

WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation

Karen J. Rothermel[†], Curtis R. Cook[†], Margaret M. Burnett[†],
Justin Schonfeld[†], T. R. G. Green[‡], Gregg Rothermel[†]

[†]Department of Computer Science
Oregon State University
Corvallis, OR 97331

[‡]Computer-Based Learning Unit
University of Leeds
Leeds LS2 9JT, U.K.

[†]{rotherka,cook,burnett,schonfju,grother}@cs.orst.edu

[‡]thomas.green@ndirect.co.uk

ABSTRACT

Is it possible to achieve some of the benefits of formal testing within the informal programming conventions of the spreadsheet paradigm? We have been working on an approach that attempts to do so via the development of a testing methodology for this paradigm. Our “What You See Is What You Test” (WYSIWYT) methodology supplements the convention by which spreadsheets provide automatic immediate visual feedback about values by providing automatic immediate visual feedback about “testedness”. In previous work we described this methodology; in this paper, we present empirical data about the methodology’s effectiveness. Our results show that the use of the methodology was associated with significant improvement in testing effectiveness and efficiency, *even with no training* on the theory of testing or test adequacy that the model implements. These results may be due at least in part to the fact that use of the methodology was associated with a significant reduction in overconfidence.

Keywords

spreadsheets, testing, visual programming, empirical studies

1 INTRODUCTION

Perhaps the most widely used programming paradigm today is the spreadsheet paradigm. Yet, almost no work has been done to help with the software engineering tasks that arise in the creation and maintenance of spreadsheets. This inattention is rather surprising given

the influential role played by spreadsheets in decisions about a variety of real-world issues, such as those involving budgets, student grades, and tax calculations.

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 [1]. 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 those of affected cells (at least those that are visible), and immediately displays new results. Spreadsheet languages are used for computational tasks ranging from simple “scratchpad” applications developed by single users to large-scale, complex systems developed by multiple users [18].

Despite the perceived simplicity of the spreadsheet paradigm, research shows that spreadsheets may often contain faults. A survey of the literature provides details: in 4 field audits of operational spreadsheets, errors were found in an average of 20.6% of spreadsheets audited; in 11 experiments in which participants created spreadsheets, errors were found in an average of 60.8% of the spreadsheets; in 4 experiments in which the participants inspected spreadsheets for errors, the participants missed an average of 55.8% of the errors [18].

Contributing to the problem of reliability is the unwarranted confidence that spreadsheet programmers seem to have in the correctness of their spreadsheets [3, 28]. A possible cause of this overconfidence could be related to the findings of Gilmore and of Svendsen, who showed that too much feedback and responsiveness, as featured in the immediate visual feedback of *values* in spreadsheet languages, can actually interfere with peo-

ple’s problem-solving ability in solving puzzles [10, 25], a task with much in common with programming.

To address the reliability problem associated with spreadsheets, we are investigating the possibility of bringing some of the benefits of formal testing to the informal, highly interactive, declarative spreadsheet paradigm. Our theory is that by providing feedback about the “testedness” of a spreadsheet, as opposed to just the values, we can cause programmers to have less overconfidence about the correctness of their spreadsheets. This testing feedback may motivate programmers to test their spreadsheets more thoroughly, and provide guidance that helps them test more efficiently. This could lead to better testing even in this informal programming domain, and to a reduction in faults before a spreadsheet is relied upon in decision making.

We have therefore developed a “What You See Is What You Test” (WYSIWYT) methodology for testing spreadsheets [23]. Our methodology is designed to accommodate the declarative evaluation model of spreadsheet formulas, the incremental style of development, and the immediate visual feedback expected of spreadsheet languages. The methodology is designed for use by the wide range of programmers using spreadsheet languages, especially taking into account the lack of formal background of many of these programmers.

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

First, is the methodology efficient enough to coexist with the immediate cell redisplay expected after each formula edit? In our previous work, we showed that most of our algorithms can be implemented in ways that add only $O(1)$ to the existing cost of maintaining the interactive environment.

Second, will the methodology uncover faults in programs? Our methodology guides programmers (though they need not be aware of it) in meeting a dataflow test adequacy criterion. (This criterion will be described in the next section.) We used our implementation to empirically study the fault detection characteristics of test suites that meet this criterion. Our 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 [23, 20].

Third, will programmers who use the methodology be less overconfident and be more effective, more efficient testers than programmers who do not use the methodology? To investigate this third question, we have begun a series of empirical studies with human subjects. The first of these studies is now complete, and is the subject of this paper.

2 BACKGROUND

The literature on testing primarily addresses the testing of imperative programs (e.g. [7, 9, 11, 15, 19, 27]), with a few attempts to address the testing of functional and logic programs (e.g. [2, 12, 14, 17]). However, differences between the spreadsheet and imperative language paradigms directly impact the development of a 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. This dependence-driven evaluation model allows evaluation engines for spreadsheets flexibility in the scheduling algorithms and optimization devices they employ to perform computations. A methodology for testing spreadsheets must be compatible with this flexibility, 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 formula. A testing methodology for spreadsheets must be flexible enough to operate upon partially-completed programs and efficient enough to support responsiveness. Third, and most critical, whereas most imperative programs are developed by professional programmers, spreadsheets are developed by a wide variety of users, many of whom have no training in formal testing principles. Our methodology for testing spreadsheets takes these factors into account. We overview the foundations of that methodology here; a detailed presentation can be found in [22, 23].

