IEEE Symposia on Human-Centric Computing Languages and Environments, Arlington, VA, Sept. 2002 (to appear).

Reasoning about Many-to-Many Requirement Relationships in Spreadsheets

Laura Beckwith, Margaret Burnett, and Curtis Cook
Oregon State University, Corvallis, Oregon, 97331 USA
{beckwith, burnett, cook}@cs.orst.edu

Abstract

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." 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, and it is not clear how the system should reason and communicate about them in a way that will make sense to end users. In this paper, 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 Economics—and from the users themselves in a small think-aloud study.

1. Introduction

Although end-user programming has received a growing amount of attention, 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 [15].

We have been working on how to improve the reliability of end-user programs in general and 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 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 [3]. In our prototype, we refer to requirement specifications as *guards*. 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 [21].

In our early prototype, guards pertained to only one cell, although their implications were propagated through formulas to other cells. That is, whenever a user put a guard into a spreadsheet (a *user-entered guard*) it was propagated 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 did not match exactly. As Figure 1 shows, to communicate the fact that there is 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.

In this paper, we focus on a matter necessary for real-world spreadsheets: how to establish scalable guard mechanisms. By "scalable," we mean guard mechanisms that are viable for end users when programming spreadsheets with grids of many cells with repeated patterns of relationships, often due to shared or replicated formulas across the rows or columns. Most large spreadsheets consist of grids with such

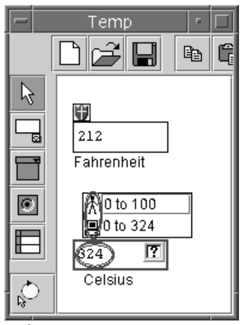


Figure 1: A Forms/3 temperature converter. Stick figure icons identify user-entered guards, and the computer icon identifies a computer-generated guard. The computer-generated guard's conclusion that the Celsius value ranges from 0 to 324 degrees provides a clue that there is an error in Celsius's formula.

repeated patterns of relationships.

Scaling up guards to support such grids presents a difficult challenge regarding finding a reasoning mechanism that will be understandable and sensible to end users. 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 2 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.

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 2, 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?

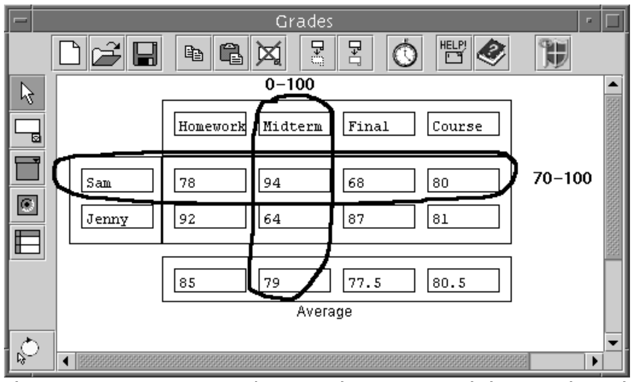


Figure 2: A Forms/3 grades spreadsheet, hand-annotated to illustrate issues arising from many-to-many relationships.

In this paper, we consider from a human-centric perspective, issues such as the one above, that fundamentally affect how the system should reason about these many-to-many relationships.

2. Background and related work

2.1 Background

Our design was guided in part by our use of the Cognitive Dimensions framework (CDs) [9] 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?" [9]. 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 Economics [2]. Attention Economics 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 Economics 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.

2.2 Related work: correctness in end-user programming

Guards are the same idea as assertions, which are found in some programming languages for professional programmers (e.g., [7, 20]). Assertions are used to help keep errors out of programs, and there is research indicating their effectiveness at detecting runtime errors [17]. However, assertions have not previously been aimed at end-user programmers.

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, end users work much better with a concrete program they are able to criticize [14].

A need for some form of explicit requirement specifications present in the spreadsheet seems indicated by one study that found that if people have a vague recollection of requirements, they will not take the time to look them up in another document [8]. 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." We propose an explanation for this behavior from an Attention Economics perspective: Users simply want to be efficient. That is, even when the information was accessible, users would still lose time retrieving the values, such as by context switching from working with the spreadsheet system to finding the right document and looking things up in it. Our approach 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.

Our group has worked in another way to help end users with the correctness of their programming, with a visual methodology for testing that allows users to incrementally edit, test, and debug their spreadsheets as their spreadsheets evolve [4, 16, 18, 19]. 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. Some of WYSIWYT's features have recently also been adapted for the visual dataflow paradigm of Prograph [10]. In studies we have conducted, subjects performed significantly better in testing, debugging, and maintenance tasks with the help of WYSIWYT (e.g., [11]). WYSIWYT is seamlessly integrated into the Forms/3 environment, and our approach to guards is integrated with WYSIWYT.

There is also research regarding helping end-user programmers find errors through outlier analysis [13]. 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 much more than was intended, or might not replace everything intended if there were slight spelling differences in the document, and the attention cost required for a user to check each change is often too much. Their 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.

3. Design constraints for many-to-many guards and cells

To guide our investigation into how to handle multiple guards on one cell, we have developed five design constraints, which draw from several researchers' work relevant to end-user programming.

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 [5], 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.

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 [1, 5].

Design Constraint 4 is to not demand unwarranted attention from the user. Drawn from Attention Economics, 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 obtainable visual feedback after an edit.

The design constraints were used to help shape the approach, as will be seen throughout this paper, 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.

4. How do users expect to reason about the many-to-many relationships?

To investigate these issues, we conducted a small think-aloud study [6] with five end-user subjects. Think-aloud studies are particularly well suited to learning qualitative information about behaviors, such as strategies employed, and reasons for those behaviors. Controlled experiments provide only indirect evidence of the cognitive processes involved in these activities. In contrast, verbal protocols provide direct clues about behaviors and activities. We were interested in *how* end users reasoned about information provided by guards, and a think-aloud study can provide this kind of information.

A previous experiment conducted by our collaborators had already shown that users understand guards when grid-oriented issues are not present [21]. Thus, it was not necessary for us to explore whether users could understand the basic ideas of guards in this study.

4.1 Procedures

The study was conducted one-on-one in small study rooms. We conducted the study using Excel-like grids with sketchy icons on paper such as the one shown in Figure 3, which were then hand-annotated by the examiner (simulating the computer's feedback) as the problems progressed.

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; research has shown that subjects are less likely to criticize software that has a "finished" appearance (e.g., [12]). We could have chosen to use either Excel or Forms/3, since the paper-based study did not require an implementation; we opted for Excel because it had the advantage of staying as close to these users' previous experiences as possible, which helps avoid some kinds of confounding factors.

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. The experiment began with a tutorial on what spreadsheet guards mean and some practice thinking aloud. The subjects' understanding was monitored through questions and a post-tutorial test, and if necessary sections were repeated until the subjects exhibited competence in the necessary skills.

Once the subjects began working the problems, we tape-recorded the sessions, and also kept their paperwork. Pen color was changed between each task to differentiate the work done on each different task. 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 make the spreadsheet work as stated in the problem description. The examiner also interviewed the subjects after they had completed all the tasks.

4.2 The spreadsheet problems and tasks

The spreadsheet in Figure 3 is representative of the spreadsheet problems used. This spreadsheet was based on a real-world spreadsheet, namely the one used by the VL'96 conference organizers. This simplified version listed attendees (with names changed) and their tutorial fees. The possible prices of tutorials were $0 if not attending, the student price of $130, or the regular price of $145. The spreadsheet given to the subjects included these specifications as guards on each of the tutorial columns. The subjects' first task was to place an appropriate guard on Sue's row reflecting the fact that Sue was a student.

For the spreadsheet in Figure 3 and for three additional spreadsheets (Grades, a grades problem similar to Figure 2; Sales, a sales accounting spreadsheet; and Wait Time, a customer waiting time projection), the subjects read a problem description and then performed the following five tasks:

Task 1: Subjects needed to place a guard on a specific row of the spreadsheet.

Task 2: Subjects were told to make the spreadsheet work as described in the problem description. This required making decisions about any guards needed and where they should go in the spreadsheet, to make sure "bad" values would not go unnoticed.

Task 3: Subjects played the role of the computer to determine the correctness of the following values (which were specified by the examiner interactively).

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

Task 5: Subjects were 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.

5. Bringing the think-aloud results into the design

There are important differences between user-entered guards versus computer-generated guards, and between guard conflicts and value violations. To keep the discussion clear, we make precise the vocabulary terms important to this section:

A *user-entered guard* on cell X is a guard typed in by the user on cell X or for a group of cells (e.g., an entire row or column) containing that cell.

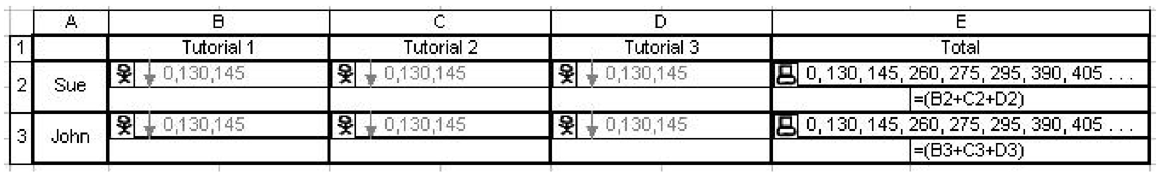
A *computer-generated guard* is a guard that results from propagating another guard through a formula.

A *guard conflict* occurs if and only if two guards do not agree. (What constitutes disagreement will be explored later in this paper.) If the guard conflict involves only user-entered guards, it is a *user-user guard conflict.* If the conflict is between a user-entered guard and a computer-generated guard, it is a *user-system guard conflict.*

A *value violation* occurs if and only if a value does not agree with its guards. (Again, disagreement will be explored later.)

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

As we have said, given that a cell's guards can come from both its row and its column, there can be multiple guards explicitly entered by the user on the same cell such as by virtue of row and column intersections as in Figure 2. At the outset, we wondered whether the users would regard this as an error situation, or might instead regard it as being



	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 3: One of the problems the subjects worked on in 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 Total column. Guards with down arrows were replicated down the entire column.

useful and/or valid in some sense, requiring further reasoning by the system during propagation, for example. To explore this issue, we consider it first from the many-to-one perspective (many guards on one cell), and then from the one-to-many perspective (one guard on many cells).

Thus, the first question to address was whether users regarded having many-to-one relationships among guards and cells as being valid and/or useful.

5.1.1 What the subjects did. 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 had come 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. For example:

S1 (thinking aloud): "I wouldn't want to make the president unhappy [refers to a guard on the president's row of the Wait Time problem], so I would probably just go ahead and get rid of [the extra guard]."

Subjects 2 and 5 were somewhat more tolerant of multiple user-entered guards on one cell. For example, Subject 5 did not remove the extra guard the examiner put on the cell in the Wait Time problem, but seemed uncomfortable with it by the end of the problem. In the interview afterward, the examiner followed up, asking:

Examiner: "Are you comfortable giving the spreadsheet to the customer?"
S5: "...They wouldn't want something that isn't matching."

She also stated:

S5: "... 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."

Similarly, Subject 2 seemed uncomfortable with the extra guards, but did not remove them. By the last problem she was expressing her discomfort with the multiple guards:

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

Subjects 1, 4, and 5 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 3:

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

Unlike the range of views on the many-to-one relationships, subjects consistently chose to make use of the ability to work with one-to-many relationships (one guard for many cells). All 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. For example:

S4 (thinking aloud): "I'll just take off the 145 …guard for [the row labeled] Sue."

S1 (during interview): "I would find myself … crossing the 0-100 out."
Examiner: "For the whole row?"
S1: "Yes."

5.1.2 Impact on the design. It is interesting to consider 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 them is 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. That is, to support the one-to-many relationships, it is necessary to also support the many-to-one relationships that arise at row/column intersections.

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, the way the system explains guard conflicts is, if a user mouses over the guard conflict circle, a one-sentence message appears saying "all guards must match."

5.2 Issue: How should multiple 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 chose 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.

5.2.1 Which guards "win"? Given the findings of Section 5.1, it is not surprising that Subjects 1 and 4 always immediately selected a guard that should "win" (and thus propagate forward), and deleted the other guard. When faced with the propagation question, Subject 2 did the same:

S1: "Just going to cross off the 145 on each column of Sue's."

S2: "I want to change the president's time to the 0-5 seconds for each of them because he's different."

S4: "It seems like since Sam's a special case you could just change the range of his guards from 0-100 to 70-100."

On the other hand, Subject 5 decided that retaining all user-entered guards available to all cells was important, 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: "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 denote 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."

Even when they did not "win" a conflict, Subject 5's lower-precedence guards still propagated forward.

Subject 3's solution was also precedence based, but guard by guard rather than conflict by conflict. This subject 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 problem (which required users to make sure the company's president did not have to wait long for service), when she added a guard to one of the president's cells, she said:

S3: "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 each other 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.

5.2.2 Impact on the design. The subjects demonstrated a variety of propagation decisions that were all reasonable. Thus, the approach 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 the next section.) To resolve conflicts among multiple user-entered guards, one option we have 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. 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 lists among guards in a generic fashion, and can also set up overriding precedence lists for a group of rows, columns, or a single cell. The most specific precedence list "wins." 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.

5.2.3 How the subjects propagated the guards. 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 chose to propagate guards when there were guard conflicts. What we observed was that the subjects did not exhibit any coherent or consistent propagation strategy in the presence of guard conflicts. Because of this, we decided not to propagate conflicting guards. This decision was motivated largely 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.

5.3 Issue: what constitutes a conflict?

We have mentioned that in our early prototype that handled only one-to-one relationships, 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.

When we thought about this issue, we were able to predict two possible approaches upon which subjects might base their decisions: basing decisions on the purposes of the guards involved, or using set-based reasoning. An example of reasoning based on guard purpose would be to choose the "0 to 100" guard for Sam's Midterm as being not in conflict (due to greater importance) than the "70 to 100" guard, because the first is a validity guard whereas the second is more of an omni-present query. An example of using set-based reasoning would be to decide that if there is intersection, there is no conflict, such as deriving "10 to 20" if a cell had one guard of "1 to 20" and a second guard of "10 to 30."

Because we thought of these two 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. However, we did not restrict ourselves to looking for only these possible reasoning patterns.

5.3.1 How the subjects defined guard conflicts. Our work in devising spreadsheet problems whose 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.

On the other hand, as Table 1 indicates, set-based reasoning was extremely common in reasoning about guard conflicts, primarily (but not always) using intersection. In other words, 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.

5.3.2 How the subjects dealt with value violations. We chose to also have subjects identify value violations by circling cell values not satisfying the relevant guards, because doing so would require them to think deeply about the implications of the guards and guard conflicts.

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

Table 1: 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.

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 1 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: "[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."

Recall that Subject 2 used union-based reasoning about user-user guards. When faced with determining value violations, she changed her opinion. 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. When she noticed the guards were different she commented:

S2: "Fine in one, but not in the other. I would change [the guard] because [the value is] fine in one and not in the other."

5.3.3 Impact on the design. 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 were not sound—they would result in incorrect resolutions of conflicts. We have already explained the problem with the "all-knowing computer." 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 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 4 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

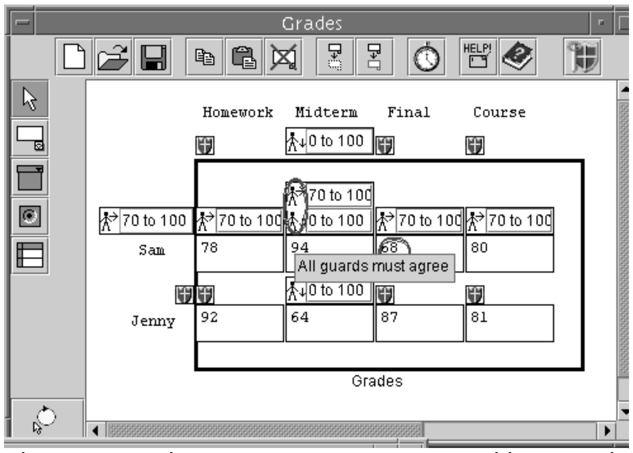


Figure 4: The current prototype, with guards displayed on the second column and the top row. The value of Sam's final is circled because it has a value violation. There is also a guard conflict, on the top row's second cell, which is circled in red. The user has moved the mouse over the guard conflict, causing the explanation "All guards must agree" to pop up.

is the fact that a reasoning explanation can be given in just one sentence, which is helped by following Design Constraint 2 (consistency) to avoid special-case reasoning.

6. Conclusion

In this paper, 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 Economics, 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 to the design an explanation-based approach 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 are working to provide this support via an explanation-based approach for all aspects of the system's behind-the-scenes reasoning.

Acknowledgments

We thank the members of the Visual Programming Research Group at Oregon State University for their feedback and help. This work was supported in part by the National Science Foundation under Awards CCR-9806821 and ITR-0082265.

References

[1] Belkin, N. Helping people find what they don't know, *Comm. ACM* 41(8), Aug. 2000, 58-61

[2] