

AN ABSTRACT OF THE THESIS OF

Laura A. Beckwith for the degree of Master of Science in Computer Science
presented on November 18, 2002.

Title: Reasoning about Many-to-Many Requirement Relationships in
Spreadsheet Grids.

Abstract approved **Redacted for privacy**

Margaret M. Burnett

Traditionally, research into end-user programming has focused on how to make programming more accessible to end users. However, few researchers have considered providing end users with devices to help improve the reliability of the programs they create. To help improve the reliability of spreadsheets created by end users, we are working to allow users to communicate the purpose and other underlying information about their spreadsheets using a form of requirement specifications we call “guards.” Guards were initially designed for individual cells but, for large spreadsheets, with replicated/shared formulas across groups of rows or columns, guards can only be practical if users can enter them across these groups of rows or columns. The problem is, this introduces many-to-many relationships, at the intersection of rows and columns with guards. It is not clear how the system should reason and communicate about many-to-many relationships in a way that will make sense to end users. In this thesis, we present the human-centric design rationale for our approach to how the system should reason about such many-to-many relationships. The design decisions are presented with their reasons gleaned from two design-time models—Cognitive Dimensions and Attention Investment—and from the users themselves in a small think-aloud study.

©Copyright by Laura A. Beckwith

November 18, 2002

All rights reserved

Reasoning about Many-to-Many Requirement Relationships in Spreadsheet
Grids

by

Laura A. Beckwith

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the degree of

Master of Science

Presented November 18, 2002
Commencement June 2003

Master of Science thesis of Laura A. Beckwith presented on November 18, 2002.

APPROVED:

Redacted for privacy

Major Professor/representing Computer Science

Redacted for privacy

Head of the Department of Computer Science

Redacted for privacy

Dean of Graduate School

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

Redacted for privacy

Laura A. Beckwith, Author

ACKNOWLEDGEMENTS

I would first and foremost like to express my gratitude towards my advisor, Margaret Burnett, for her support, encouragement, and guidance. I would also like to thank her for introducing me to an area of Computer Science that I have come to love, but had no experience with prior to meeting and working with her.

Thanks, also, to all members of the Forms/3 team, but in particular to those who helped out during my periods of “oh-my, I need ANOTHER screenshot and NOTHING is working!” I also extend this thanks to everyone else in Dearborn 201, who made working in the office more enjoyable.

Lastly, I’d like to mention a few of the many others in my life who have been extremely supportive as I completed my masters: my parents, Melissa, Fabio and Jenny.

This work was supported in part by the National Science Foundation under Awards CCR-9806821 and ITR-0082265.

TABLE OF CONTENTS

	<u>Page</u>
1. Introduction	1
1.1 Introduction	1
1.2 Forms/3 and Support for Guards.....	2
1.3 Related Work	5
1.4 The Problem Addressed by this Thesis.....	15
2. Experiment Design.....	20
2.1 What we hoped to accomplish by doing this study.....	20
2.2 Procedure.....	20
2.3 Subjects	24
2.4 Tutorial.....	25
2.5 Tasks	27
2.6 Spreadsheet problems.....	28
3. Results.....	34
3.1 Results of Question 1:.....	34
3.2 Results of Question 2:.....	39
3.3 Results of Question 3:.....	41
4. Applying the Results by Study Question	45
4.1 Do users regard having many-to-many relationships among guards and cells as being valid and useful?	45
4.2 How should multiple guards propagate?.....	51
4.3 What constitutes a conflict?	54
5. Conclusion	57
Bibliography.....	59
Appendices.....	63
Appendix A: Tutorial Materials.....	64
Appendix B: Subject Spreadsheets	65
Appendix C: Quotes.....	75

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1: Temp Spreadsheet.	3
2: Basics Grades Spreadsheet.	4
3: A Forms/3 temperature converter.	5
4: Grades Spreadsheet with WYSIWYT.....	6
5: Stagecast Creator.....	10
6: Match Forms.	15
7: Grades Spreadsheet, hand annotated.	17
8: Grades Spreadsheet from the study.	23
9: Temperature Conversion.	26
10: Second tutorial spreadsheet.....	27
11: Post-test Spreadsheet.	27
12: Experiment Grades Spreadsheet.	30
13: Sales Spreadsheet.	32
14: Wait Time Spreadsheet.	32
15: Conference Spreadsheet.	33
16: The Sales Spreadsheet for subject 5.	36
17: The Wait Time Spreadsheet for Subject 5.	37
18: Subject 3's Grades Spreadsheet.	38
19: Grades with Guards.	46
20: Guard conflict explanation.	47
21: Previous internal design.	48
22: New internal design.	49
23: Grades Spreadsheet with guards closed.	50
24: Guard Priorities.....	52
25: Grades Spreadsheet with priorities set.....	53
26: Value violation explanation.....	56

LIST OF TABLES

<u>Table</u>	<u>Page</u>
1.1: Design Constraints. The constraints our solution must follow.	19
2.1: Subject information summary.	25
2.2: Classification Schemes.	29
2.3: Spreadsheet classifications.	33
3.1: Answers: Did subjects remove the extra user guard when one existed? ..	39
3.2: Subjects' set-based reasoning or lack thereof.	43

LIST OF APPENDIX FIGURES

<u>Figure</u>	<u>Page</u>
27: Subject 1 Grades.	65
28: Subject 1 Conference.	65
29: Subject 1 Sales.	66
30: Subject 1 Wait Time.	66
31: Subject 2 Grades.	67
32: Subject 2 Conference.	67
33: Subject 2 Sales.	68
34: Subject 2 Wait Time.	68
35: Subject 3 Grades.	69
36: Subject 3 Conference.	69
37: Subject 3 Sales.	70
38: Subject 3 Wait Time.	71
39: Subject 4 Grades.	71
40: Subject 4 Conference.	71
41: Subject 4 Sales.	72
42: Subject 4 Wait Time.	72
43: Subject 5 Grades.	73
44: Subject 5 Conference.	73
45: Subject 5 Sales.	74
46: Subject 5 Wait Time.	74

Reasoning about Many-to-Many Requirement Relationships in Spreadsheet Grids

1. Introduction

1.1 Introduction

In recent years a number of authoring environments and other kinds of programming devices have become available to allow end users to do their own programming. In fact Boehm et al. projected the number of end-user programmers to be 55 million by 2005 while the number of professional programmers is expected to reach only 2.75 million [Boehm and Basili 2000]. Although end-user programming has received a growing amount of attention (one sign of which is the increased focus on end users at conferences such as at the Human Centric Computing Languages and Environments Conference), there has been little research into aspects of end-user programming beyond the programming part per se. Programming is only one part of the development process, and focusing on other aspects is important for reliability of the programs end users create. In fact, reliability is an issue in end-user programming, as shown by statistics about spreadsheets, a widely used type of end-user programming language. Panko compiled field audits done on spreadsheets and found that a disturbing number of spreadsheets have errors: a very conservative estimate is that 20%-40% of spreadsheets contain errors, and in some studies, as many as 91% of the studied spreadsheets had errors [Panko 1995, Panko 1998, Panko 2000].

We have been working on how to improve the reliability of end-user programs in general and of spreadsheets in particular. One of our hypotheses is that spreadsheet reliability can be improved if the spreadsheet users work collaboratively with the system to communicate more information about known relationships. Spreadsheet users know more about the purpose and underlying requirements for their spreadsheets than they are currently able to communicate to the system, and our goal is to allow end users to communicate this information

about requirements. This will allow for checks and balances, so that the system can detect and point out ways in which the spreadsheet does not conform to the user's requirements.

We are pursuing the question of requirement specifications for end users using the research spreadsheet language Forms/3 [Burnett et al. 2001]. In our prototype, we refer to requirement specifications as *guards* (guards are analogous to assertions for professional programmers). We began this work with an early prototype for individual cells, which afforded empirical investigations into how users problem solve in the presence of guards [Wallace et al. 2002]. From research conducted concurrently with what is reported here, we know guards significantly help users find and fix bugs [Burnett et al. 2002a].

The work presented in this thesis investigates scalable guard mechanisms. By "scalable," we mean guard mechanisms that are viable for end users when programming large spreadsheets. Typically, large spreadsheets contain grids of many cells with repeated patterns of relationships, often due to shared or replicated formulas across the rows or columns. Allowing users to place guards on grids (such as on rows and columns) can lead to overlapping guards, which may be difficult for users to reason about.

1.2 Forms/3 and Support for Guards

As mentioned, Forms/3 is the language in which we are prototyping our work. Forms/3 is a declarative spreadsheet language, although it varies from traditional spreadsheet languages. One of the most visible variations is the lack of a predefined grid layout that cells must belong to; cells can be placed anywhere within the form (see Figure 1). Although cells can be placed anywhere within the spreadsheet, there is also support for more structure in grids. In Forms/3 grids, rows and columns are determined by user-specified formulas. Grids can then be divided into regions of cells; with each region having one formula that applies to

all cells within that region. An example can be seen in Figure 2, where the cells in the average row are in one region and share the same formula.

The screenshot shows a window titled "Temp" with a dark header bar. Below the header, there are two icons: a floppy disk and a printer. The main area contains two input fields. The first field contains the number "45" and is labeled "Fahren" with a small downward arrow icon to its right. The second field contains the number "7.2222" and is labeled "Celsius". Below the "Celsius" label, there is a small dialog box with "Hide" and "Apply" buttons, and the formula $(\text{Fahren} - 32) * 5/9$ is displayed below the buttons. A mouse cursor is visible near the top left of the input fields.

Figure 1: Temp Spreadsheet. A simple Forms/3 spreadsheet that takes a Fahrenheit temperature and converts it to its Celsius equivalent. In Forms/3 a cell's value and formula can be viewed at the same time (Celsius's Formula is currently open). The tab on the top of the cells is how the users enter guards (which will be explained shortly).

	HomeWork	Midterm	Final	Course
Sam	78	94	68	80
Jenny	92	64	87	81
Average	85	79	77.5	80.5

Formula: $(\text{Grades}[10j] + \text{Grades}[20j]) / 2$

Figure 2: Basics Grades Spreadsheet. This Forms/3 Grades Spreadsheet will be a running example throughout the thesis. Each student has a grade for “Homework”, “Midterm”, and “Final” which are averaged in the “Average” row. The “Average” row is a region with four cells, and its formula (applying to all cells in the region) refers to the other cells in the grid.

Before scaling guards up to grids, guards pertained to only one cell, although their implications were propagated through formulas to other cells. That is, whenever a user placed a guard into a spreadsheet cell (a *user-entered guard*) that guard was propagated [Wallace et al. 2002] through formulas downstream generating *computer-generated guards* on downstream cells. A cell with both a computer-generated and user-entered guard was in a conflict state (a *guard conflict*) if the two guards are not identical. As Figure 3 shows, to communicate a guard conflict, the system circles the conflicting guards. Since the cell’s value is inconsistent with a guard on that cell (termed a *value violation*), the value is also circled.

Other relevant aspects of Forms/3 will be discussed as they are needed.

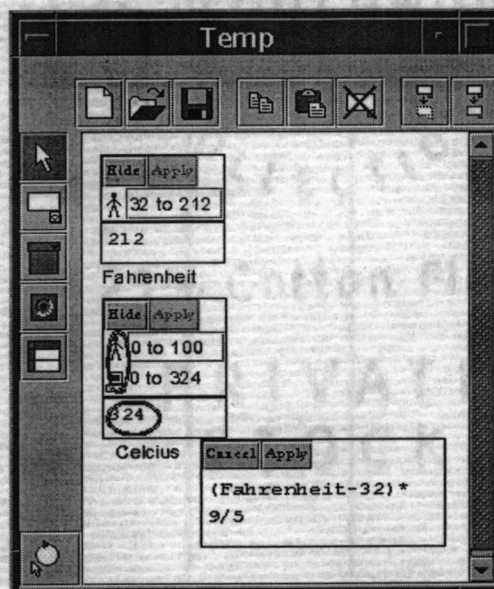


Figure 3: A Forms/3 temperature converter. Stick figure icons identify user-entered guards, and the computer icon identifies a computer-generated guard. The computer-generated guard's conclusion that the Celsius value ranges from 0 to 324 degrees provides a clue that there is an error in Celsius's formula.

1.3 Related Work

1.3.1 Software Engineering for End Users

The primary focus of research in end-user programming has been on the programming aspect, not on other aspects of the software engineering process. Nardi argues that end users need to have the power to program because, despite the best efforts of designers, it is impossible to know in advance what a user may need in a program [Nardi 1993]. One solution to this uncertainty of requirements is to give users the power to make customizations and add features that help them become more efficient in completing their work.

Guards are one aspect of an approach we are devising, termed software engineering for end users, aimed at helping end users improve the correctness of their programs. The first contribution of this research was a visual methodology for testing that allows users to incrementally edit, test, and debug their spreadsheets as their spreadsheets evolve [Burnett et al. 2002b, Reichwein et al.

1999, Rothermel et al. 1998, Rothermel et al. 2001, Krishna et al. 2001]. This approach, known as WYSIWYT (“What You See Is What You Test”), provides visual feedback in several ways about how much of a spreadsheet has been tested. The feedback the user receives is a change of color indicating the level of testedness, starting with red for untested and moving along in the spectrum toward blue (blue indicates that all adequacy criteria have been met). Figure 4 shows a spreadsheet using WYSIWYT. Some of WYSIWYT’s features have also recently been adapted for the visual dataflow paradigm of Prograph [Karam and Smedley 2001]. WYSIWYT and guards are seamlessly integrated into the Forms/3 environment and with each other.

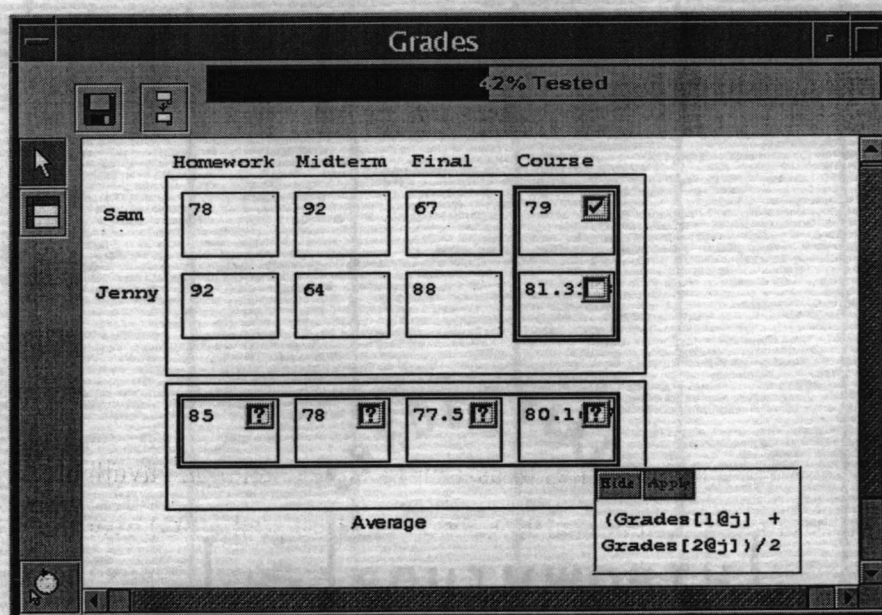


Figure 4: Grades Spreadsheet with WYSIWYT. In this spreadsheet the cells in the “Course” column have already been tested (they are blue), while the cells in the “Average” row have not (they are red). The percent testedness indicator at the top of the screen communicates how much of the spreadsheet has been tested thus far.

Other research into spreadsheets has focused on finding ways to use units (based on column and row headings) to help end users find errors [Burnett and Erwig 2002]. With their approach the user does not need to “declare” any unit

information, rather the row and column headers are used as the units, and the user is able to correct the system, via a programming by demonstration approach, if the system makes the wrong assumptions. By using units the system can detect some formula errors, such as a wrong cell or omitting some cells from aggregate calculations. These errors are detected by applying an underlying inference system for unit-based reasoning.

There is also research regarding helping end-user programmers find errors through outlier analysis [Miller and Myers 2001]. This work focuses on common maintenance tasks within text documents that can often lead to errors. For example, a “replace all” within a text document might change more than was intended, or might not replace everything intended if there were slight spelling differences in the document. The attention cost required for a user to check each change is often too much. Miller and Myers’s approach detects probable errors by a method analogous to statistical outlier detection. An empirical study showed that the approach did aid the subjects in completing their assigned tasks with fewer errors.

1.3.2 Can end users use assertions?

Various approaches to assertions have been made available to professional programmers [Ernst et al. 1999, Sankar and Mandal 1993] over the years, and while they are effective in finding runtime errors [Rosenblum 1995] they have not been geared toward end users.

Research indicates that end users can potentially work with some forms of requirement specifications. Nardi summarized work by several researchers indicating that, although end users are not particularly good at working with abstract requirements, they work much better with a concrete program they are able to criticize [Nardi 1993].

The need for some form of explicit requirement specifications is also indicated by a study Gray and Fu conducted. They found that if people have a

vague recollection of requirements, they will not take the time to look them up in another document [Gray and Fu 2001]. For example, in the task of programming a VCR to record a television show, those who had memorized the times to program made significantly fewer mistakes than those who had seen the information and had access to it, but had not actually memorized it. Instead, those who had not memorized the information relied on their recollections rather than doing the extra work to access the information. Gray and Fu refer to this as “perfect knowledge in-the-world” versus “imperfect knowledge in-the-head.”

1.3.3 Grid-based Programming Languages for End Users

Some of the programming systems for end users use grids as part of the programming. Since our work focuses on reasoning in the presence of grids, the following discussion focuses on research surrounding grid-based languages. We have categorized grid-based languages into spreadsheet languages and non-spreadsheet languages.

Spreadsheet research varies from extending the traditional uses of spreadsheets to making the dependencies within a spreadsheet more visible and using the spreadsheet paradigm to address the issue of information visualization. Similarly to Forms/3 the programming language Formulate [Ambler and Broman 1998, Ambler 1999] builds on the traditional spreadsheet model. In Formulate users program by solving simple equations. Like Forms/3, not all of Formulate is grid-based; grids are used to represent arrays, lists and tables. Ambler and Broman stress that users do not need to worry about indexing arrays or lists; nor do they need to program loops to iterate through all elements within a data structure. Users can, instead, manipulate the data structures (represented by grids) by defining partitions, called “regions.”

One potential difficulty spreadsheet users face is trying to remember the relationships between cells, which can be especially difficult when only the cell values are showing. Igarashi et al. designed some visualization techniques for

spreadsheets making the relationships between spreadsheet cells more apparent [Igarashi et al. 1998]. Their method involves a new view of the spreadsheet, the “dataflow” view, where the arrows mark the dataflow of the spreadsheet. They also added a mouse over feature that brings up lines indicating what cell(s) affect the current cell, and what cell(s) are affected. Forms/3 has similar features; for example, arrows show the dataflow of the spreadsheet. Additionally, in Forms/3 both a cell’s value and formula can be seen at the same time, along with any number of arrows the users chooses to view; this differs from Igarashi’s work where users can view arrows and values together, but not all three aspects.

Chi et al. have used a spreadsheet approach to visualizing large amounts of data [Chi et al. 1997]. In this case, the spreadsheet based approach can be considered grid-based, as the data (they used graphics in one example) is represented in cells of a grid. The formula for a cell indicates how to display the data, or whether to do any transformations to the data before displaying it.

Grids are used not only in spreadsheets. AgentSheets [Repenning 2000, Repenning et al. 2000, Repenning and Citrin 1993], Stagecast [Cypher and Smith 1995, Seals et al. 2002], and Kara [Hartmann et al. 2001] are all (non-spreadsheet) languages that use grids. Agentsheets combines grid-based programming, end-user programmable agents and Java authoring to allow end users to build simulations. Agents are programmed to respond to specific conditions (these conditions and responses are called rules and are programmed by dragging and editing conditions and actions stored in tool palettes). The agents are placed into a grid, and as the program runs, the agents respond to the environment and can move around within the grid. Agentsheets was a sibling to another simulation programming language, called Stagecast Creator (previously called KidSim and Cocoa) [Cypher and Smith 1995]. Stagecast is “based on a movie metaphor where users create a cast of characters who interact and move within a simulation microworld” [Seals et al. 2002]. The grid-based system is where all of the characters reside, similar to Agentsheets. Stagecast characters are programmed by

demonstration, where before and after pictures define the rules, which differs slightly from how the agents are programmed in Agentsheets. A picture of a Stagecast world can be seen in Figure 5. Kara [Hartmann et al. 2001] uses a grid similar to Stagecast and Agentsheets, where characters move around within the grid following some programmed instructions. Kara, though, is meant to represent finite state machines and is limited to four types of objects; only one of these objects can be programmed to react to its surroundings. The purpose of Kara, according to the authors, is to be used for programming instruction.

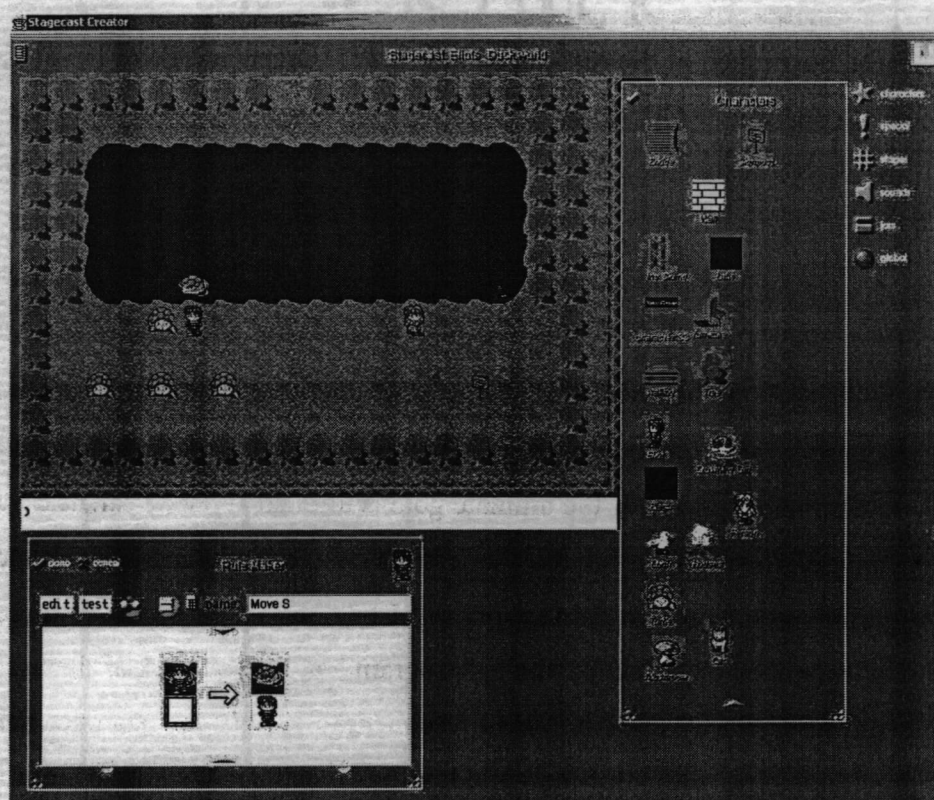


Figure 5: Stagecast Creator. In this picture the large box in the upper left corner is the “World,” a grid that the characters are placed in; when the user presses play (not shown) the characters respond according to their rules. Just below it is an example of a rule. The box on the right side of the picture shows all the available characters for the world.

1.3.4 Fundamental Principles

When language designers/researchers design a new language there is a body of research they can rely on for evaluating their design. The first of these are the Cognitive Dimensions (CDs) [Green and Petre 1996].

Our design was guided in part by our use of the CDs during the design process. CDs are a set of factors that help designers to assess usability at design time. The CDs are not rules, but instead provide vocabulary with which to talk about design. One example is consistency: “When some of the language has been learnt, how much of the rest can be inferred?” [Green and Petre 1996]. In addition to the consistency CD, other CDs that impacted our design include visibility, progressive evaluation, abstraction gradient, and premature commitment, as will be seen.

Another influence on our design was Attention Investment [Blackwell and Green 1999, Blackwell 2002]. Attention Investment is an analytic model of user problem-solving behavior that allows a designer to consider the costs, benefits, and risks users weigh in deciding how to complete a task. For example, consider a programmable phone. If the ultimate goal is to make a phone call, then programming the number into the phone has a cost, benefit, and risk. The cost is figuring out how to program the phone. A benefit is the freedom to forget the phone number. The risk is that the “program” might not work as the user intended. In our research, we use Attention Investment to guide our design decisions toward providing users with a low cost and low-risk mechanism whose benefit will be a higher probability that their programs’ (spreadsheets’) errors will be automatically detected and brought to their attention.

Recall Gray and Fu’s research [Gray and Fu 2001] on “perfect knowledge in-the-world” versus “imperfect knowledge in-the-head;” we propose an explanation for the subjects’ behavior from an Attention Investment perspective: Users simply want to be efficient. That is, even when the information was accessible, users would still lose time retrieving the needed information, such as

by context switching from working with the spreadsheet system to finding the right document and looking things up in it. Our approach to guards attempts to eliminate some use of imperfect knowledge in-the-head by making perfect knowledge in-the-world time-efficient to access in the same context as the spreadsheet.

1.3.5 Studies

Some aspects of end-user programming languages have been studied empirically and can be divided into two categories: formative and summative. Formative studies are conducted prior to design and help researchers form the design. Summative studies, on the other hand, are conducted post design and are generally used to measure users' improvement in some specific task which can be attributed to using the new design. The following sections cover a variety of summative and formative studies in end-user programming.

1.3.5.1 Summative studies

Outside of the summative studies we have conducted with end users, there are a number of other summative studies. The Forms/3 research team has done summative studies evaluating many aspects of our system, including WYSIWYT, guards, and a feature called "Help Me Test" (helping users find new testcases) [Krishna et al. 2001, Wallace et al. 2002, Burnett et al. 2002a, Wilson et al. 2002]. Outside of Forms/3 other empirical studies on end users have involved children, teachers, and other community members (such as the studies on Agentsheets and Stagecast Creator [Rosson and Seals 2001, Seals et al. 2002, Rader et al. 1997]). The following discussion covers a few summative studies and their results.

A study by Engebretson and Weidenbeck [Engebretson and Weidenbeck 2002] looked at whether end-user programs written with task-specific constructs aids subjects' comprehension. The subjects were teachers and the programming language was a common end-user programming language for teachers (HyperCard). In the first part of the study the teachers were asked to examine

code and answer questions regarding what the code would do when run. In the second part of the study, teachers we asked to modify the code. This study did not actually evaluate a particular programming language design or method, but rather examined the effects the differences of the actual code had on users' comprehension (task-specific versus non-task-specific constructs). They found that teachers with the task-specific programming constructs completed the programming tasks and questions more successfully.

As previously mentioned, guards are part of a larger methodology for end users, including WYSIWYT. Our research team has performed summative studies on WYSIWYT that revealed subjects performed significantly better in aspects of testing, debugging, and maintenance tasks with the help of WYSIWYT [Krishna et al. 2001].

In addition to the WYSIWYT studies we have also conducted two studies investigating guards. The first of these studies was a think-aloud study designed to investigate if end users were really able to understand and use guards, and whether they were distracted by the red ovals the system uses to make known inconsistencies [Wallace et al. 2002]. The examiners found that subjects were able to understand and use guards effectively, and that the subjects did not seem to be distracted by the system's marking inconsistencies in the spreadsheet. The subjects would simply attend to those problems, as they were ready.

The second study, investigating whether guards helped users find and debug their spreadsheets, was conducted concurrently with the work reported in this thesis [Burnett et al. 2002a]. The results of this study were promising; subjects who were supplied guards¹ performed significantly better on finding and correcting errors in their spreadsheets.

During the first two studies investigating guards, the subjects were given spreadsheets that already had guards entered. The third study [Wilson et al. 2002]

¹ The subjects did not actually need to enter the guards themselves; the spreadsheet they were given already had guards, which the subject could choose to display.

examined if subjects could be enticed to enter guards on their own using a surprise-explain-reward strategy. We used the surprise aspect to get the users attention, and then took the opportunity (though tool tips in a non-intrusive manner) to explain guards (guards were not taught as part of the tutorial), including communicating the possible rewards of using guards. By means of this strategy 94% of the subjects entered guards.

1.3.5.2 Formative

Summative studies help in post-design evaluation and improvements. In this thesis, we conducted a formative study to inform our design from the start, basing our design on choices our subjects made during the study.

Research by Pane illustrates one example of an exceptional set of formative studies that guided his design of an end-user programming language. When Pane started his research into a new programming language for end users, his goal was to “elevate” usability to one of the most important aspects of the programming language. In other words, Pane conducted many formative studies; his findings were then incorporated into the design of his new language, HANDS.

One significant finding of the studies Pane conducted was the subjects’ use of Boolean expressions. End-users understand the meanings of the words “and,” “or,” and “not” differently from the traditional view of computer scientists. In order to approach this problem Pane and Myers designed a non-textual query form [Pane and Myers 2000] (see Figure 6) and they compared this design against textual versions that avoided the troublesome AND and OR operators. This study was both summative and formative. Since they were evaluating two new designs the study is summative. However, because they used the findings about issues end users face when dealing with Boolean expressions (to inform the design of HANDS) the study was also formative.

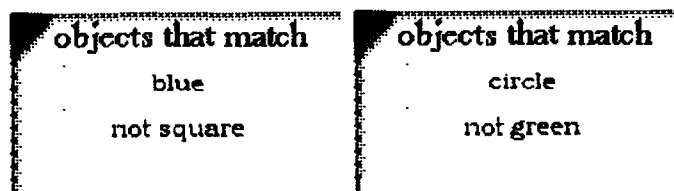


Figure 6: Match Forms. The query language designed by Pane and Myers to help avoid the common pitfalls end users have with traditional Boolean expressions. They have illuminated the words “and” and “or” by using these boxes where elements inside are placed together with an “and”, and the two boxes are placed together with an “or” forming one query.

Pane et al. conducted a series of studies prior to the study above which contributed to motivation for the aforementioned study. They examined how non-programmer devised solutions to programming problems [Pane et al. 2001]. These purely formative studies examined solutions of both children and adults. Some of the highlights of the results of these studies include indications that the traditional model of variables (whose contents are not available except when the program is running, and then often difficult to access) is not ideal for end users. To address the problems with traditional variable rules, Pane has developed a concrete model where variables exist whenever they are visible. A recommendation Pane made was that the syntax of the language should be something the users are already familiar with, for example using text such as “add 100 to score” instead of “score = score + 100.” He also found that users are more likely to use aggregate operations instead of performing loops on a dataset. The results of how these series of studies ultimately helped shape the programming language HANDS can be found in [Pane et al. 2002].

1.4 The Problem Addressed by this Thesis

A challenge in scaling up guards to support grids is finding a reasoning mechanism that will be understandable and sensible to end users. It is this challenge that this thesis sets out to address.

Until now, our system allowed only one user-entered guard per cell, and communicating with the user about the one-to-one relationship between a user-entered guard and its cell was relatively straightforward. However, multiple user-entered guards per cell seem necessary in grids. For example, a user may need to specify a guard on a row and another on a group of columns, and these guards would overlap on at least one cell. The issue is how to reasonably handle multiple relevant guards—many entered directly by the user on a row and a column—that pertain to the same cell where the row and column intersect. The reason allowing the user to enter guards for entire rows and columns at a time makes choosing an understandable reasoning mechanism difficult is that the new feature introduces not only one-to-many (one guard for several cells) but also many-to-one (several guards applicable to one cell) relationships—and hence, many-to-many relationships.

