Table 7: Average testing self-rating for Ad Hoc and WYSIWYT group subjects (A=4, B=3, C=2, D=1, F=0).

The post session questionnaires also asked subjects to rate "on a familiar A-F scale" how well they tested their spreadsheet. In spite of the dramatic and significant difference between the number of WYSIWYT subjects who executed at least one test compared to the number of Ad Hoc subjects who executed at least one test, the Ad Hoc subjects' average testing self-rating [Table 7] was higher than the WYSIWYT subjects' average self-rating (Mann-Whitney p=0.1095). Further, of the three pattern groups (group \, group V and group W), the \ group subjects who ran no tests had the highest tested self-rating and W group subjects had the lowest self-ratings. The differences among the three groups were not significant (Kruskal-Wallis, p=0.1515). Table 8 shows the average testing self-rating for all the three pattern groups.

Group	Tested Self-Rating
Group \	3.429(.756)
Group V	3.250 (.622)
Group W	2.700 (.949)

Table 8: Average testing self-rating for the pattern groups.

Chapter 5
DISCUSSION

The strong results from the experiment showed that the subjects receiving the visual testedness and change impact feedback provided by the WYSIWYT methodology were greatly aided and encouraged by it, especially in their testing. One question for which we only have a partial answer is: which portions of the visual feedback were most helpful? To help us begin to answer this question, the WYSIWYT group version of the post-problem questionnaire asked the subjects to rank the helpfulness of each of the feedback devices as very helpful, quite helpful, helpful, somewhat helpful or not helpful. The results, summarized in Table 9, indicate that the subjects found the cell border colors, arrows and percent tested indicator most helpful, checkmarks, exclamation point, and blanks less helpful, and question marks, least helpful.

How helpful were:	Helpful - Very Helpful	Somewhat Helpful or Not Helpful
colored cell border	90%	10%
Arrows	90%	10%
"Tested" indicator	84%	16%
Checkmarks	78%	22%
Exclamation	78%	22%
Blanks	77%	23%
question marks	72%	28%

Table 9: WYSIWYT subjects' helpfulness ratings.

Two-thirds of our subjects' spreadsheets contained errors after the modification was completed (71% for Ad Hoc, 63% for WYSIWYT). This error rate seemed high for a simple task and made us wonder how it compared with error rates in other

experiments. Panko's [25] survey of studies on spreadsheet errors included 12 experiments in which the subjects developed spreadsheets. The error rates in those experiments ranged from 35%-84%. Further, subjects in most of those experiments were business students and they were given hours and in some instances days to complete the development task. Hence, the error rate for our subjects fell into this range even though we gave them only 15 minutes to complete the task.

The tutorial described validation as the process of clicking the checkboxes of the cell whose value seemed to be correct. One question to speculate upon is whether the subjects were actually validating a cell value rather than a cell formula. We analyzed the transcripts to check how many subjects actually pronounced a cell value to be correct even though it had wrong values. The analysis revealed that 7 WYSIWYT subjects and 2 Ad Hoc subjects validated a cell with a wrong value [Table 10]. Three possible reasons for behavior by the subjects are: (1) Since the cells did not have nice numbers, verifying those values by hand would have been a tedious task and hence these subjects assumed the numbers to be correct without actually verifying them with a calculator. (2) They validated a cell based on the formula rather than the value. (3) They interpreted validation as the process of turning cell border blue by clicking checkboxes.

Group	Total number of subjects who ran at least one test	Number of Subjects who had no errors	Number of subjects who tested, but validated erroneous cells	Number of subjects who tested, but did not validate erroneous cells
WYSIWYT	17	7	7	3
Ad Hoc	5	5	2	3

Table 10: Validation information for subjects who ran at least one test.