Spreadsheet programmers are not likely to write specifications for their spreadsheets; thus, our 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. [9, 15, 19]), and several empirical studies (e.g. [8, 11, 27]) have demonstrated their usefulness.

Our methodology incorporates a test adequacy criterion adapted from the *output-influencing-All-du* dataflow adequacy criterion defined originally for imperative programs [7]. 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-associations; 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 [9, 27]; thus, data flow test adequacy criteria typically require that test data exercise (cover) only executable du-associations. In this respect, our criterion is no exception. 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 faults in spreadsheets this is valuable. Moreover, the criterion does not enforce any expectation of a particular cell execution order: the du-associations in a spreadsheet are the same regardless of the order in which the evaluation engine executes cells. Further, 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. Finally, the criterion facilitates the incremental testing 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.

3 EXPERIMENT DESIGN

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

RQ 1: Do programmers who use our testing methodology create test suites that are more effective in terms of du-adequacy than programmers who use an ad hoc approach?

RQ 2: Do programmers who use our testing methodology create test suites more efficiently than programmers who use an ad hoc approach?

RQ 3: Are programmers who use our testing methodology less overconfident about the quality of their testing than programmers 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 spreadsheet programmers to gain testing effectiveness or efficiency from using our methodology?

Student Grades						
	NAME	ID	HWAVG	MIDTERM	FINAL	COURSE
1	Abbott, Mike	1035	89	91	86	89
2	Farnes, Joan	7649	92	94	92	93
3	Green, Matt	2314	78	80	75	78
4	Smith, Scott	2316	84	90	86	87
5	Thomas, Sue	9857	91	87	90	90
AVERAGE			87	88	86	87

Figure 1: Spreadsheet for calculating student grades.

To investigate these questions, we conducted a controlled laboratory experiment. In the experiment, the subjects tested spreadsheets. Half of the subjects did so using a spreadsheet environment that includes the WYSIWYT methodology, and the other half used the same environment minus the testing methodology.

We collected the actions of each subject throughout the testing sessions, automatically recorded in transcript files; the test suites created by the subjects' actions (derived from the information in the transcript files); subject background questionnaires; and post-experiment questionnaires in which all subjects rated how well they thought they tested the spreadsheet and WYSIWYT subjects answered additional questions about their use and understanding of our methodology's feedback.

The Experimental Environment

We have prototyped our WYSIWYT testing methodology in the research language Forms/3 [4], one of many spreadsheet language research systems (e.g. [4, 5, 13, 16, 24, 26]). 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, Forms/3 spreadsheets are a collection of cells; each cell's value is defined by the cell's formula. The programmer enters a cell's formula and receives immediate visual feedback as to the cell's value. In Forms/3, ordinary formulas can be used for both numeric and graphical computations. Figure 1 shows a gradebook spreadsheet that computes averages for each student and for the class. Figure 2 shows a spreadsheet that translates numeric input values for hour and minute into a graphical clock face with hour and minute hands. As this figure shows, the programmer can freely reposition a cell and can choose to display or hide the cell's formula. In the Clock spread-

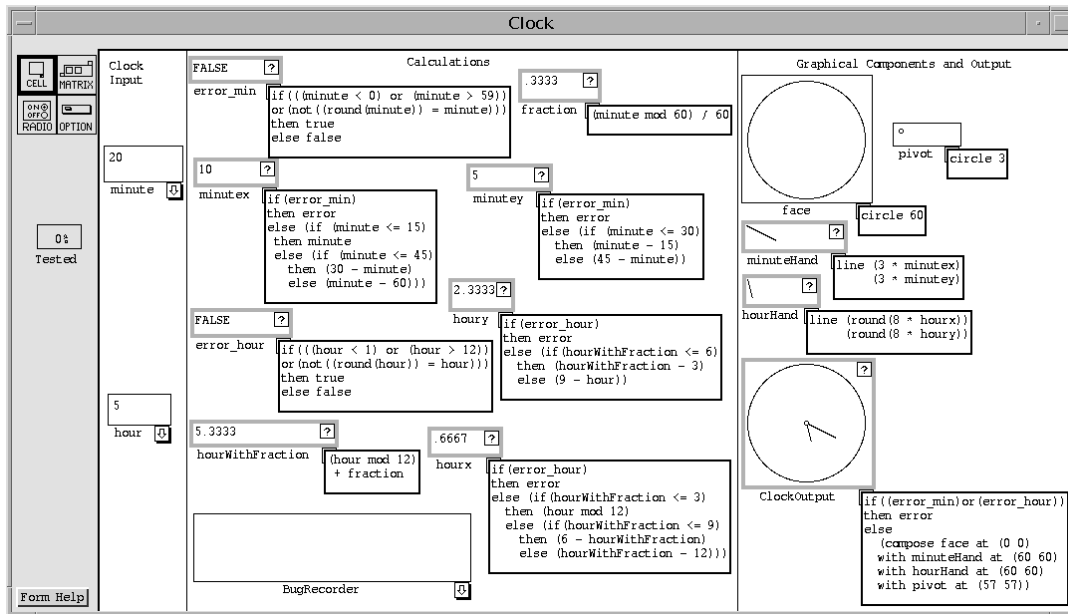


Figure 2: Spreadsheet for displaying a clock face.