For example, suppose the user specified that the Homework, Midterm, Final, and Course columns of Figure 7 all must be between 0 and 100, that the Average grid (row) must be between 0 and 100, and that the last column of Average should be the average of the previous columns. (Not all of these specifications are depicted in the figure.) This last specification would crosscheck Average's formula, which instead computes the last column as the average of the Course column. Such multiple guards give users more ways to enter checks and balances.

	Homework	Midterm	Final	Course
Sam	78	94	68	80
Jenny	92	64	87	81
Average	85	79	77.5	80.5

Figure 7: Grades Spreadsheet, hand annotated. The same Forms/3 Grades Spreadsheet from previous example, hand annotated to illustrate issues arising from many-to-many relationships.

One problem is how to define the notion of these multiple guards being in conflict. In our early prototype, we used a “must exactly match” rule, but this rule may not suffice in the presence of many-to-many relationships. For example in Figure 7, we can imagine the user of the spreadsheet wanting to know when Sam’s grades fall below a 70, because Sam requires special monitoring. To do this the user would put a guard of 70-100 on Sam’s row. If there were already guards on the columns that all grades are between 0-100, this student would have multiple user-entered guards on all cells in this student’s row that do not match exactly. Should this be considered to be a conflict among guards?

One possibility, for the above example, is to allow one guard to be a subset of another without conflict. However, this would prevent the system notifying the user of inconsistencies among their guards. The above student example was an inconsistency the user wanted; on the other hand, it could instead have been a mistake, and for the system to simply ignore that inconsistency would diminish the power of the system to provide feedback about errors.

So, either decision made by the system is incorrect for some cases.

Another possible approach is to have the user make the decision. If the user makes all the decisions, such as if two guards should match exactly or if subsets are acceptable, the benefit is the decisions are the ones the user wants, but the cost is the time the user must spend doing the deciding. We could cut the cost by having the system make the decision for the user, but this runs the risk of later cost due to potentially bad decisions.

In researching possible ways to reason about multiple user-entered guards we developed five design constraints (which determine “rules” for possible solutions to the above issues), which draw from several researchers’ work relevant to end-user programming [Belkin 2000, Corritore et al. 2001, Green and Petre 1996, Blackwell 2002].

Design Constraint 1 is that the system must immediately display the presence of inconsistencies and conflicts involving guards. From literature on on-line trust and its impact on the usefulness of on-line systems [Corritore et al. 2001], it is clear that if users can trust our system to notify them when there is a logic error, they will be more likely to provide the system with the information it needs to provide these notifications. Immediate display also relates somewhat to the visibility CD, which refers to how easily users can view components as a whole; poor visibility would be if in order to view all parts of a component the user must do so one by one.

Design Constraint 2 is to handle all similar situations consistently. This design constraint is drawn from the consistency CD. Treating similar situations consistently also helps with predictability, which helps to build trust.

Design Constraint 3 is that users should feel they understand the system’s reasoning. This design constraint is important for trust, which in turn promotes effective use [Belkin 2000, Corritore et al. 2001].

Design Constraint 4 is to not demand unwarranted attention from the user. Drawn from the model of Attention Investment, this constraint means that the

system will leave control of a user's problem-solving agenda up to the user. For example, the system will not pop up dialog boxes demanding immediate answers, will not trap users in modes, and will not require actions to be performed in a particular sequence (which also relates to the premature commitment CD).

Design Constraint 5 is that all algorithms must be fast enough to maintain immediate visual feedback. This is a corollary to Design Constraint 1. It is also tied to the progressive evaluation CD, which is about the concept of immediate visual feedback after an edit.

The design constraints, summarized in Table 1.1, were used to help shape the approach, as will be seen throughout this document, but they did not provide answers to the following issues, which are fundamental to how the system should reason in the presence of many-to-many relationships:

- Do users regard having many-to-many relationships among guards and cells as being valid and useful?
- How should many-to-many user guards propagate?
- What constitutes a conflict?

To investigate these issues, we turned to the users themselves.

System must:	
1	Display inconsistencies and conflicts involving guards
2	Handle similar situations consistently
3	Be comprehensible to users
4	Not demand unwarranted attention from the user
5	Maintain immediate visual feedback

Table 1.1: Design Constraints. The constraints our solution must follow.

2. *Experiment Design*

Experiment Questions:

1. Do users regard having many-to-many relationships among guards and cells as being valid and useful?
2. How should many-to-many user guards propagate?
3. What constitutes a conflict?

2.1 What we hoped to accomplish by doing this study.

We decided to conduct a study for two reasons. First, although the design constraints, covered in Chapter 1, answered some questions about our design they were not sufficient in answering the above questions. Second, we wanted to obtain information directly from the audience that will be using our approach.

Using empirical studies to evaluate a language is not uncommon. However, most are conducted late, after the language has been designed, and are used to improve the currently existing design, not to guide the initial design. Conducting an empirical study late has two possible weaknesses that we hoped to avoid. First, changing a system drastically as a result of a study's findings can be costly, if done after the system has already been implemented. The second weakness has to do with how subjects react during the study if they feel the system has a "finished" appearance; [Landy and Myers 2001] have shown that subjects are less likely to criticize systems that appear finished.

2.2 Procedure

We conducted a think-aloud study. A think-aloud study is closely related to a protocol analysis. The primary difference lies within the treatment of collected data. In general, both think-aloud studies and protocol analyses are well suited for learning qualitative information about behaviors such as intermediate steps or strategies employed and why. A critical part of such studies is the subjects' thinking-aloud; their voicing of what they believe is happening, or the

reasons they are acting in a particular way. During such a study the examiner watches and observes a subject's actions. Most of the literature [Dix et al. 1993, Ericsson & Simon 1984] states the examiner should be involved with the subject as little as possible. For example, the only communication between the examiner and the subject should be the examiner prompting the subject to continue talking when the subject falls silent. However, there is a second school of thought [Boren and Ramey 2000] that recognizes there is already a relationship between the subject and the examiner, and ignoring this is unnatural. Our study leaned more toward recognizing the relationship between the subject and examiner. In this second school of thought the examiner asks questions relating to what the subject is doing, although when asking questions the examiner needs to exercise care that the questions do not influence subjects' actions. Carefully selected questions, the authors suggest, will only add to the information obtained from the study and will not negatively affect the study.

Think-aloud studies and protocol analysis diverge when it comes to their treatment of collected data. In protocol analysis the data is classified into different categories, and statistically analyzed based on those classifications. By contrast a think-aloud study is not collecting statistics, but rather behaviors. For example, what did each subject do when they saw a particular situation? This often involves a range of behavior patterns, and in our analyses of the data we were more interested in the range of behavior patterns, not the number of subjects who followed a particular behavior pattern.

Prior to conducting the study with subjects we did a cognitive walkthrough. The purpose of a cognitive walkthrough [Ko et al. 2002] is to help ensure that the research questions will be answered by the users' tasks. One of the main goals of a cognitive walkthrough is to remove any confounding factors from the study. Confounding factors in a study can make it impossible to answer the research question from the data collected.

The cognitive walkthrough we conducted led to several major changes in our study's design. The biggest set of changes from the cognitive walkthrough was specifying exactly what we needed the subjects to do. For example, we considered whether we were more interested in how the subjects handled one-to-many guards or in how subjects decided on the value of the system generated guard given a guard conflict upstream from this cell. In the end we decided we were more interested in how subjects handled one-to-many situations; to make sure we obtained this information we needed to be very explicit in our tasks. We needed to tell subjects to place a guard on a particular row if that is what we wanted them to do. We made this change.

Another change we made from the cognitive walkthrough was changing the size of the spreadsheets. Many of the spreadsheets were needlessly complicated and large. Since we were not attempting to test subjects on how they comprehended large spreadsheets we decided this was a factor that would simply complicate getting answers to our questions. We reduced the size of the spreadsheets by taking out extra rows and columns that were not essential to answer our study questions. We made several other changes, many of which will be mentioned throughout this chapter.

After conducting the cognitive walkthrough but prior to conducting the actual study, we ran through the study using a pilot subject. The goal of a pilot subject is help find unforeseen problems that may appear during the actual study. The combination of the pilot subject and performing a cognitive walkthrough of the study aided in finding, and then correcting problems that might have influenced the actual study.

The think-aloud study of five subjects was conducted one-on-one in a small study room. We conducted the study using Excel-like grids with sketchy icons on paper such as the one shown in Figure 8. Our reason for a paper-based study was to avoid restricting the users to only those possibilities we had managed to predict in advance. Our reason for the drawings' informal appearance and use of hand annotations to develop the problems was to encourage the subjects to freely criticize and change the system's reasoning; as mentioned before, research has shown that subjects are less likely to criticize software that has a "finished" appearance (e.g., [Landay and Myers 2001]). We decided to use Excel instead of Forms/3 for several reasons; first, it was just as easy to use either since the paper-based study did not require an implementation. More importantly, Excel had the advantage of staying as close to these users' previous experiences as possible, which helps avoid some kinds of confounding factors.

We tape-recorded the sessions, and also kept their paperwork. Pen color was changed between each task to differentiate the work done during each task. Prior to the cognitive walkthrough, we had planned to give subjects a fresh spreadsheet after each task. We decided instead to change pen colors, since it was important that decisions they made in earlier tasks still be present for later tasks; additionally, the subjects did not have to reacquaint themselves with a new piece

	A	B	C	D	E
1		Homework	Midterm	Final	Average
2	Sam	☹ ↓ 0-100	☹ ↓ 0-100	☹ ↓ 0-100	☹ ↓ 0-100 =(B2+C2+D2)/3
3	Jenny	☹ ↓ 0-100	☹ ↓ 0-100	☹ ↓ 0-100	☹ ↓ 0-100 =(B3+C3+D3)/3

Figure 8: Grades Spreadsheet from the study. The top half of each cell shows the guard. The stick figure versus computer indicates whether the user or the computer placed the guard on the cell. The bottom half of each cell has space for the cell's value (which was written in interactively during the experiment), and shows the cell's formula if one is present, such as in the Average column. Guards with down arrows were replicated down the entire column.

of paper several times.

During each problem we needed to hear subjects' reasoning when they ran into problems they were not sure how to handle. The strength of the think-aloud method is it allows us to capture the details of their reasoning. If the subjects were quiet for any length of time the examiner asked "What are you thinking?" or "Why?" to prompt them to resume speaking. When subjects asked the examiner for help, they were simply instructed to refer to the problem description. If the examiner did not understand the way users expressed their statements, she simply encouraged them to keep talking, using the same prompting questions as above. If the users' actions or statements inspired questions the examiner wanted to ask, these questions were saved until the question period at the end of all the spreadsheet tasks. This procedure was because the examiner did not want to influence users' answers to upcoming tasks. The examiner also interviewed the subjects after they had completed all the tasks.

2.3 Subjects

The subjects were students from majors that do not entail computer programming, namely Nutrition, Health Promotion and Education, and Soil Science. All the subjects had previous spreadsheet experience. All subjects were native English speakers. Four of the five subjects were female, one male. The Table 2.1 gives additional information on each subject.

Subject	Major	GPA	Year	Programming	Spreadsheet use
1	Nutrition & Food Management	3.0	Senior	High school	College, personal
2	Health Promotion & Education	3.14	Sophomore	High school	High school, college, professional, personal
3	Nutrition & Food Management	3.80	Senior	High school	College
4	Agronomy	3.7	Graduated	None	Professional
5	Dietetics	3.0	Senior	None	College

Table 2.1: Subject information summary. Shows each subjects' major, GPA, Current year in school, computer programming experience, and reported uses of spreadsheets.

2.4 Tutorial

The experiment began with some practice thinking aloud. The examiner read a short statement to the subjects describing the subjects' role in a think-aloud study. After this the subjects were given two tasks, both non-mathematical, because some end-user subjects of our previous experiments have become anxious when asked to perform mathematically oriented tasks. The first task included a short paragraph with inserted errors such as spelling, grammatical, and nonsense meaning; this paragraph can be found in Appendix A. The subjects were asked to read the paragraph aloud and fix any mistakes they found along the way. After this first task the examiner either moved onto the second think-aloud practice (noting to the subject that he/she did a good job), or worked with the subject to let them know more clearly what she was expecting. The second task was to name the states beginning with the letter 'A' that people could ski in. This task required a two-step process, first to think of the states that start with an 'A', then to determine whether those states have skiing. As with the first practice, if the subjects did not think aloud the examiner would let them know what it was that she was hoping to hear from them.

The second part of the tutorial introduced guards. During this part of the tutorial the examiner encouraged subjects to continue thinking aloud and to ask

any questions they had. The examiner first worked through the temperature conversion problem of Figure 9. This covered the basics of the guards: guard propagation (for a simple problem), value conflicts, and guard conflicts. After explaining the basics to the subjects the examiner asked the subjects if specific values would result in value conflicts, and how they might resolve the guard conflict on the spreadsheet. If the subjects appeared to be unclear on any point the examiner would go over this information again, until it appeared the subjects understood. It was important that each subject understand the concepts covered in the tutorial; tutorial times ranged as little as 5 minutes for one subject and as much as 15 minutes for another.




	A	B	C
1		 32-212	 0-100  0-100
		32	0 =(B1-32)* 5/9

Figure 9: Temperature Conversion. The examiner worked through this spreadsheet introducing guards, propagation, guard conflicts and value conflicts.

The second spreadsheet in the tutorial, Figure 10, kept track of the number of calories a patient in the hospital was supposed to consume for each meal. This spreadsheet was used to introduce grids and column guards on grids. The purpose of this spreadsheet was to explain the reasoning of guards within a spreadsheet with rows and columns. Initially we had planned using two spreadsheets, but a third spreadsheet was added to the tutorial after the cognitive walkthrough, when we realized the subjects never saw how to place guards on rows. The third tutorial spreadsheet introduced row guards on grids and how to use a comma to represent non-range guards.

	A	B	C	D	E	F
1		Breakfast calories	Lunch Calories	Dinner Calories	Snack Calories	Total
2	22-Dec	300-500	200-320	200-320	50-100	750-1240
3	23-Dec	300-500	200-320	200-320	50-100	750-1240
4	24-Dec	300-500	200-320	200-320	50-100	750-1240
5	avg	300-500	200-320	200-320	50-100	750-1240
		=SUM(B2:B4)/3	=SUM(C2:C4)/3	=SUM(D2:D4)/3	=SUM(E2:E4)/3	=SUM(F2:F4)

Figure 10: Second tutorial spreadsheet. During this spreadsheet the examiner introduced the subjects to grids and to column guards.

After finishing all tutorial tasks the examiner had the subjects complete a short practice spreadsheet Figure 11. The subjects were given a description of the spreadsheet along with the guards for the spreadsheet and were asked to place the guards on the spreadsheet and to compute the computer-generated guard. The examiner addressed any difficulties the subjects had with this task before moving to the experimental task.

	A	B	C
1		Fred Meyer	Safeway
2	Gum		
3	Milk		
4	Oranges		
5	Total	=SUM(B2:B4)	=SUM(C2:C4)

Figure 11: Post-test Spreadsheet. This is the blank copy of the final spreadsheet the subjects used as their post-test. They were asked to place guards on the rows of the spreadsheets to ensure the spreadsheets' conformance to particular specifications given to the subjects.

2.5 Tasks

After the above preliminaries the following experimental tasks were assigned for each spreadsheet:

Task 1: Place a specific guard on a specific row of the spreadsheet.

Task 2: Make the spreadsheet work as described in the problem description. This required making decisions about guards that would make sure “bad” values would not go unnoticed.

Task 3: Play the role of the computer to determine the correctness of values interactively specified by the examiner.

Task 4: (If any computer guards were missing, due to subjects’ spreadsheet changes): Play the role of the computer to fill in the missing computer guards.

Task 5: Given the scenario that someone else had worked on the spreadsheet and had left a particular set of (multiple) guards on the cells, and were asked what, if anything, needed to be changed in the spreadsheet.

2.6 Spreadsheet problems

The four spreadsheets used in the experiment were: Grades, Wait Time, Sales and Conference. Each subject saw these spreadsheets in a different order. The figures of each spreadsheet are included with detailed descriptions of each later in this section.

In preparing to design the problems and then to analyze the results of users’ reasoning about guards, we predicted two sets of patterns we thought we might see, and discovered a third set during the study. We will refer to all three sets of patterns as “classification schemes”. See Table 2.2. We designed the experiment’s problems in a way that probed whether users based their reasoning decisions on these schemes.

Classification Schemes	
Guard purpose	Do users reason about guards based upon whether guards are used in these different ways: “hard”: value is wrong if outside guard. “external”: guard due to something outside the spreadsheet. “data exception”: guard is used to protect against unreasonable values, guard is logically based on spreadsheet purpose.
Set reasoning	Do users use set-based reasoning in making decisions about guards, and if so what kind? (intersection, union, subset, etc.)
All-knowing computer	Is the correctness of values always determined by the computer guard? (yes or no)

Table 2.2: Classification Schemes. This table summarizes the three classification schemes.

The first classification scheme is by “guard purpose.” People often use features in ways not originally intended by their designers, regardless of whether their designers approve of these uses. In considering ways to understand the reasoning patterns users followed, we thought of three hypothetical purposes that might motivate users to use guards. Although we predicted three possible purposes, we did not limit ourselves to these in our analysis of the results.

For example, consider Figure 12, the Grades Spreadsheet, which was one of the problems in the experiment. The midterm score must be between 0 and 100 points. Therefore a guard of 0-100 on this column is said to have a guard purpose of being “hard”: any values that fall outside the range are wrong. The other two purposes are not “hard” in that they do not mean that a value is definitely incorrect. For example, a teacher might place a 70-100 guard on a student’s row in order to especially monitor that particular student’s progress. We term this guard purpose as “external”, because it was derived from information that is external to the spreadsheet, such as the reason this student needs to be monitored. (It is not “hard” in that it does not indicate that values are incorrect.) The third purpose we hypothesized was that a guard might be used to notify spreadsheet users of “data exceptions”. A “data exception” guard could be used to call unusual values to the user’s attention, such as a salesperson selling an unusually high or low amount compared to his/her usual average. It is different from an “external” guard

because this kind of a guard comes from data that is explicitly present in the spreadsheet.









	A	B	C	D	G
1		Homework	Midterm	Final	Average
2	Sam	 0 - 100	 0 - 100	 0 - 100	 0 - 100
					$=(B2+C2+D2)/3$
3	Jenny	 0 - 100	 0 - 100	 0 - 100	 0 - 100
					$=(B3+C3+D3)/3$

Figure 12: Experiment Grades Spreadsheet. One of the spreadsheets each subject saw during the experiment. The guard on the columns, such as the midterm, is said to have a guard purpose of being “hard”: any values that fall outside the range are wrong.

The second classification scheme examines whether subjects reason about guards in some set-based way. For example, if there are two non-matching user-entered guards on one cell, do users want to combine these into one guard using union or intersection? In the design of our spreadsheets we created a variety of conflicts that were intersections or disjoint to see how users would handle each case.

The third classification scheme was noticed after the fact, as a result of our analysis. We present it here to keep the discussion of classification schemes together. The scheme is binary, and considers whether our subjects simply decided that the computer-generated guard was always right. We term this kind of reasoning the “all-knowing computer”. An example is if a cell had both user-entered and computer-generated guards that were different, and a subject chose to follow whatever the value of the computer guard was in determining the correctness of a value.

Because we thought of two of the three approaches in advance, we were able to devise spreadsheet problems that offered a variety of set-based relationships among guard values and whose guards had a variety of purposes.

Each spreadsheet provided information to help answer the study questions, and each was different with respect to the classification schemes.

Grades: The Grades Spreadsheet, shown in Figure 12, was a simple spreadsheet which included homework, midterm, and final scores as input cells. The Average column used a basic formula to calculate the average of the input cells. This spreadsheet, when the subjects first saw it, had user-entered guards on the first three columns; the Average column had a system-generated guard. The subjects' first task was to place the guard 70-100 on Sam's row because, as described in the problem description, he was on academic probation and the teacher wanted to make sure his grades did not fall below 70. The cognitive walkthrough helped us realized that we needed to directly tell the subjects what to do. We wanted them to place another guard on the spreadsheet so we could see how they reacted when two user-entered guards intersected. From this task we hoped to gain understanding regarding what the subjects do with conflicts. We knew from pervious work by our colleagues [Wallace et al. 2002] that subjects understood user-computer conflicts, thus, our tasks were designed to see how they reasoned about user-user conflicts.

The classification for the Grades Spreadsheet can be found in Table 2.3.

Sales: The Sales Spreadsheet (Figure 13) calculated salespeople's bonuses based on their sales. During this experimental spreadsheet task subjects were asked to use the table at the top of the spreadsheet to determine an appropriate guard, and then place this guard onto the salespeople's rows in the lower part of the spreadsheet. Note that the "Adjusted Salary" column has two guards on it, one user-entered guard, and one computer-generated guard, which do not match. We used this conflict to discover what the subjects did when determining the validity of values with guards that were in conflict with each other. This spreadsheet also offered more opportunity for the propagation of guards than any of the other spreadsheet problems.

	A	B	C	D	E	F	G	H
1	City Size	Rank	Min Sales	Max Sales				
2	Large	1	90	400				
3	Large	2	110	550				
4	Large	3	120	680				
5	Medium	1	30	120				
6	Medium	2	45	150				
7	Medium	3	50	210				
8	Small	1	2	15				
9	Small	2	8	25				
10	Small	3	11	37				
11								
12								
13		Sales	Rank	City Size	Total Sales	Bonus	Salary	Adjusted Salary
14		AS01	3	M	↓ 0-1000	↓ 0-200 = (20% * E14)	↓ 20,000-100,000	↓ 20,000-115,000 = F14 + G14
15		AS02	1	L	↓ 0-1000	↓ 0-200 = (20% * E15)	↓ 20,000-100,000	↓ 20,000-115,000 = F15 + G15

Figure 13: Sales Spreadsheet. The subjects used the table in the upper left hand corner to determine the guard for the cells of the total sales column.

Wait Time: The Wait Time Spreadsheet, Figure 14, describes the average wait time for customers calling different departments within a company. The description of this problem emphasized the need for all departments to answer the President's phone call within 0-5 seconds, this translates into a row guard of 0 to 5. This spreadsheet covered two of the set relationships that the other spreadsheets did not cover.

	A	B	C	D
1		Sales	Customer Service	avg. wait time
2	Non-Member	↓ 3-90	↓ 15-80	9-85 = (B2+C2)/2
3	Member	↓ 3-90	↓ 15-80	9-85 = (B3+C3)/2
4	President	↓ 3-90	↓ 15-80	9-85 = (B4+C4)/2
5	avg wait time	↓ 3-90 =SUM(B2:B4)/3	↓ 15-80 =SUM(C2:C4)/3	9-85 = (B5+C5)/2

Figure 14: Wait Time Spreadsheet. Subjects were told that the amount of time the President was allowed to wait was between 0 and 5 seconds, they were expected to place this guard on the President's row.

Conference: The final spreadsheet, Conference Spreadsheet, Figure 15, calculates the cost of attending tutorials at a conference. This spreadsheet, unlike the other spreadsheets, was based upon a real spreadsheet, with the names

changed. The description included the fact that the cost of the tutorials was different depending on the type of registration, a student, or normal attendee. Another unique feature of this spreadsheet is that each guard is classified as being “hard”, which is not true for any other spreadsheet problem (see Table 2.3).

	A	B	C	D	E
1		Tutorial 1	Tutorial 2	Tutorial 3	Total
2	Sue	0,130,145	0,130,145	0,130,145	0, 130, 145, 260, 275, 295, 390, 405 ...
					=(B2+C2+D2)
3	John	0,130,145	0,130,145	0,130,145	0, 130, 145, 260, 275, 295, 390, 405 ...
					=(B3+C3+D3)

Figure 15: Conference Spreadsheet. Each column guard included the price for students and non-students to attend a tutorial at a conference. Since Sue was a student the subjects were told to put a guard on her row to allow for only the student price.

Spreadsheet	Column guard	Row guard	Guards- set classification
Grades	Hard	External	Subset
Sales	Exception	Exception	Subset
Conference	Hard	Hard	Subset
Wait Time	External	External	Disjoint and intersection

Table 2.3: Spreadsheet classifications. This table shows each spreadsheet and the classification scheme of each guard. The first column shows the spreadsheet, the second columns show the guard purpose of the guard that existed on the spreadsheet when the subjects first received it, and the third column shows the guard purpose of the guard the subjects were asked to add to the spreadsheet. The fourth column shows the kind of set formed in the user-user guard conflicts.

3. Results

3.1 Results of Question 1:

Do users regard having many-to-many relationships among guards and cells as being valid and useful?

This question can be broken down to two smaller questions:

- a. How do users work with many-to-one guards (many guards on one cell)?
- b. How do users work with one-to-many guards (one guard on many cells)?

3.1.1 How do users work with many-to-one guards (many guards on one cell)?

Subjects dealt directly with multiple user guards on one cell during tasks 1 and 5. Task 1 required subjects to place a guard on a row of a grid that already had column guards (the column guards were already on the spreadsheet to insure the subjects had to deal with an interaction of two guards). Task 5 required subjects to deal directly with a conflict between two user guards (the conflict was created by the examiner). The list of tasks can be found in Chapter 2.

Although at the outset of working with the spreadsheets subjects had differing attitudes about the validity of multiple user-entered guards on the same cell, by the time they were finished with their tasks, four of the five came to regard multiple user-entered guards on one cell as being a situation that required some kind of fixing (by the user). The remaining subject, however, had quite a different outlook.

Subjects 1 and 4 were the most obvious in their opinions that multiple user-entered guards on one cell should not be allowed to remain: both subjects removed the guards they decided were "extra" ones right away. These two subjects immediately removed the extra guards during both Task 1 and Task 5 of the spreadsheets.

S1 (Wait Time, task5): "I wouldn't want to make the president unhappy, so I would probably just go ahead and get rid of [the guards already here]."

S1 (Grades, task5): "I would find myself going in and crossing out the 0-100."

S4 (Grades, task1): "It seems like since Sam's a special case you could just change the range of his guards from 0-100 to 70-100, say at or above 70 points. So just change the 0 to [a] 70 for his homework."

S4 (Conference, task1): "... the values should only be 0 or 130, just take the 145 value off the guard."

During the interview at the end of the tasks Subject 1 was asked, as a follow-up to the fact that she had removed a guard, whether having two different user guards on one cell was wrong. She responded that it was wrong.

Subjects 2 and 5 were somewhat more tolerant of multiple user-entered guards on one cell. Subject 2's behavior was like subjects 1 and 4 while completing Task 1; she did not think that two different guards on one cell were valid. But, during Task 5 her opinion wavered. She appeared to be less confident about removing the extra guards on the cells. She expressed this discomfort during the last problem:

S2 (Conference, task5): "It would be better if it was just the 0 and the 130, if they deleted [the other guard] or erased it..."

Subject 2 was clearly not confident about removing the extra guards herself, since someone else had placed them. Subject 5 did not lack this confidence, but did act in similar ways to Subject 2's actions of leaving multiple guards on cells. Subject 5, like Subject 2, was more tolerant of multiple user-entered guards on one cell. In fact, during the Sales Spreadsheet she left more than one guard on a cell explaining that she was using both guards on the cell. During the interview at the end of the tasks the examiner asked her about using the two guards on the cell, and whether she would use the 50-210 over the 0-1000 (because she had seemingly been basing decisions on values on the more constraining guard, see Figure 16):

S5 (Sales, post session): "yeah, because that is what the whole thing is focusing on. So, I guess this [0-1000 guard] is just to make sure there is no huge mistake. To stay focused I would use [50-210]."

A	B	C	D	E	F	G	H
City Size	Rank	Min Sales	Max Sales				
2 Large	1	60	490				
3 Large	2	110	550				
4 Large	3	120	880				
5 Medium	1	30	120				
6 Medium	2	45	190				
7 Medium	3	50	210				
8 Small	1	2	15				
9 Small	2	8	25				
10 Small	3	11	37				

Sales Person	Rank	City Size	Total Sales	Bonus	Salary	Adjusted Salary
AS01			0 - 1000	0-200	20,000-100,000	23,000-115,000
	3		50-210 (215)	$= (20\% * E14)$		19,000 $= F14 + G14$
AS02			0 - 1000	0-200	20,000-100,000	23,000-115,000
	1	L	50-210 (9)	$= (20\% * E15)$		25,000 $= F15 + G15$

Figure 16: The Sales Spreadsheet for subject 5. Notice in the cell we have circled here, although she had left two guards on the cell, 0-1000 and 50-210, she used the range 50-210 in determining whether the system would circle the value 215 (circled in red).

Although she sometimes found a way to reason with all the guards, this was not always her preference:

Examiner: "Are you comfortable giving the spreadsheet to the customer?"

S5 (Wait Time, task5): "...They wouldn't want something that isn't matching."

She also stated:

S5 (Grades, task5): "... you shouldn't use both [guards] at the same time, because that just doesn't work. ... You would somehow let the computer know which guard to use."

Additionally, like Subject 2, Subject 5 would sometimes cross off the guard that she did not want. Subject 5 started by crossing off the guard she did not

want in both the Grades and Conference Spreadsheets; during the Grades Spreadsheet she said:

S5 (Grades, task1): "I would cross out the 0 and make it so it's between 70 and 100..."

During the Wait Time Spreadsheet she did not just remove the unwanted guard, instead she appeared confused and separate the spreadsheet cell (so the guard on the cell, and location for the cell's value were 2 individual cells); hence the guard that she added was not interacting with the other guard for that cell, this can be seen in Figure 17.

	A	B	C	D
1		Sales	Customer Service	avg. wait time
2	Non-Member	12-40	15-30	9-85
		(0)		$=(B2-C2)/2$
3	Member	13-80	15-80	9-85
		(2)		$=(B3-C3)/2$
4	President	13-80 10-5	15-80 10-5	9-85
		13-80 10-5 (6) 1	15-80 10-5 0	9-85 3
				$=(B4+C4)/2$
5	avg wait time	13-80	15-80	9-85
		(2.8)		
		$=SUM(B2:B4)/3$	$=SUM(C2:C4)/3$	$=(B5+C5)/2$

Figure 17: The Wait Time Spreadsheet for Subject 5. In this spreadsheet she split the cell we have circled here (and all others in this row) into two cells, ignoring the guards on the top part of the cell.

Like Subject 5 did, subjects 1, 4, and 2 also questioned the validity of having two user-entered guards on one cell. As Subject 1 put it, "How can it have two guards on it?"

However, unlike the other subjects, Subject 3 did not indicate any difficulties with two user-entered guards being on one cell. Rather, she saw them as working cooperatively together. As she put it while working on the spreadsheet in Figure 18:

S3 (Grades, task1): "Here is her other guard, more of a filter I guess ... It is like an additional guard on her."

	A	B	C	D	E
1		Homework	Midterm	Final	Average
2	Sam	72	58	110	65
3	Jenny	72	72	83	

Handwritten notes and formulas in the spreadsheet:

- Cell B2: 72
- Cell C2: 58
- Cell D2: 110
- Cell E2: 65
- Formula in E2: $=B2+C2+D2/3$
- Formula in E3: $=B3+C3+D3/3$

Figure 18: Subject 3's Grades Spreadsheet. As Subject 3 made decisions during this Grades Spreadsheet she used both the existing guard (0-100) and the guard she placed (70-100) when making decisions about values.

3.1.2 How do users work with one-to-many guards (one guard on many cells)?

Unlike the range of views on the many-to-one relationships, subjects consistently choose to make use of the ability to work with one-to-many relationships. All subjects made decisions about how guards applied row by row. They could have instead made such decisions one cell at a time, but none of them did. (We did not give them tasks conducive to decisions column by column.) This suggests that users indeed wanted to reason in groups where repeated patterns of relationships existed, rather than one cell at a time. For example:

S4 (Conference, Task 5): "I'll just take off the 145 guard for [the row labeled] Sue."

S1 (Grades, post session): "I would find myself crossing the 0-100 out."

Examiner: "For the whole row?"

S1: "Yes."

3.1.3 Conclusion of results for Question 1

Summarizing the above results, during the first task subjects encountered a many-to-one relationship (see Table 3.1). They again saw a many-to-one

relationship during Task 5. How they dealt with it in both cases helped us answer the first sub-question of research question one. Our subjects handled many-to-one relationships in three ways. One of these was to remove the conflict by immediately deleting the unwanted guard. A second way was to leave multiple user guards but “ignore” one when reasoning. The third way these conflicts were handled was by allowing any number of guards on a cell and reasoning with all guards. Regarding the second sub-question of Research Question 1 we found a more uniform result: all subjects choose to work with the one-to-many relationship. We weren’t able to conclude that they would create new guards on to grids “in bulk” because we taught them to do this. However, we do know from our results that subjects chose to work with existing guards in bulk, and they did not have problems with this concept. The influence the results had on the design can be found in Chapter 4.

Subject	Task 1	Task 5
1	Yes	Yes
2	Yes	No – but did not like multiple guards
3	No	No
4	Yes	Yes
5	Usually	No – but did not like multiple guards

Table 3.1: Answers: Did subjects remove the extra user guard when one existed?

3.2 Results of Question 2:

How should many-to-many user guards propagate?

Recall that three of the subjects allowed multiple user-entered guards to exist on a cell for at least some period of time. Even the subjects who choose to immediately delete “extra” guards in Task 1 were faced with them in Task 5, because Task 5 asked them what to do with a spreadsheet containing two different user guards already on one cell. Multiple user-entered guards require the system to propagate the implications of these guards.

This question can also be broken down into two sub-questions:

- a. Which guards “win”?
- b. How do guards propagate?

3.2.1 Which Guards “win”?

Given the findings of research question 1, it is not surprising that Subjects 1 and 4 always immediately selected a guard that should “win” (and thus propagate forward – as will be seen in the next section, only one guard can propagate forward for any cell), and deleted the other guard. When faced with the propagation question, Subject 2 did the same:

S2 (Wait Time, Task 1): “I want to change the president’s time to ... 0-5 seconds for each of them because he’s different.”

S4 (Grades, Task 1): “It seems like since Sam’s a special case you could just change the range of his guards from 0-100 to 70-100.”

Instead of deleting the extra user-entered guard Subject 5 decided that she would retain all user-entered guards, and embarked on a conflict-by-conflict precedence strategy, selecting which guard to use wherever a cell had multiple non-matching user-entered guards:

S5 (Grades, Task 5): “Maybe you can enter Sam’s name and it will forget about [the 0-100] guard, and remember only something about the 70-100 guard. Or you can [specify] Guard 1 and Guard 2, and say use Guard 1 or Guard 2 on this person’s name. And then you have the guards there available, and you just type in 1 or 2.”

Subject 3’s solution was also precedence based, but guard by guard rather than conflict by conflict. She was a little unclear about the meaning of a computer guard, and reinvented it to mean that a computer guard was one that had priority. During the Wait Time Spreadsheet (which required users to make sure the company’s president did not have to wait long for service) she added a guard to one of the president’s cells and said:

S3 (Wait Time, Task 1): "I'm going to put a little computer guard on here. I'm going to use the computer guard because it's the president and you don't want it to fail."

Her wording suggests that by making it a computer guard, the guard became more important than the other kind of guards it might conflict with later in the row.

Although subjects did not agree with one another on strategy, each remained consistent with his or her own strategy. That is, they built up a method of how to handle multiple guards and once it was developed, they consistently used the same method on the remaining spreadsheets and tasks, and ultimately expressed confidence in the choices they made.

3.2.2 How do guards propagate?

Although we had taught the basic rules for propagation during the tutorial, these rules did not include two user-entered guards. We were interested in observing how users thought multiple user-entered guards should propagate. During Task 4, subjects were asked to assume the role of the system and propagate the guards, filling in any missing computer guards. We were particularly interested in observing how the subjects choose to propagate guards when there were guard conflicts. We observed that the subjects did not exhibit any coherent or consistent propagation strategy in the presence of (user-user) guard conflicts. For example, Subject 2 did not actually use the formula of the cell she was propagating in determining the propagated guard:

S2 (Wait Time, Task 2): "I just looked at my lowest number...so I would take the lowest of the first row/column thing, which would be zero, and then the higher of the second one, so it would be 0-90. I used the 90 because it was the higher number".

3.3 Results of Question 3:

What constitutes a conflict?

Recall that our early prototype of guards handled only one-to-one relationships; in this prototype any two guards that did not exactly match were considered to be in conflict. We wanted to explore both whether this rule should still hold in the presence of many-to-many relationships, and the basis subjects used in deciding which guards were in conflict. For basis, we used the classification schemes covered in Chapter 2. The rest of this section is broken down into two sections:

- a. How the subjects defined guard conflicts.
- b. How the subjects dealt with value violations.

3.3.1 How the subjects defined guard conflicts

3.3.1.1 Guard purpose

Our work in devising spreadsheet problems where the guards had a variety of purposes did not pay off. We did not find any evidence of reasoning patterns based upon a guard's purpose. It is possible that subjects based their decisions upon a guard's purpose, but they did not mention it during thinking aloud or otherwise give any hint in their reasoning patterns that they were classifying guards as "validity guards" versus "query guards" or other similar purpose-based classification schemes.

3.3.1.2 Set reasoning

On the other hand, as Table 3.2, indicates, set-based reasoning was extremely common in reasoning about guard conflicts, primarily (but not always) using intersection. In other words, for some subjects, guards did not conflict if they had a non-empty intersection.

However, Subjects 2 and 5 did not rely heavily on intersection. Although Subject 2 made decisions based on exact match when computer and user guards were not in agreement, she behaved differently for user-user guard conflicts. In this case she used the union of the two guards to guard the cell (i.e., a value must

satisfy at least one of the guards), in essence defining guard conflicts out of existence.

Subject 5 showed a different strategy, one that we had not anticipated in advance. In her view, the computer guard was always right, and any other guard on the cell should then be ignored. We call this strategy “the all-knowing computer.” This is problematic, because a computer-user guard conflict is often due to a formula error, in which case the user guard—not the computer guard—is the correct one.

Subject	User-User	User-Computer
1	N/A	Intersection
2	Union	Exact match
3	Intersection	Intersection
4	N/A	Intersection
5	Intersection	All-knowing computer

Table 3.2: Subjects’ set-based reasoning or lack thereof. The User-User column shows set reasoning subjects used for user-user guard conflicts. (N/A indicates that the user eliminated the conflict.) The User-Computer column shows the reasoning used for user-computer guard conflicts.

3.3.2 *How the subjects dealt with value violations*

We choose to have subjects identify value violations by circling cell values not satisfying the relevant guards because doing so required them to think deeply about the implications of the guards and guard conflicts.

Subjects 1, 3, and 4 (and to some degree Subject 5) all reasoned about value violations in the same way: if the value fell outside any of the guards they circled it. This is consistent with the intersection-based reasoning of Table 3.2 in that to avoid a value violation, a value had to fall in the intersection of all the guards on a cell. For example:

S3 (Wait Time, Task 3): “[The cell value] 12 would be wrong because it would go through this first filter [a user guard of 3-90], but it would not go through [the computer guard of 0-5], so the computer would get it there.”

However, when faced with determining value violations, Subject 2 changed her reasoning technique from the union-based reasoning she had used about user-user guards. Instead, she decided it did not make sense to reason about the correctness of a value within a cell in which the two guards did not match exactly. An example of this occurred when she noticed the guards were different on a cell where she was deciding about a value:

S2 (Sales, post session): "Fine in one, but not in the other. I would change [the range] because [the value is] fine in one and not in the other."

In determining what constitutes a guard conflict we found no one way that subjects agreed upon. The same was true for determining if the value of a particular cell is in conflict. The effects of this on our design will be discussed in the next chapter.

4. Applying the Results by Study Question

The previous section answered the research questions from the study; this section explains how the results of the study affected our design. Like the last section, this section is broken down by research question.

4.1 Do users regard having many-to-many relationships among guards and cells as being valid and useful?

Recall that we examined the answer to this question from the perspective of one-to-many relationships and many-to-one relationships. Recall also, the subjects' lack of consistency within their own problem-solving and their lack of agreement with each other about whether multiple user-entered guards per cell were valid, given that they uniformly demonstrated that working with a single guard for multiple cells was useful. The fact that subjects were not entirely consistent about the validity of multiple user-entered guards per cell suggests that the right way to reason about many-to-one was not obvious to them. (In fact, it is possible that there is no single "right way" to reason about multiple user-entered guards per cell, but even if not, there at least needs to be a default way for the system to reason.)

It might at first seem tempting to conclude from these results that the system should support the one-to-many relationships but not the many-to-one relationships (and hence not many-to-many relationships). However, without severely restricting the way users can apply guards in spreadsheets, this solution is not possible. To restrict the user, the system could only allow the user to place a row or column guard, but not both on one grid. (In Figure 19, this would mean a user would not be able to place a guard on Sam's row once a column guard had been placed on "Midterm".) So, in order to support the one-to-many relationships, it is necessary to also support the many-to-one relationships that arise at row/column intersections. An example of the one-to-many and many-to-one relationships as we have implemented them is shown in Figure 19.

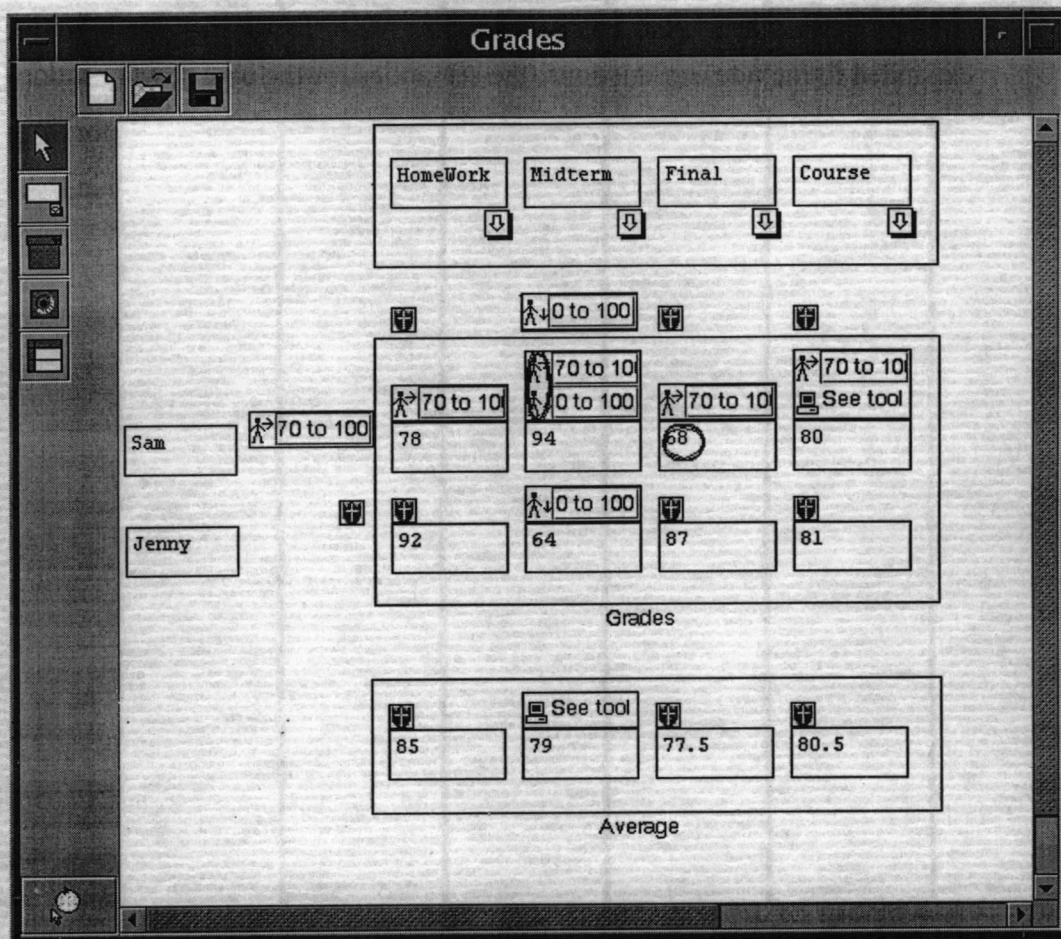


Figure 19: Grades with Guards. Guards have been placed on the "Midterm" column, and on "Sam's" row. The stick figure now has an arrow next to it indicating where the guard came from.

The users' lack of consistency and agreement suggested to us the need for a tightly integrated explanation system to clarify any reasoning the system employs. As a result, we have decided that all reasoning done by the system will be accompanied by a visible explanation. For example, in our prototype, if a user moves the mouse over a guard conflict oval, the system displays a one-sentence message explaining that "all guards for this cell must match." Figure 20 shows an example of the explanation the user receives for a guard conflict. The explanations are aimed at supporting Design Constraint 3: that users should feel they understand the system's reasoning. The explanation system has since been

expanded to include explanations (the semantics for the object, what action can be taken regarding this object, and the reason for this action – reward) for all parts of the system whose meanings are not otherwise obvious, as explained in [Wilson et al. 2002].

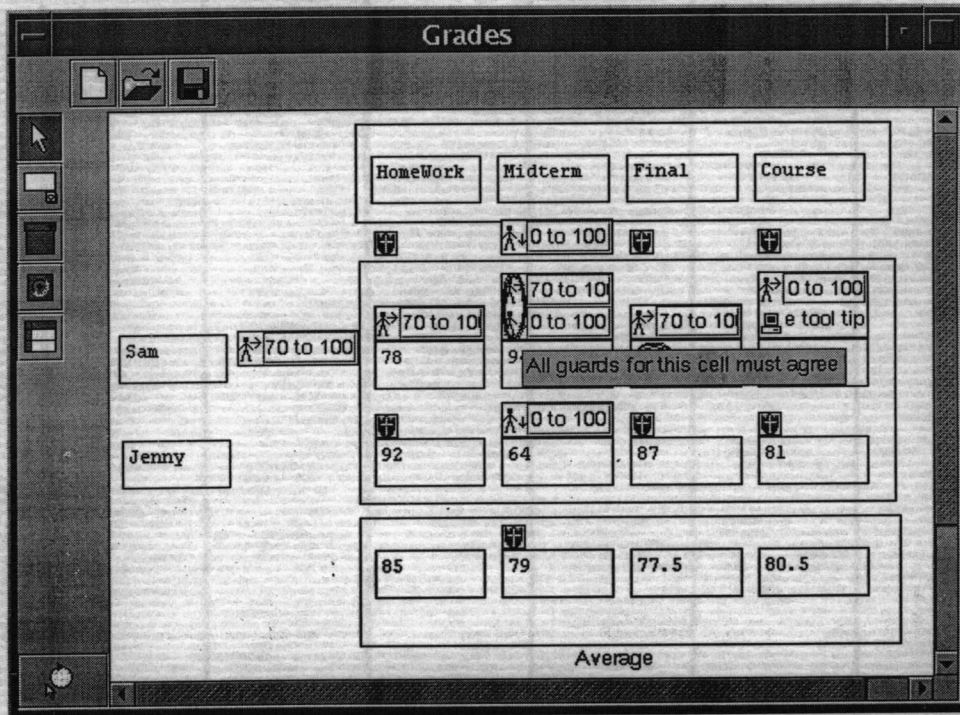


Figure 20: Guard conflict explanation. The explanation the user gets when they move their mouse over a guard conflict. This message is the same for a user-user conflict, or user-computer conflict.

4.1.1 Implementation

As stated previously, prior to many-to-many relationships, we supported only one user guard per cell. In order to support many-to-many relationships, we had to make some changes as to how the system reasoned about guards. The first of these was a new internal structure to support multiple guards per cell. Figures 21 and 22 show the old and new designs. One significant change to the design was to allow an unlimited number of guards for each cell. The change, which can be seen in Figure 22, was made by allowing a list of guards for each cell, and by

extending the guard (assertion) class to include all types of guards a cell can have. We opted to add new subclasses under the original assertion class to better keep track of an guard's source: user, computer, row, or column.

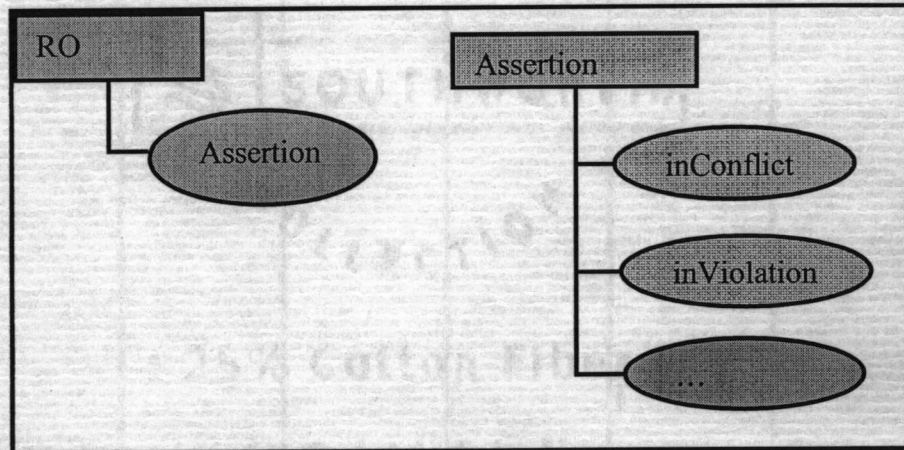


Figure 21: Previous internal design. In the old design, an RO (i.e. a cell) could have only one assertion (guard). A square box represents classes, and the ovals represent the “has-a” relationship.

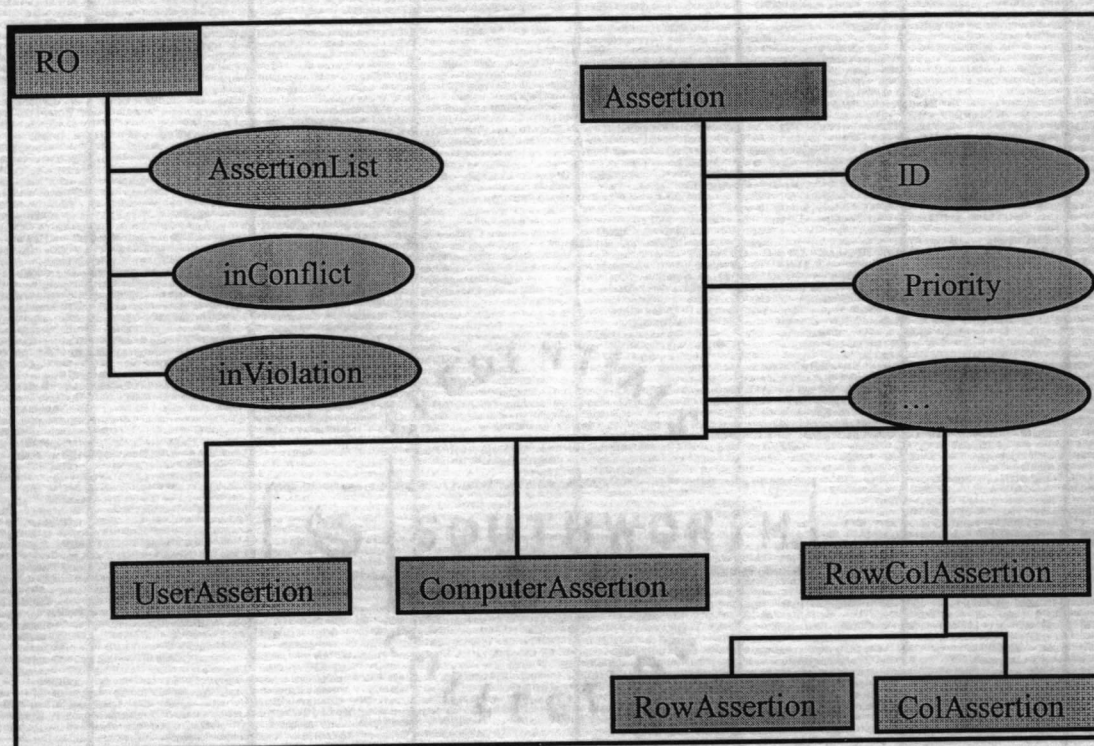


Figure 22: New internal design. In the new design an RO can have a list of guards. Additionally, some of the class variables have moved from being associated with a guard to being part of the RO class.

When scaling up guards, there were two specific Forms/3 attributes that we wanted to preserve. One attribute was conserving screen real-estate and minimizing user memorization. Since guards cannot fit into the amount of space between grid cells (see Figure 2 in Chapter 1), we needed to balance the amount of information to show (taking up screen real-estate) versus requiring the user to memorize the guards. An undesirable solution would be to force users to view guards only using a mouse-over because although it conserves screen real-estate, this solution requires users to memorize guards. Another undesirable solution that minimized screen real-estate involved leaving guards showing at the start of a row, instead of on individual cells. This solution would have still required memorization, especially for large grids.

The implemented approach was to put the guard on every cell. Although this approach does not conserve screen real-estate, we controlled the amount of

space used by allowing the user to hide guards (see Figure 23). Hiding the guards conserves screen real-estate by repositioning the cells within the grid.

The algorithm for determining the locations of a cell is based on using the maximum number of guards for the cell's row and the cell above it. In other words, to determine the location of a cell in row five, use the y position of cell in row four and the maximum number of guards showing on any cell in row five. Since the location of any cell is based on the cell above it, the layout for the cells within a grid must be done row-by-row.

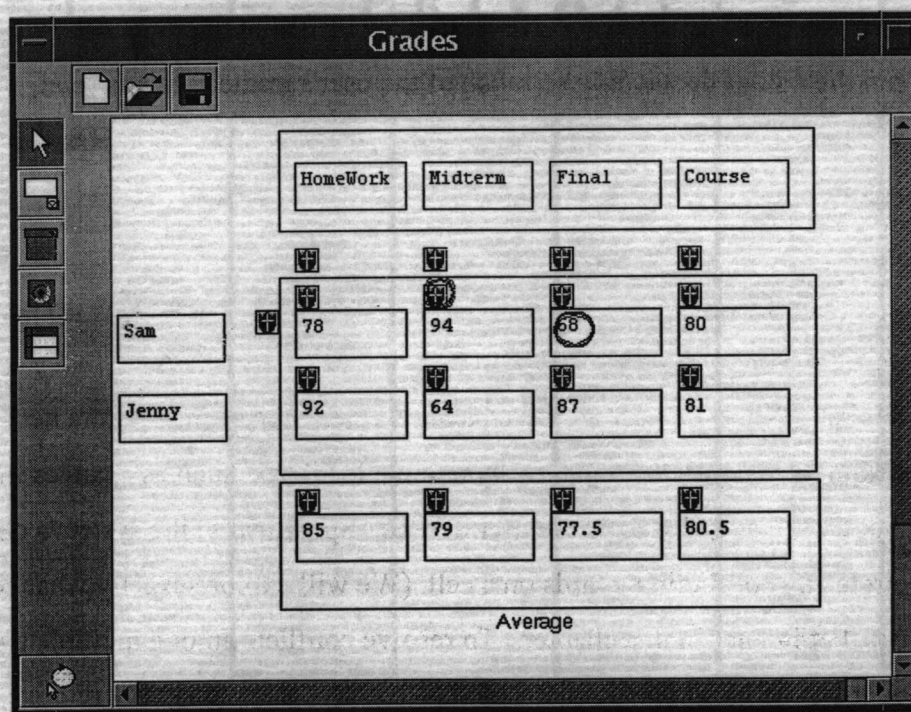


Figure 23: Grades Spreadsheet with guards closed. When the guards are closed the cells adjust to save screen real-estate.

4.2 How should multiple guards propagate?

4.2.1 Which guards "win"?

Recall from Chapter 3, the subjects demonstrated a variety of propagation decisions that were all reasonable. Thus, the approach to propagation needed to support such differences.

One way to support these differences might have been to require the users to select which guard "won" each time a new propagation was needed with competing guards. However, if we had proceeded in this direction, we would have run the risk of demanding so much of the user's immediate attention, using guards would become non-productive, violating Design Constraint 4. On the other hand, if the system made all the decisions for the users, some of the decisions would be wrong, because subjects did not all use the same strategies.

Finding the balance between requiring users to make the decisions versus making decisions for them to save time is, in our view, critical to the success of this research. The way we have balanced these competing factors here is to use default decisions accompanied by passive feedback, such as changes in markings that can be attended to as the user desires. Specifically, the system's default is to circle any conflicting guards on a cell. (We will explore exactly what constitutes a conflict in later in this chapter.) To resolve conflicts among multiple user-entered guards, one option we implemented is to allow users to simply remove a user-entered guard from any cell or cells, which is the way Subjects 1, 2, and 4 demonstrated.

To support the precedence-oriented view, we also decided to support another, more sophisticated option, namely that users can define precedence relationships among guards (this can be seen in Figure 24). Given such precedence relationships, the system uses only the guard with the highest precedence and ignores the other user-entered guards. Users can define precedence relationships at either the cell or row/column level. Figure 24 shows the way users can set precedence: by selecting to set the precedence for a

particular cell they are able to apply the precedence to this cell only or to the row/column level (effectively setting the precedence of that specific guard for every cell in the row/column). This is a gentle slope approach: users can simply delete extraneous guards if they choose, but can establish precedence for more subtle control. This relates to Design Constraint 3 (as well as to the CD termed Abstraction Gradient), because users are not forced to grapple with guard precedence unless they prefer to.

Note that these devices are only available for user-entered guards. The computer-generated guards produced by propagating user-entered guards through formulas cannot be overridden or ignored, and always are considered as having equal weight to the highest-precedence user guards. This can be seen in Figure 25, no changes can be made to the precedence of the computer guard.

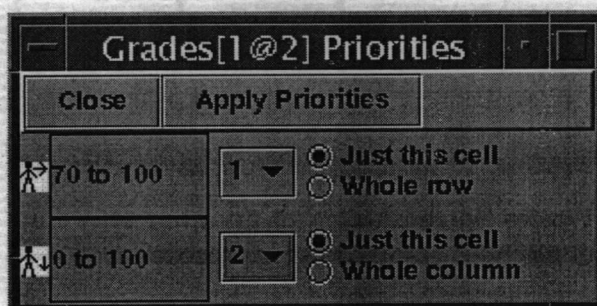


Figure 24: Guard Priorities. The users can set priorities to resolve guard conflicts. These priorities can apply to the whole row or column, or just the selected cell.

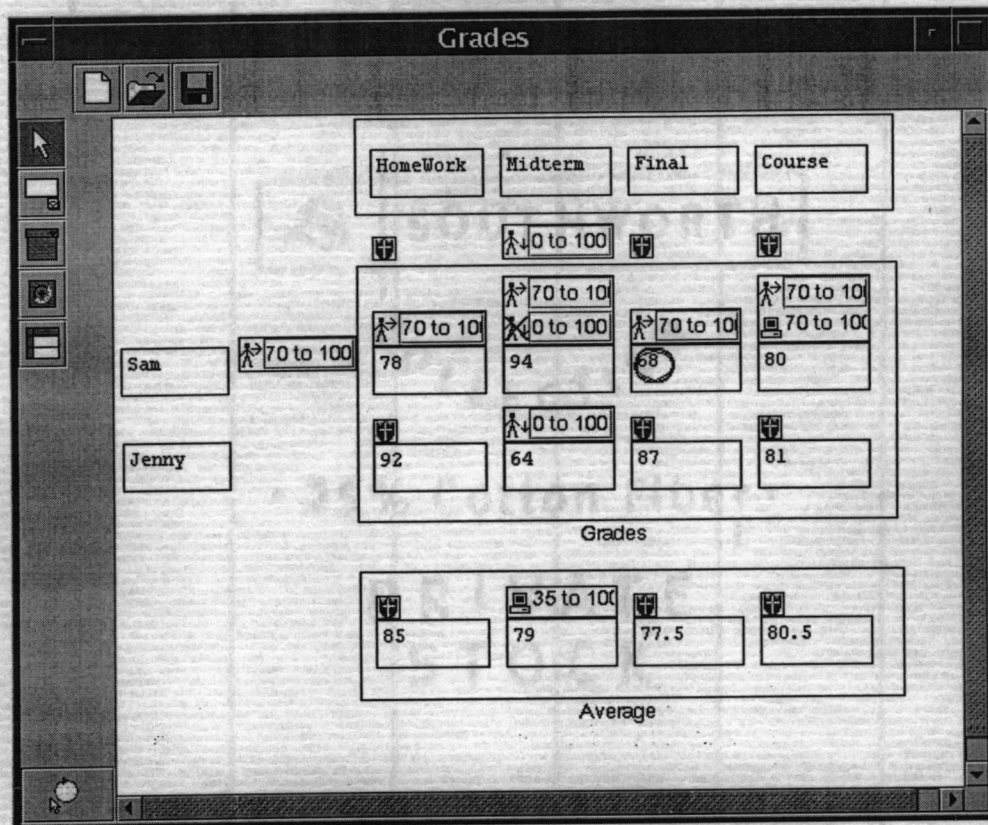


Figure 25: Grades Spreadsheet with priorities set. Shows the spreadsheet after a guard conflict has been resolved using precedence.

4.2.2 Implementation

Support for priorities required a few changes. Each guard knows its priority. The user guard with the highest priority becomes the effective guard for that cell. A computer guard's priority is always equal to the highest priority of the user guards. In other words, a computer guard can never have higher priority than all the user guards, which means a user-computer conflict cannot be resolved by rearranging the priorities of a cell with a user-computer conflict.

4.2.3 How the subjects propagated the guards

As a result of the inconsistent decisions subjects made when propagating guards during task 4, we decided not to propagate conflicting guards. This

decision was largely motivated by Design Constraint 3: since there was no clear propagation scheme that either the users or we were able to devise of what to propagate, we chose not to propagate at all rather than to devise some complicated scheme that might have been too confusing for end users. Figure 19 shows a picture of the resulting computer guard when a contributing cell has a user-user guard conflict (this also holds true for user-computer conflicts).

4.3 What constitutes a conflict?

Recall this question looked at set-based reasoning for guard conflicts – user-user, and user-computer – (summarized in Table 3.2), and it also looked at how subjects dealt with value violations. The outcome of these results might be expected to be that we took the way the majority of subjects reasoned, and incorporated it into our design. However, we could not take this approach for reasoning about conflicts, because the reasoning mechanisms most subjects showed for reasoning about conflicts allowed for more bugs to go unnoticed. Resolving the conflict, without the users input, could result in (incorrect) removal of conflicts tied with formula errors. We have already explained the problem with the "all-knowing computer" in Chapter 3. Using intersection would also incorrectly allow errors to slip through the system unnoticed, eroding the value of the guards. For example, referring again to Figure 3 in Chapter 1, using intersection-based reasoning would mean that no guard conflict would be shown on the Celsius cell. The other alternative, union-based reasoning, would never result in guard conflicts, and would accept even more erroneous values than intersection-based reasoning.

Thus, to keep the errors out, it is necessary for guards to exactly match to be considered to be free of guard conflicts, provided that all guards are at equal precedence levels. Our current prototype does this, as Figure 19 shows. However, recall that users can control this behavior: they can delete guards that do not apply

to particular cells, or can establish precedence hierarchies to cause certain guards to "win" over other guards, if this level of sophistication is desired.

Even in the presence of guard conflicts, it is necessary for the system to reason about whether value violations exist. The approach follows intersection reasoning here, as did most of the subjects, in essence saying that a value must satisfy all the (top-precedence) guards to be free of violation.

As the other issues also showed, subjects did not agree on their reasoning, and thus might not understand the system's reasoning choices without explicit support. As we pointed out before, part of our design includes an explanation system in the form of consistent, one-sentence explanations for each reasoning outcome. For example, if the user mouses over a value violation, the system displays the message "value must satisfy all guards" . Key to this strategy is the fact that a reasoning explanation can be given in just one sentence (with the aforementioned expanded explanation system the one sentence explanation is longer to accommodate all aspects of the explanations, as shown in Figure 26), which is helped by following Design Constraint 2 (consistency) to avoid special-case reasoning.

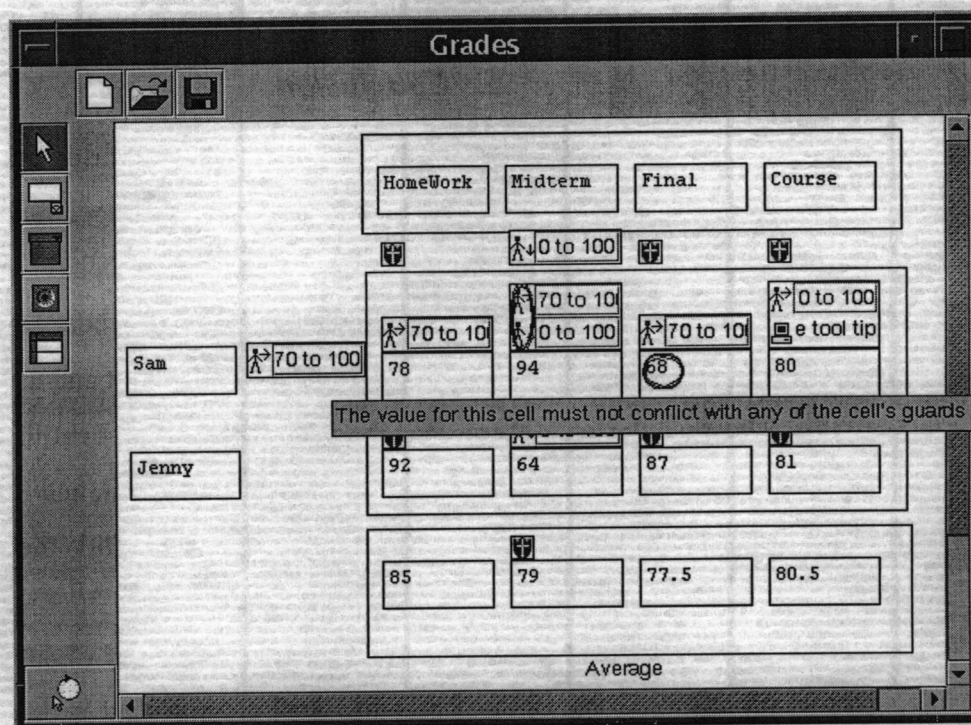


Figure 26: Value violation explanation. This shows the message the users see as they move their mouse over a value violation.

5. Conclusion

In this work, we have considered how a system should reason behind the scenes about many-to-many relationships between requirement specifications (guards) and spreadsheet cells. Although the approach is about behind-the-scenes reasoning, the reasoning is not really very far behind the scenes, because end users need to understand how the reasoning works if they are to trust it and use it effectively. Thus, we designed the approach following a human-centric procedure. First, we drew from Cognitive Dimensions, Attention Investment, and literature about on-line trust, to devise a set of five design constraints to guide the development of our approach.

Second, we turned to the users themselves for additional insights, via a small think-aloud study. The subjects demonstrated that the troublesome side of the many-to-many relationships lay on the many-to-one side (many guards to one cell). They exhibited a variety of reasoning approaches, some of which were reasonable and some of which were faulty. We were able to employ some of their reasoning mechanisms in our approach, as is shown in the following issue-by-issue summary:

Many-to-many relationships: Subjects were inconsistent in their attitudes and reasoning about multiple guards on one cell (many-to-one), but all chose to work with one guard over multiple cells (one-to-many). Because of their difficulties with many-to-one relationships—which are required to support the one-to-many relationships—we added an explanation-based approach to the design for all reasoning about guards.

Propagation: Subjects demonstrated a variety of propagation mechanisms, all of which were reasonable. The impact on our design was to support all the propagation mechanisms they demonstrated via a gentle slope approach, except with guard conflicts (in which case no propagation at all occurs).

What is a conflict/violation: Regarding guard conflicts, subjects demonstrated several mechanisms for defining guard conflicts, many of which

were unsound. Instead of adopting unsound mechanisms, the design uses an exact match rule to define guard conflicts. Regarding value violations, subjects demonstrated intersection reasoning, which was adopted by our design.

The most important outcome was that subjects' differing approaches made clear that many-to-many relationships will require careful support to be viable for end users. We provide this support via an explanation-based approach for all aspects of the system's behind-the-scenes reasoning. Since the time of the study we have expanded the explanation-based approach beyond guards, including explanations for any object whose meaning is not immediately apparent.

Bibliography

- [Ambler 1999] A. Ambler, "The Formulate Visual Programming Language," Dr. Dobb's Journal, Aug. 1999, 21-28.
- [Ambler and Broman 1998] A. Ambler and A. Broman, "Formulate Solution to the Visual Programming Challenge," *Journal of Visual Languages and Computing* 9(2), April 1998, 171-209.
- [Belkin 2000] N. Belkin, "Helping people find what they don't know," Comm. ACM 41(8), Aug. 2000, 58-61
- [Blackwell and Green 1999] A. Blackwell, A. and T. R. Green, "Investment of attention as an analytic approach to cognitive dimensions." In T. Green, R. Abdullah & P. Brna (Eds.) *Collected Papers Wkshp. Psychology of Programming Interest Group*, 1999, 24-35.
- [Blackwell 2002] A. Blackwell, "First steps in programming: a rationale for attention investment models," *Proc. IEEE Human-Centric Computing Languages and Environments*, Arlington, VA, Sept. 2002, 2-10.
- [Boehm and Basili 2000] B. Boehm and V. Basili, "Gaining intellectual control of software development," *Computer* 33(5), May 2000, 27-33.
- [Boren and Ramey 2000] M.T. Boren and J. Ramey, "Thinking aloud: reconciling theory and practice," *IEEE Transactions on Professional Communication*, 43(3), Sept. 2000, 261-278.
- [Burnett and Erwig 2002] M. Burnett and M. Erwig. "Visually customizing inference rules about apples and oranges," *Proc. IEEE Human-Centric Computing Languages and Environments*, Arlington, VA, Sept. 2002, 140-148.
- [Burnett et al. 2001] M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang, "Forms/3: a first-order visual language to explore the boundaries of the spreadsheet paradigm," *Journal of Functional Programming*, Mar. 2001, 155-206.
- [Burnett et al. 2002a] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, "End-User Software Engineering with Assertions," Technical Report 02-60-05, Oregon State University, Sept. 2002.
- [Burnett et al. 2002b] M. Burnett, A. Sheretov, B. Ren, and G. Rothermel, "Testing homogeneous spreadsheet grids with the 'What You See Is What You Test' methodology," *IEEE Trans. Software Engineering*, May 2002, 576-594.
- [Chi et al. 1997] E. H. Chi, P. Barry, J. Riedl, and J. Konstan, "A spreadsheet approach to information visualization." *Proc. IEEE Symposium on Information Visualization '97*, Phoenix, AZ, Oct. 1997, 17-24.
- [Corritore et al. 2001] C. Corritore, B. Kracher, and S. Wiedenbeck, "Trust in the online environment," *HCI International, Vol. 1*, New Orleans, LA, Aug. 2001, 1548-1552.
- [Cypher and Smith 1995] A. Cypher, and D. C. Smith, "KidSim: End User Programming of Simulations". *ACM Conf. Human Factors in Computing Systems*, Denver, CO, May 1995, 27-34.
- [Dix et al. 1993] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human-Computer Interaction*, Prentice Hall, New York, 1993.

- [Ericsson and Simon 1984] K. Ericsson, and H. Simon, *Protocol Analysis*, MIT Press, Cambridge, MA, 1984.
- [Engebretson and Wiedenbeck 2002] A. Engebretson and S. Wiedenbeck, "Novice comprehension of programs using task-specific and non-task-specific constructs," *Proc. IEEE Human-Centric Computing Languages and Environments*, Arlington, VA, Sept. 2002, 11-18.
- [Ernst et al. 1999] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," *Int'l. Conf. Software Engineering*, Los Angeles, CA, May 1999, 213-224.
- [Gray and Fu 2001] W. Gray and W. Fu, "Ignoring perfect knowledge in-the-world for imperfect knowledge in-the-head: implications of rational analysis for interface design," *ACM Conf. Human Factors in Computing Systems*, Seattle, WA, March 2001, 112-119.
- [Green and Petre 1996] T. R. G. Green, and M. Petre, "Usability analysis of visual programming environments: a 'cognitive dimensions' framework," *J. Visual Languages and Computing* 7(2), June 1996, 131-174.
- [Hartmann et al. 2001] W. Hartmann, J. Nievergelt, R. Reichert, "Kara, finite state machines, and the case for programming as part of general education," *Proc. IEEE Symposia on Human-Centric Computing Languages and Environments*, Stresa, Italy, Sept. 2001, 135-141.
- [Igarashi et al. 1998] T. Igarashi, J. D. Mackinlay, B. Chang, and P. T. Zellweger, "Fluid Visualization of Spreadsheet Structures," *ACM Conf. Human Factors in Computing Systems*, Los Angeles, CA, Apr. 1998, 118-125.
- [Karam and Smedley 2001] M. Karam and Trevor Smedley, "A testing methodology for a dataflow based visual programming language," *Proc. Human-Centric Computing Languages and Environments*, Stresa, Italy, Sept. 2001, 280-287.
- [Krishna et al. 2001] V. Krishna, C. Cook, D. Keller, J. Cantrell, C. Wallace, M. Burnett, and G. Rothermel, "Incorporating incremental validation and impact analysis into spreadsheet maintenance: an empirical study," *IEEE Int'l. Conf. Software Maintenance*, Florence, Italy, Nov. 2001, 72-81.
- [Ko et al. 2002] A.J. Ko, M. Burnett, T. R. G. Green, K. Rothermel, and C. Cook, "Improving the Design of Visual Programming Language Experiments Using Cognitive Walkthroughs," *Journal of Visual Languages and Computing* 13(5), Oct. 2002, pp. 517-544.
- [Landay and Myers 2001] J. Landay and B. Myers, "Sketching interfaces: toward more human interface design," *Computer* 34(3), Mar. 2001, 56-64.
- [Miller and Myers 2001] R. Miller, R., and B. Myers, "Outlier finding: focusing user attention on possible errors," *ACM Symp. User Interface Software and Technology*, Orlando, FL, Nov. 2001, 81-90.
- [Nardi 1993] B. Nardi, *A Small Matter of Programming: Perspectives on End-User Computing*, MIT Press, Cambridge, MA, 1993.
- [Pane et al. 2002] J. Pane, B. Myers, and L. Miller, "Using HCI techniques to design a more usable programming system," *Proc. IEEE Human-Centric Computing Languages and Environments*, Arlington, VA, Sept. 2002, 198-206.

- [Pane et al. 2001] J.F. Pane, C.A. Ratanamahatana, and B.A. Myers, "Studying the language and structure in non-programmers' solutions to programming problems," *International Journal of Human-Computer Studies* 54(2), Feb. 2001, pp. 237-264.
- [Pane and Myers 2000] J.F. Pane and B.A. Myers, "Tabular and Textual Methods for Selecting Objects from a Group," *Proceedings of VL 2000: IEEE International Symposium on Visual Languages*, Seattle, WA, Sept. 2000, pp. 157-164.
- [Panko 1995] R. Panko, "Finding spreadsheet errors: most spreadsheet models have design flaws that may lead to long-term miscalculation," *Information Week*, May 29, 1995, 100.
- [Panko 1998] R. Panko, "What we know about spreadsheet errors," *J. End User Computing*, Spring 1998.
- [Panko 2000] R. Panko, "Spreadsheet errors: what we know, what we think we can do," *Symp. European Spreadsheet Risks Interest Group*, July 2000.
- [Rader et al. 1997] C. Rader, C. Brand, and C. Lewis, "Degrees of Comprehension: Children's Understanding of a Visual Programming Environment," *ACM Conf. Human Factors in Computing Systems*, Atlanta, GA, Mar. 1997, 351-358.
- [Reichwein et al. 1999] J. Reichwein, G. Rothermel, and M. Burnett, "Slicing spreadsheets: an integrated methodology for spreadsheet testing and debugging," *Conf. Domain-Specific Languages*, Austin, TX, Oct. 1999, 25-38.
- [Repenning 2000] A. Repenning, "AgentSheets®: an Interactive Simulation Environment with End-User Programmable Agents," *Interaction 2000*, Tokyo, Japan, 2000.
- [Repenning et al. 2000] A. Repenning, A. Ioannidou, and J. Zola, J. "AgentSheets: End-User Programmable Simulations," *Journal of Artificial Societies and Social Simulation*, 2000, 3(3).
- [Repenning and Citrin 1993] A. Repenning, and W. Citrin, "Agentsheets: Applying Grid-Based Spatial Reasoning to Human-Computer Interaction," *IEEE Workshop on Visual Languages*, Bergen, Norway, Aug. 1993, 77-82.
- [Rosenblum 1995] D. S. Rosenblum, "A Practical Approach to Programming with Assertions," *IEEE Trans. Software Engineering* 21(1), Jan. 1995, 19-31.
- [Rosson and Seals 2001] M.B. Rosson and C. Seals, "Teachers as simulation programmers: minimalist learning and reuse," *ACM Conf. Human Factors in Computing Systems*, Seattle, WA, Mar. 2001, 237-244.
- [Rothermel et al. 1998] G. Rothermel, L. Li, C. DuPuis, and M. Burnett, "What you see is what you test: a methodology for testing form-based visual programs," *Int'l. Conf. Software Engineering*, Kyoto, Japan, Apr. 19-25, 1998, 198-207.
- [Rothermel et al. 2001] G. Rothermel, L. Li, C. DuPuis, and A. Sheretov, "A methodology for testing spreadsheets," *ACM Trans. Software Engineering and Methodology*, Jan. 2001, 110-147.
- [Rothermel et al. 2000] K. Rothermel, C. Cook, M. Burnett, J. Schonfeld, T. R. G. Green, and G. Rothermel, "WYSIWYT testing in the spreadsheet paradigm: an empirical evaluation," *Int'l. Conf. Software Engineering*, Limerick, Ireland, June 2000, 230 to 239.

- [Sankar and Mandal 1993] S. Sankar, and M. Mandal, "Concurrent runtime monitoring of formally specified programs," *Computer* 26, Mar. 1993, 32-41.
- [Seals et al. 2002] C. Seals, M.B. Rosson, J. Carroll, T. Lewis, L. Colson, "Fun learning Stagecast Creator: An exercise in minimalism and collaboration," *Proc. IEEE Human-Centric Computing Languages and Environments*, Arlington, VA, Sept. 3-6, 2002, 177-186.
- [Wallace et al. 2002] C. Wallace, C. Cook, J. Summet, and M. Burnett, "Assertions in end-user software engineering: a think-aloud study," *Proc. IEEE Human-Centric Computing Languages and Environments*, Arlington, VA, Sept. 3-6, 2002, 63-65.
- [Wilson et al. 2002] A. Wilson, M. Burnett, L. Beckwith, O. Granatir, L. Casburn, C. Cook, M. Durham, and G. Rothermel, "Harnessing curiosity to increase correctness in end-user programming," *ACM Conf. Human Factors in Computing Systems (to appear)*.

Appendices

Appendix A: Tutorial Materials

Think-aloud introduction read to subjects before the tutorial:

"In this experiment we are interested in what you say to yourself as you perform some tasks that we give you. In order to do this we will ask you to TALK ALOUD CONSTANTLY as you work on the problems. What I mean by talk aloud is that I want you to say aloud EVERYTHING that you say to yourself silently. Just act as if you are alone in this room speaking to yourself. If you are silent for any length of time, I will remind you to keep talking aloud. It is most important that you keep talking. Do you understand what I want you to do ? Good. Before we turn to the real experiment and the tutorial, we will start with a couple of practice questions to get you used to with talking aloud. I want you to talk aloud as you do these problems. First I will ask you to add two numbers in your head."

The following is the paragraph used during think-aloud practice, just as they saw it, seeded with grammatical and spelling errors. Users were instructed to read the paragraph aloud and make any changes they felt necessary.

"Lleyton Hewitt becomes the first Australian Open top ever seed to loose in the first round of the men's single on Tuesday hence he was sensationally dumped out by journeyman Spaniard Alberto Martin, 1-6, 6-1, 6-4, 7-6. the world No.1, recovering for a bout of chicken pox, receives courtside treatment for blisters and had his thighs massaged, as he losed to a player who before this year had won just won match in four visits to the Open."

Appendix B: Subject Spreadsheets

	A	B	C	D	G
1		Homework	Midterm	Final	Average
2	Sam	70-100 70-100 70-100 70-100	70-100 70-100 70-100 70-100	70-100 70-100 70-100 70-100	70-100 70-100 70-100 70-100
		(58)	73		=(B2+C2+D2)/3
3	Jenny	70-100 70-100 70-100 70-100	70-100 70-100 70-100 70-100	70-100 70-100 70-100 70-100	70-100 70-100 70-100 70-100
		72	(105)	58	=(B3+C3+D3)/3

blue - task #1
purple - task #2
red - task #3
orange - task #5

Figure 27: Subject 1 Grades.

	A	B	C	D	E
1		Tutorial 1	Tutorial 2	Tutorial 3	Total
2	Sue	0, 130, 145 0, 130, 145 0, 130, 145 0, 130, 145	0, 130, 145 0, 130, 145 0, 130, 145 0, 130, 145	0, 130, 145 0, 130, 145 0, 130, 145 0, 130, 145	0, 130, 145, 260, 390 0, 130, 145, 260, 390, 520 0, 130, 145, 260, 390, 520, 650 0, 130, 145, 260, 390, 520, 650, 780
		130	(45)	(45)	=(B2+C2+D2)
3	John	0, 130, 145 0, 130, 145 0, 130, 145 0, 130, 145	0, 130, 145 0, 130, 145 0, 130, 145 0, 130, 145	0, 130, 145 0, 130, 145 0, 130, 145 0, 130, 145	0, 130, 145, 260, 275, 295, 390, 405 0, 130, 145, 260, 275, 295, 390, 405, 420 0, 130, 145, 260, 275, 295, 390, 405, 420, 435 0, 130, 145, 260, 275, 295, 390, 405, 420, 435, 450
		130	0	145	=(B3+C3+D3)

blue: task #1
purple: task #2 - no changes
red: task #3

Figure 28: Subject 1 Conference.

	A	B	C	D	E	F	G	H
	City Size	Rank	Min Sales	Max Sales				
1								
2	Large	1	90	400				
3	Large	2	110	550				
4	Large	3	120	680				
5	Medium	1	30	120				
6	Medium	2	45	150				
7	Medium	3	50	210				
8	Small	1	2	15				
9	Small	2	8	25				
10	Small	3	11	37				
11								
12								
13								
14	AS01	3	M	40				
15	AS02	1	L	1001				

no action for task #1
no " " " #2
task #3
often - are you sure that the spreadsheet works the way it should?

Sales Person	Rank	City Size	Total Sales	Bonus	Salary	Adjusted Salary
AS01	3	M	40	$= (20\% * E14)$	$\$20,000 - \$100,000$	$\$20,000 - \$100,000$
AS02	1	L	1001	$= (20\% * E15)$	$\$20,000 - \$100,000$	$\$20,000 - \$100,000$

100,000
120,000

Figure 29: Subject 1 Sales.

	A	B	C	D
		Sales	Customer Service	avg. wait time
1				
2	Non-Member	$\$0 - \90	$\$15 - \180	$15 - 210$
3	Member	$\$0 - \90	$\$15 - \180	$15 - 210$
4	President	$\$0 - \90	$\$15 - \180	$15 - 210$
5	avg wait time	$\$0 - \90	$\$15 - \180	$15 - 210$

blue: task #1
purple: task #2
red: task #3
what do you think the guard is here?
task 5 - gold

Added this in @ end

Figure 30: Subject 1 Wait Time.

change task #5

	A	B	C	D	G
1		Homework	Midterm	Final	Average
2	Sam	0-100 70-100 0-100 70-100 (58)	0-100 70-100 0-100 70-100 75	0-100 70-100 0-100 70-100	0-100 70-100 0-100 70-100 $=(B2+C2+D2)/3$
3	Jenny	0-100 73	0-100 50	0-100 (105)	0-100 $=(B3+C3+D3)/3$

blue - task #1
red - task #3
green - task #5

Figure 31: Subject 2 Grades.

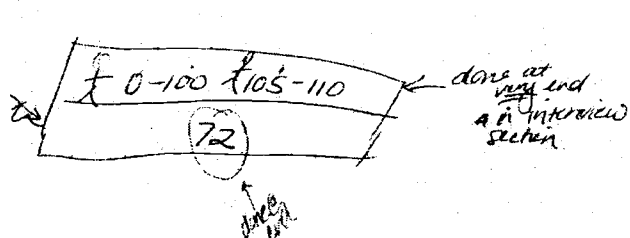
added in task #5

	A	B	C	D	E
1		Task 1	Task 2	Task 3	Total
2	Sue	0, 130, 145 0, 130, 145 0	0, 130, 145 0, 130, 145 (45)	0, 130, 145 0, 130, 145 130	0-390 0-390 275 $=(B2+C2+D2)$
3	John	0, 130, 145 0, 130, 145 145	0, 130, 145 0, 130, 145 145	0, 130, 145 0, 130, 145 (50)	0, 130, 145, 260, 275, 285, 390, 405 0, 130, 145, 260, 275, 285, 390, 405 $=(B3+C3+D3)$

blue - task #1
purple - task #2
red - task #3
green - task #5

Figure 32: Subject 2 Conference.

	A	B	C	D	G
1		Homework	Midterm	Final	Average
2	Sam	$\frac{0}{100}$ $\frac{0}{100}$ $\frac{0}{100}$ $\frac{0}{100}$	$\frac{0}{100}$ $\frac{0}{100}$ $\frac{0}{100}$ $\frac{0}{100}$	$\frac{0}{100}$ $\frac{0}{100}$ $\frac{0}{100}$ $\frac{0}{100}$	$\frac{0}{100}$ $\frac{0}{100}$ $\frac{0}{100}$ $\frac{0}{100}$ $\frac{0}{100}$
		72	58	110	65 $= (B2+C2+D2)/3$
3	Jenny	$\frac{0}{100}$	$\frac{0}{100}$	$\frac{0}{100}$	$\frac{0}{100}$
			72	83	$= (B3+C3+D3)/3$



blue - task #1
purple - task #2
red - task #3

Figure 35: Subject 3 Grades.

	A	B	C	D	E
1		Tutorial 1	Tutorial 2	Tutorial 3	Total
2	Sue	$\frac{0}{130, 145}$ $\frac{0}{130, 145}$	$\frac{0}{130, 145}$ $\frac{0}{130, 145}$	$\frac{0}{130, 145}$ $\frac{0}{130, 145}$	$\frac{0}{130, 145, 145, 145}$
		0 $\frac{0}{130, 145}$	195 $\frac{0}{130, 145}$	45 $\frac{0}{130, 145}$	$= (B2+C2+D2)$
3	John	$\frac{0}{130, 145}$	$\frac{0}{130, 145}$	$\frac{0}{130, 145}$	$\frac{0}{130, 145, 280, 275, 295, 390, 405 \dots}$
		120	145		$= (B3+C3+D3)$

blue - task #1
purple - task #2
red - task #3
green - I asked what if we have both genders on this cell?

Figure 36: Subject 3 Conference.

A	B	C	D	E	F	G	H
1	City Size	Rank	Min Sales	Max Sales			
2	Large	1	90	400			
3	Large	2	110	550			
4	Large	3	120	660			
5	Medium	1	30	120			
6	Medium	2	45	180			
7	Medium	3	50	210			
8	Small	1	2	15			
9	Small	2	8	25			
10	Small	3	11	37			
11							
12							
13							
14							
15							

Sales Person	Rank	City Size	Total Sales	Bonus	Salary	Adjusted Salary
AS01			230			
	3	M	230			
AS02			1001			
	1	L	1001			

blue - task #1
 purple - task #2
 red - task #3
 green - small mark on
 salary - in post
 SS - significant

210,000
 112,000
 120,000

Figure 37: Subject 3 Sales.

	A	B	C	D
1		Sales	Customer Service	avg. wait time
2	Non-Member	3-90	15-80	9-85
		0		$=(B2+C2)/2$
3	Member	3-90	15-80	9-85
		3		$=(B3+C3)/2$
4	President	3-90 0-5	15-80 0-5	9-85 0-5
		6	3	7.5
				$=(B4+C4)/2$
5	avg wait time	3-90 0-5	15-80 0-5	9-85 0-5
		4		0-5
		$=SUM(B2:B4)/3$	$=SUM(C2:C4)/3$	$=(B5+C5)/2$

3

5

blue - task #1
 purple - task #2
 red - task #3
 green - 1 put 2 user queries
 in the cells -
 asked what would
 happen.

Figure 38: Subject 3 Wait Time.

	A	B	C	D	E
1		Homework	Midterm	Final	Average
2	Sam	70-100 7-100	70-100 70-100	70-100 70-100	70-100 70-100
		58	72	110	$=(B2+C2+D2)/3$
3	Jenny	40-100	40-100	40-100	0-100
			59	5	$=(B3+C3+D3)/3$

Figure 39: Subject 4 Grades.

	A	B	C	D	E
1		Tutorial 1	Tutorial 2	Tutorial 3	Total
2	Sue	0, 130, 145	0, 130, 145	0, 130, 145	0, 130, 145, 280, 275, 285, 390, 405
			0	145	$=(B2+C2+D2)$
3	John	0, 130, 145	0, 130, 145	0, 130, 145	0, 130, 145, 280, 275, 285, 390, 405
			45	130	$=(B3+C3+D3)$

Figure 40: Subject 4 Conference.

	A	B	C	D	E	F	G	H
1	City Size	Rank	Min Sales	Max Sales				
2	Large	1	90	400				
3	Large	2	110	550				
4	Large	3	120	680				
5	Medium	1	30	120				
6	Medium	2	45	160				
7	Medium	3	50	210				
8	Small	1	2	15				
9	Small	2	8	25				
10	Small	3	11	37				
11								
12								
13	Sales Person	Rank	City Size	Total Sales	Bonus	Salary	Adjusted Salary	
14	AS01	3	M	150,000	25 = (20% * E14)	20,000 - 100,000	20,000 - 115,000	20,000 - 100,200
15	AS02	1	L	100,000	20 = (20% * E15)	20,000 - 100,000	20,000 - 115,000	20,000 - 100,200

110,000
112,000
120,000

Figure 41: Subject 4 Sales.

	A	B	C	D
1		Sales	Customer Service	avg. wait time
2	Non-Member	13-30	15-30	9-85
		2		=(B2+C2)/2
3	Member	13-30	15-30	9-85
		3	13	=(B3+C3)/2
4	President	13-30	15-30	9-85
		6	3	=(B4+C4)/2
5	avg wait time	13-30	15-30	9-85
		3	3	=(B5+C5)/2
		=SUM(B2:B4)/3	=SUM(C2:C4)/3	

Figure 42: Subject 4 Wait Time.

	A	B	C	D	G
1		Homework	Midterm	Final	Average
2	Sam	0-100 0-100	0-100 0-100	0-100 0-100	0-100
		(58)	72	(110)	$=(B2+C2+D2)/3$
3	Jenny	0-100	0-100	0-100	0-100
			(50)	78	$=(B3+C3+D3)/3$

Figure 43: Subject 5 Grades.

	A	B	C	D	E
1		Tutorial 1	Tutorial 2	Tutorial 3	Total
2	Sue	0, 130, 145	0, 130, 145	0, 130, 145	0, 130, 145, 260, 275, 285, 390, 405 ...
		(75)	130	(75)	$=(B2+C2+D2)$
3	John	0, 130, 145	0, 130, 145	0, 130, 145	0, 130, 145, 260, 275, 285, 390, 405 ...
			145	(75)	$=(B3+C3+D3)$

Figure 44: Subject 5 Conference.

	A	B	C	D	E	F	G	H
1	City Size	Rank	Min Sales	Max Sales				
2	Large	1	90	400				
3	Large	2	110	550				
4	Large	3	120	680				
5	Medium	1	30	120				
6	Medium	2	45	150				
7	Medium	3	50	210				
8	Small	1	2	15				
9	Small	2	8	25				
10	Small	3	11	37				
11								
12								
13	Sales Person	Rank	City Size	Total Sales	Bonus	Salary	Adjusted Salary	
14	AS01	3	M	10 - 1000 2250 (215)	0-200 = (20% * E14)	20,000-100,000	20,000-115,000 19,000 = F14 + G14	20,000 - 100,200
15	AS02	1	L	10 - 1000 1200 (1)	0-200 = (20% * E15)	20,000-100,000	20,000-115,000 25,000 = F15 + G15	20,000 - 100,200

2-10 2-15
A = ..

100,000
112,000
120,000

she would check if outside prescription + ml + tuition

Figure 45: Subject 5 Sales.

	A	B	C	D
1		Sales	Customer Service	avg. wait time
2	Non-Member	13-90 (6)	15-80	9-85 =(B2+C2)/2
3	Member	13-90 (2)	15-80	9-85 =(B3+C3)/2
4	President	13-90 205 1200 (6) 1	15-80 20-5 1200 50 1 0	9-85 1200 3 =(B4+C4)/2
5	avg wait time	13-90 2.8 =SUM(B2:B4)/3	15-80 15.0 =SUM(C2:C4)/3	9-85 9-85 = (B5+C5)/2

blue-task #1

Figure 46: Subject 5 Wait Time.

Appendix C: Quotes

Quotes not included in the body of the thesis, broken down by spreadsheet:

Wait Time:

S1 (task1): "I'm going to correct and change the values that are in here right now."

S4 (task5): "take [the extra guard] off the president and leave the 0-5 value."

Grades:

S1 (task1): "Take out the guard that is already there and replace it with the 70-100, do this for all the columns."

Conference

S1 (task1): "Just going to cross out the 145 on each column of Sue's."

S5 (task1): "I just crossed off the 145 because we're dealing with 0 and 130."