Based on the scores for the three modification subtasks (change tax rate, adding spaghetti for February and March, and adding an additional worker for February and March), the spaghetti modification was the most difficult (Table 2). A plausible reason for this is that the spaghetti changes are "global" (dependent on other cells) while the other two modifications are "local" (independent of other cells). The tax rate change involved changing the rate in the cell formula from 25% to 28% for each month and the addition of a worker change involved adding $1000 to the salaries formulas for February and March. In both cases, the changes did not depend on or use information from another cell. However, for the spaghetti modification subtask, changes to the revenue, expenses, and pretax cell formulas depended on other cells. By far the most common mistake in this modification as shown in Table 3, was not correctly computing the pretax totals for February and March. Pretax errors accounted for more than half of the errors made by the Ad Hoc subjects and over 40% of the errors made by the WYSIWYT subjects. It would seem that the dependency of the pretax for February and March on the correctness of both the new total revenue and new total expenses might account for such a high error rate.

The higher testing self-rating of the Ad Hoc subjects and the fact that most of these subjects did not execute a single test, suggested comparing testedness ratings of Ad Hoc and WYSIWYT subjects who executed at least one test with subjects who did not. Although not significant, the average rating was 3.43 for the no test group and 3.00 for the test group. There may be several different interpretations of this difference. One possibility is the subjects believed the spreadsheet was so simple that it did not need testing. Another is that end-users are unfamiliar with the concept and importance of testing. Another possibility might be that the subjects were pretty much overconfident about the quality of their work.

It is interesting to compare the testing behavior differences for the V and W groups. The subjects in the W group seemed to be more cautious than the subjects in the V

group. They tested the changes immediately they made them, while the subjects in the V group finished making all the changes before they started testing. The subjects in the W group shifted their attention towards testing as soon as a cell border became red after making some modifications. They seemed to feel the need to change the red border of the cells affected by the change to blue borders before continuing with the other modifications. In contrast, the V group subjects were able to ignore cell border color changes and continued with the modification task.

The subjects were given 15 minutes to finish making all the changes and making sure that their changes were correct. Would they have performed better if they had been given more time? When we analyzed the transcripts, we found out that most of the subjects finished their tasks in less than 15 minutes. Of the 36 subjects, only 7 of them actually used up the whole time. Thus giving them additional time for their modification task would not have changed the results significantly.

The difficulty level of the problem chosen for the experiment seemed to be appropriate. Of the 36 subjects, only 3 felt the need for more time to complete their modifications and 7 felt the need for more time to finish their testing activities. They felt that an additional 5 to 10 minutes would have been sufficient for them to complete the modification and testing tasks.

Chapter 6
THREATS TO VALIDITY AND CONCLUSIONS

In this chapter we consider some of the threats to validity of our investigation and present the conclusion of this thesis.

6.1 Threats to Validity

In this experiment, we addressed the threats to internal validity by balancing the two groups of subjects based on their year in school and their GPAs, by equalizing their training time, and by selecting an experimental problem from a domain which the subjects were familiar with. However, threats to external validity are more difficult to address, given the need to control all other factors. For example, business students represent only a fraction of the spreadsheet programmer population. Similarly, the spreadsheet used in the experiment may not be representative of the population of spreadsheets. Since the main focus of the study was a modification task and testing, the original spreadsheet did not contain any errors. This may be unrealistic as the spreadsheets encountered in a real life situation may contain some errors. However, including errors in the spreadsheet would have confounded the data about modification and testing effectiveness, as the subjects would not be focused on the actual experimental task of modifying and testing the spreadsheet. Moreover, there was an opportunity for the subjects to introduce errors into the spreadsheet by themselves during their modification task. We wanted to see if the subjects could catch such errors with the help of our WYSIWYT methodology.

For scoring purposes, the modification task was broken into abstract subtasks, such as the new total expenses includes the cost of spaghetti ingredients and an additional worker, without specifying exactly how these should be done.

Modification accuracy was measured based on scores for each of the abstract subtasks. Testing effectiveness was measured by whether or not tests were executed. An alternative measure of testing effectiveness could be du-adequacy. However, du-adequacy was not used because the subjects did the modification task in different ways, with some subjects creating more new cells or changing more cells than other subjects. Hence the resulting student programs contained different numbers of du-associations.

Our experiment was conducted in a smaller setting, rather than in a larger setting. There were at most 6 subjects during each experimental session. We chose to conduct the experiment in a smaller setting as we thought the subjects would be more attentive in a smaller setting. We also felt that it would provide a more uniform knowledge of Forms/3 and testing since we could observe the subjects more closely during the tutorial and they would be more likely to ask questions in a small group then in a large group.

The tutorial for this experiment was given using overhead transparencies. It would be interesting to see how the results might vary if we explained the WYSIWYT features using a computer during the tutorial and asking the subjects to repeat our steps.

6.2 Future work

This modification experiment is just one part of a much larger investigation of the problems involved in bringing some of the benefits of software engineering to the informal environment of spreadsheet languages. There are questions remaining about WYSIWYT methodology:

1. Will the end users derive the same benefits from the methodology as the participants of this study?

2. How would the results change if the subjects were just able to look at the values and not the formulas or vice-versa.
3. Would it have mattered had the experimental problem been from a totally different domain than what the subjects are familiar with?

Some of the future experiments that we're planning to conduct may reveal the answers to these questions.

6.3 Conclusions

One improvement for the future experiments would be automating the process of setting up the experimental environment and starting and stopping the transcripts.

The software engineering properties of spreadsheet languages have rarely been studied; this is a serious omission because these languages are being used to create production software upon which real decisions are based. Further, research shows that many of the spreadsheets created with these languages contain faults. For these reasons, it is important to provide support for mechanisms, such as maintenance and testing, that can help spreadsheet programmers determine the reliability of the values produced by their spreadsheets.

In this thesis we reported empirical results about a methodology aimed at improving the maintainability of spreadsheet programs. The major results were:

- Subjects using the WYSIWYT methodology were more accurate in a modification task.

- Subjects using the WYSIWYT methodology were significantly more effective in testing. The methodology encouraged subjects to test their programs after making changes. Most of the subjects without the methodology did no testing.

- Subjects using the WYSIWYT methodology were better able to predict the accuracy of their modification and were less overconfident about how well they had tested their programs.

Further, it was possible for the subjects using the WYSIWYT methodology to achieve these benefits even without training in the theory of the underlying test adequacy criterion and its relationship to the visual devices. This is encouraging because it suggests that it is possible for end-users with little or no programming experience to achieve some of the benefits of the formal theory without training in the testing principles behind our methodology.

This is the first empirical study of spreadsheet language program maintenance. Although no accurate figures are available, given the extensive use of commercial spreadsheets by the business and other communities, it is safe to say that spreadsheet maintenance is a common and important task. Therefore it is surprising that software engineering research has to a large extent ignored this area. That many of the spreadsheet programs are developed by end-users who have little or no programming experience and who cannot be expected to become fluent in software engineering methodology makes this an especially challenging area of research. The tools and techniques for spreadsheet programmers must be accessible to someone without a software engineering background. Our study has demonstrated one promising approach that could serve as a stepping stone for future research.

BIBLIOGRAPHY

[1] C. M. Allwood, "Errors Detection Processes in Statistical Problem Solving", *Cognitive Science, 8(4)*, pages 413-437, 1984.

[2] A. Ambler, M. Burnett, and B. Zimmerman, "Operational versus definitional: A perspective on programming paradigms", *Computer*, 25(9): 28-43, Sept. 1992.

[3] S. Bagnara, F. Stablum, A. Rizzo, A. Fontana, and M. Ruo, "Error Detection and Correction: A Study on Human-Computer Interaction in a Hot Strip Mill Planning and Production System", *Preprints of the First European Meeting on Cognitive Engineering Approaches to Process Control*, Marcoussis, France, 1987.

[4] F. Belli, O. Jack, "A test coverage notion for logic programming", In *Proc. Of the 6th Intl. Symp. on Softw. Rel. Eng.*, pages 133-142, 1995.

[5] P. Brown and J. Gould, "Experimental study of people creating spreadsheets", *ACM Trans. Office Info. Sys.*, 5(3): 258-272, July 1987.

[6] M. Burnett and H. Gottfried, "Graphical definitions: Expanding spreadsheet languages through direct manipulation and gestures", *ACM Trans. Computer Human Interaction*, pages 1-33, Mar. 1998.

[7] Business Week "How Personal Computers Can Trip Up Executives," (2861) September 24, 1984, pages 94-102 passim.

[8] E. H. Chi, P. Barry, J. Riedl, and J. Konstan, "A spreadsheet approach to information visualization", *In IEEE Symp. Info. Visualization*, Oct. 1997.

[9] J. S. Davis, "Tools for spreadsheet auditing", *Intl. J. of Human-Computer Studies*, 45: 429-442, 1996.

[10] S. Ditlea, "Spreadsheets Can be Hazardous to Your Health," *Personal Computing* (11:1) January 1987, pages 60-69.

[11] E. Duesterwald, R. Gupta, and M. L. Soffa, "Rigorous data flow testing through output influences", *In Proc. 2nd Irvine Softw. Symp.*, Mar. 1992.

[12] P. Frankl and S. Weiss, "An experimental comparison of the effectiveness of branch testing and data flow testing", *IEEE Trans. Softw. Eng.*, 19(8): 774-787, Aug. 1993.

[13] P. Frankl and E. Weyuker, "An applicable family of data flow criteria", *IEEE Trans. Softw. Eng.*, 14(10): 1483-1498, Oct. 1988.

[14] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow and control flow-based test adequacy criteria", In *16th Intl. Conf. Softw. Eng.*, pages 191-200, May 1994.

[15] A. Kay, "Computer Software", *Scientific American*, Sept. 1984, 53-59.

[16] V. Krishna, C. Cook, D. Keller, J. Cantrell, C. Wallace, M. Burnett, G. Rothermel, "Incorporating Incremental Validation and Impact Analysis into Spreadsheet Maintenance: An Empirical Study", (submitted to) *IEEE Intl. Conf. on Softw. Maintenance,* 2001.

[17] W. Kuhn and A. U. Frank, "The use of functional programming in the specification and testing processes", In *Intl. Conf. and Wkshp. Interoperating Geographic Info. Systems*, Dec. 1997.

[18] J. Leopold and A. Ambler, "Keyboardless visual programming using voice, handwriting, and gesture", *IE Syn 1997 IEEmp. Vis. Lang.*, pages 28-35, Sept. 1997.

[19] G. Luo, G. Bochman, B. Sarikiya, and M. Boyer, "Control-Flow based testing of Prolog Programs", In *Proc. of the 3rd Intl. Symp. Softw. Rel. Eng.*, Pages 104-113, 1992.

[20] M. Marre and A. Bertolino, "Reducing and estimating the cost of test coverage criteria", In *1996 IEEE 18th Intl. Conf. Softw. Eng.*, pages 486-494, Mar. 1996.

[21] B. Myers, "Graphical techniques in a spreadsheet for specifying user interfaces", In *ACM CHI '91*, pages 243-249, Apr. 1991.

[22] F. Ouabdesselam and I. Parissis, "Testing techniques for data-flow synchronous programs", In *AADEBUG'95: 2nd Intl. Wkshp. Automated and Algorithmic Debugging*, May 1995.

[23] R. Panko and R. Halverson Jr., "Are Two Heads Better Than One? (At Reducing Spreadsheet Errors in Spreadsheet Modeling?)", In *the Association of American Systems Americas Conference*, Aug. 1996.

[24] R. Panko, "Finding spreadsheet errors: Most spreadsheet models have design flaws that may lead to long-term miscalculations", *Information Week*, pg. 100, May 29, 1995.

[25] R. Panko, "What we know about spreadsheet errors", *J. End User Comp., pages 15-21*, Spring 1998.

[26] R. Panko and R. Halverson Jr., "Spreadsheets on trial: A framework for research on spreadsheet risks", *29th Hawaii Intl. Conf. on System Sciences*, Vol. II, pages 326-335, Jan. 1996.

[27] R. Panko, Spreadsheet Research (SSR), http://www.cba.hawaii.edu/panko/ssr.

[28] K. Rajalingham, D. Chadwick, B. Knight and D. Edwards, "An integrated spreadsheet engineering methodology", *IFIP TC11 WG11.5 Third Working Conf. on Integrity and Internal Control in Info. Systems*, pages 41-58, Nov. 1999.

[29] S. Rapps and E. J. Weyuker, "Selecting software test data using data flow information", *IEEE Trans. Softw. Eng., 11(4): 367-375*, Apr. 1985.

[30] J. Reichwein, G. Rothermel, and M. Burnett, "Slicing spreadsheets: An integrated methodology for spreadsheet testing and debugging", *2nd Conf. on Domain Specific Languages*, pages 25-38, Oct. 1999

[31] B. Ronen, R. Palley, and H. Lucas, "Spreadsheet Analysis and Design", In *Communications of the ACM*, 21(1): 84-93, 1989.

[32] K. Rothermel, C. Cook, M. Burnett, J. Schonfeld, T.R.G. Green, and G. Rothermel, "WYSIWYT testing in the spreadsheet paradigm: An empirical evaluation", *22nd Intl. Conf. on Softw. Eng.*, Limerick, pages 230-239, June 2000.

[33] G. Rothermel, M. Burnett, L. Li, C. DuPuis, A. Sheretov, " A methodology for testing spreadsheets", A*CM Trans. on Softw. Eng. and Methodology,* (to appear).

[34] G. Rothermel, L. Li, C. DuPuis, and M. Burnett, "What you see is what you test: A methodology for testing form-based visual programs", In *The 20th Intl. Conf. Softw. Eng.*, pages 198-207, Apr. 1998.

[35] G. Rothermel, L. Li, and M. Burnett, "Testing strategies for form-based visual programs", In *Proc. of the 8th Intl. Symp. Softw. Rel. Eng.*, pages 96-107, Nov. 1997.

[36] T. Smedley, P. Cox, and S. Byrne, "Expanding the utility of spreadsheets through the integration of visual programming and user interface objects", In *Adv. Vis. Int. '96*, May 1996.

[37] T. Teo and M. Tan, "Quantitative and Qualitative errors in Spreadsheet Development", In *Proc. of the 30th Hawaii Int'l Conf. on Sys. Sci.*, volume 3, pages 149-155, Jan. 1997.

[38] G. Viehstaedt and A. Ambler, "Visual representation and manipulation of matrices", *Journal Vis. Lang. and Comp., 3(3):273-298*, Sept. 1992.

[39] E. J. Weyuker, "More experience with dataflow testing", *IEEE Trans. Softw. Eng., 19(9): 912-919*, Sept. 1993.

[40] E. Wilcox, J. Atwood, M. Burnett, J. Cadiz, and C. Cook, "Does continuous visual feedback aid debugging in direct-manipulation programming systems", In *ACM CHI'97,* pages 258-265, Mar. 1997.

[41] G. G. Woodbury, "Re: 'Computer Error' in Durham N.C. Election Results," *The Risks Digest* (9:42) http://catless.ncl.ac.uk/Risks, November 13, 1989.

[42] D. D. Woods, "Some Results on Operator Performance in Emergency Events", *Institute of Chemical Engineers Symposium Series*, 90, pages 21-31, 1984.

APPENDICES

Appendix A

Code _____

All Subjects Background Questions

1.Major _____

2.Overall GPA _____

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

High school course(s). Number?_____

College course(s). Number?_____

Professional. How long?_____

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

A high school course

A college course

Professional use

Personal use

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

6.Is English your primary language? Yes / No

If not, how long have you been speaking English? _____

Appendix B

Forms/3 Quick Reference Card

Ad Hoc

Edit a cell's formula	1.Click a cell's formula tab to show the formula. 2.Make changes. 3.Click the "Apply" button at the top of the formula window.
Formula display	Click the formula tab.
Formula hide	Click the "X" at the top left corner of the formula window.
Record a testing decision	1.Select the cell. 2.Click its checkbox.
Add a Cell	1.Select the New Cell tool from the tool palette. 2.Click the right mouse button anywhere on the form to place a new cell.
Delete a Cell	1.Select the cell to be deleted. 2.Click the button with the red "X" located at the top of the form.

Appendix C

Forms/3 Quick Reference Card

WYSIWYT

Basic Cell Operations

Add a Cell	1.Select the New Cell tool from the tool palette. 2.Click the left mouse button anywhere on the form to place a new cell.
Delete a Cell	1.Select the cell to be deleted. 2.Click the button with the red "X" at the top of the form window.
Edit a cell's formula	1.Click a cell's formula tab to show the formula. 2.Make changes. 3.Click the "Apply" button at the top of the formula window.
Formula display	Click the formula tab.
Formula hide	Click the "X" at the top left corner of the formula window.
Arrows On / Off for a cell	Right click on the cell whose arrows you wish to see.
Arrows with detail for a cell	Turn Arrows on for the cell and display formulas at sources and destinations of arrows.
Checkbox	Cell's upper-right corner which records a testing decision.

Testedness Information

Record a testing decision and update testedness colors.	Click on the cell's checkbox when its advice box contains an exclamation mark.
Remove most recent testing decision for a cell.	If the check mark is still on screen, click on the check mark to remove it, otherwise it can't be removed.
Red Cell Border/Arrow	Cell/Relationship not tested.
Blue Cell Border/Arrow	Cell/Relationship fully tested.
Purple Cell Border/Arrow	Cell/Relationship only partially tested.
Question mark	You have made a decision for this cell's formula for some previous value.
Check mark	You have made a decision for this cell's formula for this value.
Blank	You have made no decision about this cell's value or formula.
Exclamation Point	By checking this cell's checkbox, you will increase the testedness of the spreadsheet.

Appendix D

Description of the Financial Spreadsheet Model

The cashflow projection model for a Pizza Parlor Restaurant consists of information about sales, expenses and final cashflow for each particular month. The projections for January, February and March are listed below:

Sales	**Jan**	**Feb**	**Mar**
Total Pizza Sales	20000	24000	28000
Expenses			
Cost of Ingredients	12400	14880	17360
Salaries	8000	8000	8000
Total Expenses	20400	22880	25360
Pretax Cashflow	-400	1120	2640
Taxes	0	280	660
After Tax Cashflow	-400	840	1980

The spreadsheet that you have been given shows the above information is based on the following assumptions:

Assumptions

- There is a monthly increase of $4000 in pizza sales.
- The cost of ingredients for pizza is 62% of the total pizza sale for that month.
- The restaurant has a manager and 4 workers. The manager's salary is $4000/month and every worker is paid $1000/month.
- Total expenses include the cost of ingredients and the salary.
- Pretax cashflow is the difference between the total sales and expenses.
- Tax is calculated based on the pretax cashflow. If the cashflow is positive there is a tax of 25% on the pretax cashflow, otherwise there is no tax.
- After tax cashflow is the balance after deducting the tax from the pretax cash flow.

Appendix E

Comprehension Questions

Answer the following questions for the cash flow projection model:

1. What is the name of the cell containing the total pizza sales for the month of March?

2. What is the total expense amount for the month of February?

3. What is the name of the cell containing the after-tax cash flow for the month of January?

4. What is the tax amount for the month of March?

Appendix F

Description of the Modification Task

The cash flow projection model is being used in all of the Pizza Parlor Restaurants throughout the U.S. It has been thoroughly tested. Your task is to modify the model to include the changes below. Since the modified spreadsheet is to be used at all Pizza Parlor Restaurants, it should be working correctly.

1. Recent action by the Congress changed the tax rate from 25% to 28% starting in January.
2. The restaurant management decided to include Spaghetti in their menu, starting in the month of February. The projection is $5000 Spaghetti sales for month of February and $6000 for the month of March. The cost of ingredients for spaghetti is 53% of the total spaghetti sale for that month.
3. The management also decided to hire one more worker to the introduction of spaghetti.

Appendix G

Code _____

Post Session Questionnaire

Ad Hoc

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 completed the tax rate modification? 1 2 3 4 5

How confident are you that the tax rate modification is correct? 1 2 3 4 5

How confident are you that you completed the spaghetti modification? 1 2 3 4 5

How confident are you that the spaghetti modification is correct? 1 2 3 4 5

How confident are you that there are no errors in the spreadsheet? 1 2 3 4 5

2. How much additional time would you need to complete the modifications?

   _____ None. It only took me _____ minutes.

   _____ None. I took about the whole time.

   _____ I would need about _____ more minutes.

   _____ I'm not sure.

3. How well do you feel that you tested the spreadsheet?

_____ Really, really well. (If you graded it, I'd get an A)

_____ Better than average. (If you graded it, I'd get a B)

_____ About average. (If you graded it, I'd get a C)

_____ Worse than average. (If you graded it, I'd get a D)

_____ Poorly (If you graded it, I'd get an F)

4. How much additional time would you need to complete testing?

_____ None. It only took me _____ minutes.

_____ None. I took about the whole time.

_____ I would need about _____ more minutes.

_____ I'm not sure.

Appendix H

Code _____

Post Session Questionnaire

WYSIWYT

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 completed the tax rate modification? 1 2 3 4 5

How confident are you that the tax rate modification is correct? 1 2 3 4 5

How confident are you that you completed the spaghetti modification? 1 2 3 4 5

How confident are you that the spaghetti modification is correct? 1 2 3 4 5

How confident are you that you completed the new worker modification?

1 2 3 4 5

How confident are you that the new worker modification is correct? 1 2 3 4 5

2. How much additional time would you need to complete the modifications?

   _____ None. It only took me _____ minutes.

   _____ None. I took about the whole time.

   _____ I would need about _____ more minutes.

   _____ I'm not sure.

3. How well do you feel that you tested the spreadsheet?

   _____ Really, really well. (If you graded it, I'd get an A)

   _____ Better than average (If you graded it, I'd get a B)

_____ About average. (If you graded it, I'd get a C)

_____ Worse than average. (If you graded it, I'd get a D)

_____ Poorly (If you graded it, I'd get an F)

_____ I'm not sure (If you graded it, I'd get an F)

4. How much additional time would you need to complete testing?

_____ None. It only took me _____ minutes.

_____ None. I took about the whole time.

_____ I would need about _____ more minutes.

_____ I'm not sure.

5. Use this scale to answer questions regarding the testing tools:

1 = Not Helpful

2 = Somewhat Helpful

3 = Helpful

4 = Quite Helpful

5 = Very Helpful

Seeing the arrows was:

Seeing the cell border colors was: 1 2 3 4 5

Seeing a blank in a checkbox was: 1 2 3 4 5

Seeing a questionmark in a checkbox was: 1 2 3 4 5

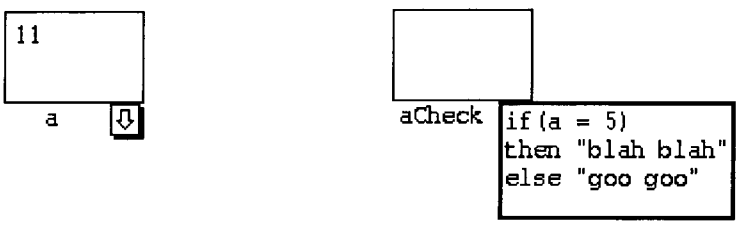
Seeing a checkmark in a checkbox was: 1 2 3 4 5

Seeing an exclamation point next to a checkbox was: 1 2 3 4 5

Seeing the percent tested indicator was: 1 2 3 4 5

For the remaining questions, please check the last choice if you have to guess at the answer.

6. Given the cells pictured below, what is the correct output value for the cell aCheck?



_____ blah blah

_____ goo goo

_____ 11

_____ None of the above

_____ I'm not sure

7. Suppose the arrow in the picture below is blue. What does a blue arrow between cells indicate?

_____ the cells at both ends of the arrow are both fully tested

_____ the relationship between the two cells is fully tested

_____ the cell the arrow comes from is fully tested

_____ None of the above

_____ I'm not sure

8. If there is an exclamation point next to a cell's checkbox and you click that checkbox, what color does the cell's border become?

_____ red

_____ purple

_____ blue

_____ purple or blue

_____ I'm not sure

9. Suppose a cell has a question mark in its checkbox. What is the color of the border of the cell?

_____ Red

_____ Purple

_____ Blue

_____ Red or Purple or Blue

_____ I'm not sure