sheet, the programmer has chosen to display all of the non-constant formulas. Eventually, when the programmer has finished testing the Clock spreadsheet, the input cells (those whose formulas contain only constants) could be replaced with references to the system clock.

Suppose that a programmer has begun creating the graphical clock shown in Figure 2, and has entered a few of the cells and formulas. 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 devices keep the programmer continually informed as to testedness status, to draw attention to the untested sections of the evolving spreadsheet, and to suggest where testing activity will provide new progress (according to our adequacy criterion) over previous testing activities.

For example, suppose the spreadsheet programmer now decides that cell MinuteHand’s displayed value is correct, given the other cells’ current values, and clicks on the check box in the upper right corner of cell MinuteHand to validate it. The system responds with immediate visual feedback as to the new testedness of each visible cell¹ and arrow, as well as for the spreadsheet, as shown in Figure 3. The underlying validation algorithm is given in [23]; the overall notion is that it recurses back through the du-associations that affect, directly or in-

¹Formally, du-adequacy does not define test adequacy at the granularity of cells. When we refer to a cell being tested in this paper, this is a shortcut for saying that all the du-associations whose uses are in the cell have contributed to a value that the programmer pronounced correct.

directly, the currently computed value of MinuteHand, and marks them tested (covered). 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 shading from red through purple to blue (darker gray). If the programmer chooses to display arrows among some of the cells, the arrows follow the same color scheme. We provide additional testing information through the marks in the cell’s check box: a question mark indicates that validating this cell’s value will increase the spreadsheet’s testedness, a blank indicates that validating the cell’s value will not increase testedness, and a check mark indicates that the user’s validation was recorded.

The methodology also accounts for the retesting that may be required whenever the spreadsheet programmer edits a non-input formula, but this aspect was not included in the experiment, so we do not discuss it here.

Half of the subjects used the environment described above. The other half used an identical environment minus the colors, arrows, question marks, check marks, and “% Tested” indicator. A check box was still present (to allow the subjects to communicate their testing decisions), and the system gave a quick flash of feedback to indicate that the decision was recorded.

The Subjects

We have previously pointed out that the spreadsheet paradigm serves a range of audiences, from end users to professional programmers. In this experiment we have chosen to focus on the professional programmers side of this range, choosing advanced Computer Science students as our subjects. Programmers are one segment

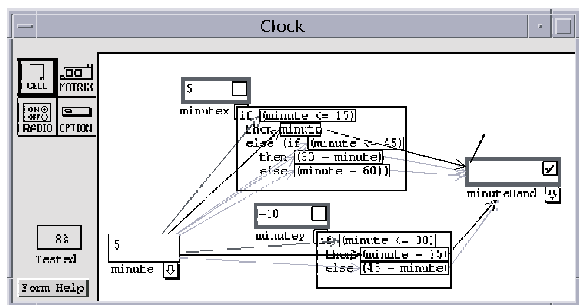


Figure 3: “Under construction” Clock after a validation.

of the population served by spreadsheets, and are an especially interesting group of subjects for exploration of our research questions because of their previous experience with testing. That is, programmers and CS students are much more experienced in testing than are end users; hence, because of their previous experience and practice with testing, they might already be efficient and effective at testing without much room for improvement due to our methodology.

78 subjects participated in the experiment, drawn from three Computer Science courses: two upper-division undergraduate courses and one graduate course in Computer Science. The subjects had very little or no previous exposure to the experimental environment. The subjects were randomly divided into two groups, subject to balancing the number of undergraduate and graduate students between the groups. The control (Ad Hoc) group did not have access to our WYSIWYT methodology and represents programmers who test in an ad hoc fashion. The treatment (WYSIWYT) group did have access to our methodology.

Of the 78 subjects, the Ad Hoc group and WYSIWYT groups contained 37 and 41 subjects, respectively. The difference in group size was due to a few subjects failing to arrive for their appointments. The group sizes were subsequently reduced when we decided to omit data for subjects whose measurable activity level was zero (as revealed by transcripts of the sessions), whose computer crashed during the experiment, or who inadvertently corrupted their data in other ways. The removal of these subjects reduced the number of subjects in the Ad Hoc and WYSIWYT groups to 30 and 39, respectively.

To ascertain whether the subjects in the two groups had reasonably similar backgrounds, we administered a background questionnaire to each subject and analyzed the data. Results are summarized in Table 1. Our analysis showed homogeneity between the groups except for the GPA category, where we found that the Ad Hoc subjects had significantly higher GPAs than the WYSIWYT subjects. Since the GPAs were self-reported by the subjects and approximately one-third of the subjects did not report a GPA, this non-homogeneity of GPAs is

tenuous. More to the point, however, is that if anything, the difference in GPAs would work against the hypothesis that the methodology would improve subjects’ testing ability; hence the presence of this non-homogeneity would only further strengthen our results.

The Tutorial

The experiment was conducted in a lab with subjects seated one per workstation, using Forms/3. The experiment began with a 20-minute tutorial of Forms/3, in which each subject worked with the example spreadsheets on their workstations following instructions given by the lecturer. Throughout the tutorial the participants had access to a handout containing a quick reference of the spreadsheet features they were being taught. They could make notes on these handouts which remained available to them throughout the experiment.

The first part of the tutorial introduced the subjects to language features (e.g., basic syntax of formulas) and environmental features (e.g., how to edit cells) that would eventually be needed in testing the spreadsheets. The second part of the tutorial described how to record testing decisions. Testing was described as the process of trying various input values and recording decisions made about the correctness of values in the other cells. All subjects were instructed to use the check boxes to record decisions about correct cell values and to record information about incorrect cell values in a special cell named BugRecorder. All subjects received the same information in the first and second parts of the tutorial.

The third part of the tutorial explained how to interpret the testing feedback and here the instructions for the Ad Hoc subjects diverged from those for the WYSIWYT subjects. For the Ad Hoc subjects, the lecturer described the feedback (a quick flash) from validating a cell’s value as indicating that the system had recorded the decision. After several examples of trying inputs and recording decisions, the Ad Hoc subjects were given unstructured time to practice their Forms/3 skills.

In designing this third tutorial part for the WYSIWYT group, we were faced with an important decision: whether or not to provide a technical explanation of the underlying concepts of such things as du-associations. Because one of our goals is that our testing methodology not require an understanding of the underlying theory, we chose to explain only that red means “not tested”, blue means “tested”, and purple means “partially tested”. The question marks were described as meaning “recording a decision here will help test part of the spreadsheet that has not been tested”, check marks as “you have recorded a decision here”, blanks as “you have previously recorded a decision that covers this case”. We did not mention the underlying concepts of du-associations, nor did we describe nonexecutable

	No. of Subjects	Overall GPA	CS GPA	programming languages known	grad students per group	subjects with spreadsheet experience	subjects with professional experience
Ad Hoc	30	3.45	3.7	4	10	12	11
WYSIWYT	39	3.2	3.5	4	8	10	20

Table 1: Subject group demographics (medians).

du-associations. At the end of these explanations, the WYSIWYT subjects were given unstructured time to practice their Forms/3 skills.

Regardless of which group a subject was in, the total time of training received was identical; subjects not receiving explanations of our 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 given the WYSIWYT subjects in the third part also provided them with additional practice in Forms/3 skills, and additional practice time could have confounded the results. Since the subjects had very little or no previous exposure to the experiment’s environment, at the conclusion of the tutorial the subjects could be considered equal in their knowledge of Forms/3.

Task and Materials

Following the tutorial, the subjects were asked to test two spreadsheets, Clock and Grades. Using two different spreadsheets reduced the chances that the results of our study would be tied to any one spreadsheet’s particular characteristics. The Clock spreadsheet was shown in Figure 2 and discussed earlier in this section. The Grades spreadsheet (Figure 4) is a single-student variant of the spreadsheet presented in Figure 1; it calculates the final letter grade for a student.

The differences between Clock and Grades are that (1) they represent different problem domains (numeric and graphics), (2) the formulas for Grades are relatively easy to understand (lending themselves to straightforward reasoning by examining the code) whereas those of Clock are relatively difficult to understand, and (3) the oracle problem—determining whether the displayed answer is correct—maybe more difficult for Grades than for Clock. Both spreadsheets were tested by all subjects.

The numeric domain is of interest because of the traditional uses of spreadsheets. The graphical domain is of interest because of the increasing use of spreadsheets to produce graphics (e.g., [5, 24]). The two particular spreadsheets were deliberately designed to be different from each other in this respect and in the other respects enumerated above. They were designed to be similar in the familiarity of their problem domains (grades and clocks) to the subjects and to be of reasonable size and complexity given a limited amount of time.

The subjects were given 15 minutes per spreadsheet. The experiment was counterbalanced with respect to

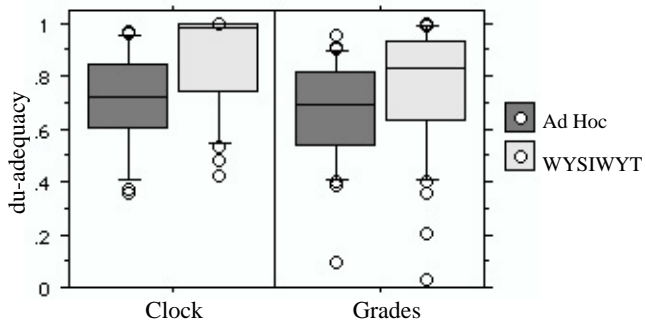


Figure 5: Effectiveness data for Clock and Grades. The boxplots are composed of 5 horizontal lines at the 10th, 25th, 50th, 75th and 90th percentiles; values above the 90th or below the 10th percentiles are plotted separately.

the problem type: subjects in each group worked both problems, but half the subjects in each group tested Grades first, whereas the other half tested Clock first. At the beginning of each testing session we instructed the subjects to read a description of the spreadsheet they were about to test. The descriptions included details of what the spreadsheet was to accomplish, the range of accepted input and output values and the error messages expected for out-of-range inputs.

4 RESULTS

Effectiveness

Our first research question considers whether using our methodology increased our subjects’ testing effectiveness. We measured effectiveness in terms of du-adequacy: given spreadsheet S that contains D executable du-associations, and test suite T that exercises D_T of those associations, the du-adequacy of T with respect to S is given by D_T/D .

We examined the data (Figure 5) to see whether it satisfied the requirements for using normal-theory analyses such as t-tests and analysis of variance. There were some indications of skew, but the analysis of variance is robust against moderate skew; however, to be on the safe side, non-parametric alternatives were also used where possible, and gave identical patterns of results.²

We analyzed effectiveness using analysis of variance with two factors, Environment (WYSIWYT or Ad Hoc)

²The results from two of the subjects appeared to be outliers, and further investigation showed that one was not following the instructions and the other had limited experience with English. The analyses that follow include results from these two subjects, but the same pattern of results emerged from analyses in which their scores were eliminated.

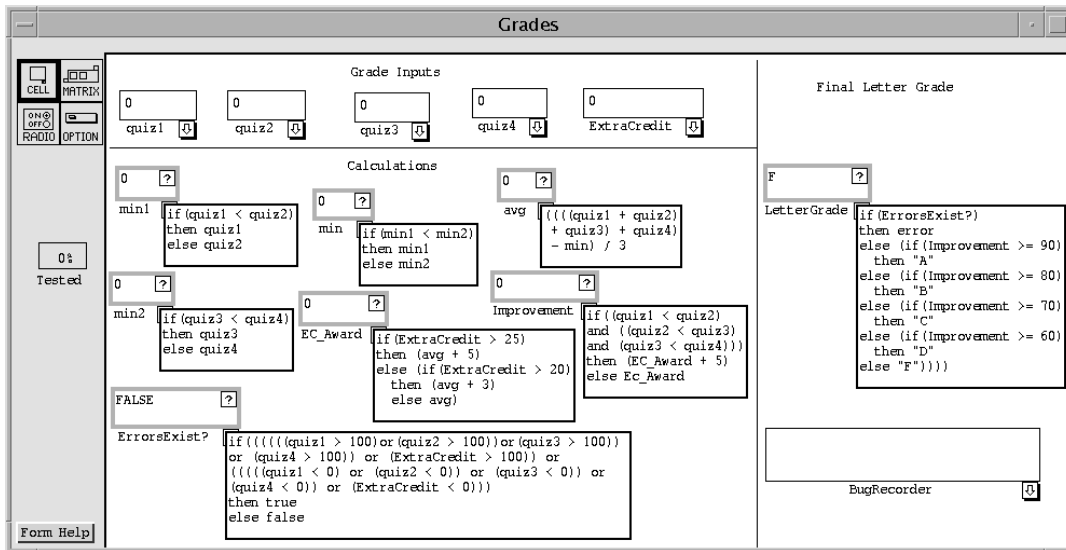


Figure 4: Version of Grades spreadsheet used in experiment.

Clock	# Tests	Redundancy
Ad Hoc	20	61.3%
WYSIWYT	19	15.4%
Grades	# Tests	Redundancy
Ad Hoc	14	44.0%
WYSIWYT	18	4.3%

Table 2: Medians for the redundancy data.

and Problem (Clock or Grades). Each subject experienced only one environment and attempted both problems, so the Environment factor was treated as independent groups and the Problem factor was treated as having repeated measures.

Analysis of variance showed that the effectiveness of the WYSIWYT group was significantly higher than the effectiveness of the Ad Hoc group ($F=8.56$, $df= 1,67$, $p=0.0047$). There was a significant difference between effectiveness on the two problems ($F=9.632$, $df= 1,67$, $p=.0028$) but no significant interaction effect (that is, the WYSIWYT subjects showed the same pattern of greater effectiveness on both Clock and Grades). Independent non-parametric tests (Mann-Whitney) were performed on the two problems considered separately and also on the pooled data, confirming the highly significant differences between environments (Clock, $p=0.0001$; Grades, $p=0.0083$; Pooled Data, $p<0.0001$).

Efficiency

Our second research question considers whether using the methodology increased the subjects' efficiency. One view of efficiency measures "wasted effort" in running redundant tests that do not increase coverage. This measure implies that multiple test cases exercising the same du-association do not increase testing effectiveness. Some evidence exists to suggest that this assump-

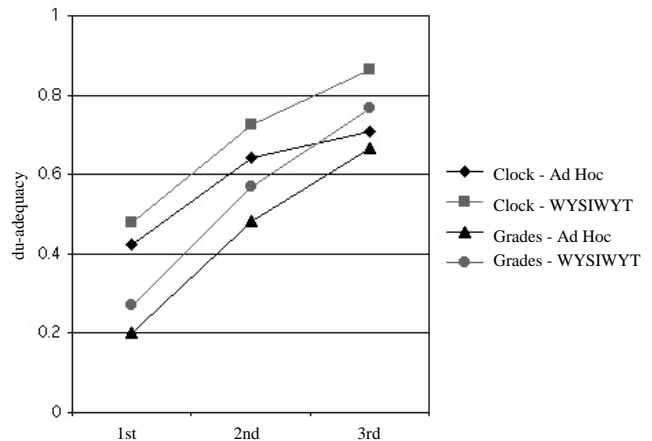


Figure 6: Speed of coverage for Clock and Grades over 3 five-minute time intervals.

tion is overly simplistic [21]; however, this assumption is common to the use of structural coverage criteria.

An analysis of variance on the redundancy data showed that the percentage of redundant test cases created by the WYSIWYT subjects was significantly lower than the percentage of redundant test cases in the test suites created by the Ad Hoc subjects ($F= 47.987$, $df= 1,67$, $p < 0.0001$). There was a significant difference between the percentage of redundant test cases on the two problems ($F= 8.37$, $df=1,67$, $p= 0.0045$) but no significant interaction effect (that is, the WYSIWYT subjects showed the same pattern of lower redundancy on both Clock and Grades). See Table 2. As with the effectiveness data, we performed independent non-parametric tests (Mann-Whitney) on the two problems considered separately and on the pooled data, again confirming the significant differences between environments.

	Clock	
	Overconfident	Not Overconfident
Ad Hoc	16	14
WYSIWYT	10	29
	Grades	
	Overconfident	Not Overconfident
Ad Hoc	20	10
WYSIWYT	14	25

Table 3: Overconfidence: Number of subjects in group, categorized according to overconfidence.

Another view of efficiency is the speed with which coverage was obtained. The data for coverage speed was derived by dividing the 15 minute testing sessions into three 5-minute intervals and determining the subjects’ du-adequacy at the end of each interval. Figure 6 shows the WYSIWYT subjects achieved coverage faster than the Ad Hoc subjects; however, the difference is not significantly larger until the third time period (Mann-Whitney, $p=0.0001$ (Clock), $p=0.0083$ (Grades)).

Overconfidence

The above results show that the WYSIWYT subjects achieved higher coverage more efficiently. However, earlier in this paper, we pointed out that spreadsheet programmers have been shown to have unwarranted confidence in the accuracy of their spreadsheets. The methodology may be of little use in practice if subjects’ overconfidence causes them to stop working on their spreadsheets before making much use of its guidance. For this reason, reducing overconfidence has been an important goal of the WYSIWYT methodology.

At the end of each testing session we asked the subjects to answer the following question:

- How well do you think 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)

We compared each subject’s answer to this question to our “grading” of their du-adequacy. We assigned grades of A - F based on a standard grading scale, coverage of 90-100% \Rightarrow A, 80-89% \Rightarrow B, ... 0-59% \Rightarrow F. If a subject’s self-grade was higher than our grade for the subject, the subject was categorized as “overconfident”, otherwise the subject was categorized as “not overconfident”; see Table 3. We analyzed the overconfidence data using Fisher’s Exact Test. The WYSIWYT group had significantly fewer than expected subjects in the overconfident category while the Ad Hoc group had significantly more than expected subjects in the overconfident category ($p=0.025$ (Clock), $p=0.0155$ (Grades)).

5 DISCUSSION

Given such strong results, a natural question that arises is whether particular portions of our methodology’s

How helpful were:	Very Helpful	Helpful	Not Helpful
question marks	69%	31%	0%
clicking to validate	64%	36%	0%
colored cell borders	56%	44%	0%
colored arrows	51%	41%	8%
check marks	44%	49%	8%
“Tested” indicator	36%	56%	8%
blanks	23%	51%	26%

Table 4: WYSIWYT subjects’ helpfulness ratings: percent of subjects who rated the device in each of the possible helpfulness categories.

A red cell border indicates that the cell is (not tested)	100%
A question mark in a cell’s check box indicates that (the current input tests part of the spreadsheet not previously tested)	87%
A blue arrow between cells indicates (the relationship between the two cells is fully tested)	64%

Table 5: Subjects’ opinions of the meanings of the visual devices. Percentages reflect the number of subjects who chose the correct response (shown in parentheses). The questions were multiple choice and directed the subjects not to guess at the answer.

communication devices are key to the results. For example, would it be possible to attain the same results without the colors to attract attention to untested areas, using only the check marks and question marks to guide the user through the testing activities? Although we do not have a rigorous answer to this question, we do have the subjects’ opinions about which aspects were the most helpful, as reported on their questionnaires. These are given in Table 4, in descending order of the subjects’ votes in the “Very Helpful” category. In the table, the choices listed are abbreviations of the questionnaire wording: to distinguish ideas of outputs from user actions in the questions, we worded the choices about output devices with the words “seeing the...” (e.g., “seeing the colored cell borders”), as opposed to actions the user could take such as “clicking to validate.”

Still, the subjects’ opinions of helpfulness could be misleading if the subjects badly misunderstood what the visual devices are intended to communicate. Given that the subjects had only 20 minutes to learn the entire environment and receive their task instructions, this was a possibility. To help assess their understanding, and provide some insight into the understandability of our methodology, we asked several questions about the meanings of the different devices. The results, which are summarized in Table 5, suggest that WYSIWYT subjects understood the features reasonably well.

Our methodology’s usefulness would be limited if a steep learning curve prevented benefits without a time-consuming initial effort. To gain some insights into the possibility of this problem, we compared the Ad Hoc and WYSIWYT subjects’ performance on the first problem (the left half of Table 6). As the table shows, despite whatever learning curve is associated with the method-

	Problem 1			Problem 2		
	Tested	# Tests	Redundant	Tested	# Tests	Redundant
Ad Hoc	69.0%	13	51.3%	71.6%	22	56.3%
WYSIWYT	82.7%	20	11.1%	97.8%	18	7.7%

Table 6: Learning Effects: compare the medians of Problem 1 and Problem 2.

ology, WYSIWYT subjects still had greater effectiveness and greater efficiency in the first problem than did the Ad Hoc subjects (Mann-Whitney, $p=0.0083$ ($p<0.0001$ (Redundancy))). Additional benefits came with experience for the WYSIWYT subjects. In particular, the WYSIWYT subjects learned to improve their testing on the second problem. They generated about the same number of test cases for each problem but significantly increased their coverage (by 15%) from the first problem to the second problem. In contrast, the Ad Hoc subjects, despite generating about one third more tests, did not significantly increase their coverage (Wilcoxon, $p=0.26$ (Ad Hoc), $p=0.0005$ (WYSIWYT)).

6 THREATS TO VALIDITY

In our experiment we attempted to address threats to internal validity by balancing the two groups of subjects according to year in school and class, by counterbalancing with respect to problem type, by equalizing training time, and by selecting problems from familiar domains. However, as in most controlled experiments, threats to external validity are more difficult to address given the need to control all other factors. For example, Computer Science students may not be representative of any sizable segment of the population of spreadsheet programmers. In particular, they cannot be said to be representative of end-user spreadsheet developers. Similarly, the spreadsheets used in the experiment may not be representative of the population of spreadsheets. However, although the spreadsheets may seem rather simple, given the limited testing time of the experiment, few subjects achieved 100% du-adequacy (Clock: 21.7%; Grades: 1.4%). To determine whether the results of this study generalize to a larger segment of the spreadsheet programming population and to other spreadsheets, we are planning to conduct additional studies.

As we have mentioned before, our methodology does not currently handle non-executable du-associations in a way that is helpful to the task of testing; yet, they do occur in spreadsheets. A large number of non-executable du-associations in a spreadsheet would be a barrier to the effectiveness of the methodology, and the experiment did not address this issue. Instead, we circumvented it to the extent possible by attempting to minimize the number of such du-associations in each spreadsheet: Grades contained 2 (out of 95), and Clock contained 10 (out of 83).

Because the focus of the study was on effectiveness and efficiency of testing, the spreadsheets contained

no faults. This may be unrealistic; however, including faults in the spreadsheets would have confounded the data about testing effectiveness and efficiency since the subjects would not be focused on the single task of testing the spreadsheets. To have a clear analysis of this task, we did not allow subjects to change formulas; including faults without allowing corrections would be even less realistic. A separate study of debugging tasks can be found in [6]. To motivate our subjects, however, the subjects were informed that their spreadsheets “might or might not” contain faults. In fact, several subjects did report faults of a cosmetic nature.

Testing effectiveness was measured by the percentage of the du-associations covered by the test cases, but this is not the only possible measure. Another measure of effectiveness is the number of faults detected; however, as discussed above, we chose not to include faults in the spreadsheets and therefore could not utilize such a measure. A correlation between effectiveness in terms of du-adequacy and effectiveness at finding faults is supported by evidence in empirical studies of imperative programs [8, 11, 27]) and in previous empirical work we performed in the spreadsheet paradigm [23].

7 CONCLUSION

Spreadsheet languages have rarely been studied in terms of their software engineering properties. This is a serious omission, because these languages are being used to create production software upon which real decisions are being 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 testing, that can help spreadsheet programmers determine the reliability of values produced by their spreadsheets.

In this paper we reported empirical results about a testing methodology aimed at this need. The results were:

- Subjects using the WYSIWYT methodology performed significantly more effective testing than did the Ad Hoc subjects, as measured by du-adequacy.
- Subjects using the WYSIWYT methodology were significantly more efficient testers than the Ad Hoc subjects, as measured by redundancy and speed.
- Subjects using the WYSIWYT methodology were significantly less overconfident than were the Ad Hoc subjects.

Further, it was possible for WYSIWYT subjects to achieve these benefits even without training on the underlying testing theory. These results are encouraging,

because they suggest that it is possible to achieve at least some benefits of formal notions of testing even without formal training in the testing principles behind a testing methodology. However, this experiment is the first step in a series of planned experiments, and thus includes only one slice of the population using spreadsheet languages. Future experiments will be required before it will be clear whether the methodology brings similar benefits to other kinds of users, especially end users, working on larger, more complex spreadsheets.

ACKNOWLEDGEMENTS

We thank the members of the Visual Programming Research Group for their help with experiment administration and implementation and their feedback on the testing methodology. We also wish to express our gratitude to the students in Fall 1999 CS381, CS411, and CS511 who participated in the study. This work was supported in part by NSF under CCR-9806821, CAREER Award CCR-9703108, and Young Investigator Award CCR-9457473. Patent pending.

REFERENCES

- [1] A. Ambler, M. Burnett, and B. Zimmerman. Operational versus definitional: A perspective on programming paradigms. *Computer*, 25(9):28–43, Sept. 1992.
- [2] F. Belli and O. Jack. A test coverage notion for logic programming. In *The 6th Intl. Symp. Softw. Rel. Eng.*, pages 133–142, 1995.
- [3] P. Brown and J. Gould. Experimental study of people creating spreadsheets. *ACM Trans. Office Info. Sys.*, 5(3):258–272, July 1987.
- [4] 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.
- [5] E. H. Chi, P. Barry, J. Riedl, and J. Konstan. A spreadsheet approach to information visualization. In *IEEE Symp. Info. Visualization*, Oct. 1997.
- [6] C. Cook, K. Rothermel, M. Burnett, T. Adams, G. Rothermel, A. Sheretov, F. Cort, and J. Reichwein. Does Immediate Visual Feedback about Testing Aid Debugging in Spreadsheet Languages? Technical Report TR: 99-60-07, Oregon State University, Mar. 1999.
- [7] E. Duesterwald, R. Gupta, and M. L. Soffa. Rigorous data flow testing through output influences. In *Proc. 2nd Irvine Softw. Symp.*, Mar. 1992.
- [8] 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.
- [9] P. Frankl and E. Weyuker. An applicable family of data flow criteria. *IEEE Trans. Softw. Eng.*, 14(10):1483–1498, Oct. 1988.
- [10] D. Gilmore. Interface design: Have we got it wrong? In K. Nordby, D. Gilmore, and S. Arnesen, editors, *INTERACT'95*. Chapman and Hall, London, 1995.
- [11] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *16th Intl. Conf. Softw. Eng.*, pages 191–200, May 1994.
- [12] W. Kuhn and A. U. Frank. The use of functional programming in the specification and testing process. In *Intl. Conf. and Wkshp. Interoperating Geographic Info. Systems*, Dec. 1997.
- [13] J. Leopold and A. Ambler. Keyboardless visual programming using voice, handwriting, and gesture. In *1997 IEEE Symp. Vis. Lang.*, pages 28–35, Sept. 1997.
- [14] G. Luo, G. Bochmann, B. Sarikaya, and M. Boyer. Control-flow based testing of prolog programs. In *The 3rd Intl. Symp. Softw. Rel. Eng.*, pages 104–113, 1992.
- [15] 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.
- [16] B. Myers. Graphical techniques in a spreadsheet for specifying user interfaces. In *ACM CHI '91*, pages 243–249, Apr. 1991.
- [17] F. Ouabdesselam and I. Parissis. Testing techniques for data-flow synchronous programs. In *AADEBUG'95: 2nd Intl. Wkshp. Automated and Algorithmic Debugging*, May 1995.
- [18] R. Panko. What we know about spreadsheet errors. *J. End User Comp.*, pages 15–21, Spring 1998.
- [19] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Trans. Softw. Eng.*, 11(4):367–375, Apr. 1985.
- [20] G. Rothermel, M. Burnett, L. Li, C. DuPuis, and A. Sheretov. A methodology for testing spreadsheets. Technical Report TR: 99-60-02, Oregon State University, Jan. 1999.
- [21] G. Rothermel, M. Harrold, J. Ostrin, and C. Hong. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Proc. Intl. Conf. Softw. Maint.*, pages 34–43, Nov. 1998.
- [22] G. Rothermel, L. Li, and M. Burnett. Testing strategies for form-based visual programs. In *The 8th Intl. Symp. Softw. Rel. Eng.*, pages 96–107, Nov. 1997.
- [23] 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.
- [24] 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.
- [25] G. Svendsen. The influence of interface style on problem-solving. *Intl. J. Man-Machine Studies*, 35:379–397, 1991.
- [26] G. Viehstaedt and A. Ambler. Visual representation and manipulation of matrices. *J. Vis. Lang. and Comp.*, 3(3):273–298, Sept. 1992.
- [27] E. J. Weyuker. More experience with dataflow testing. *IEEE Trans. Softw. Eng.*, 19(9), Sept. 1993.
- [28] 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 22–27, Mar. 1997.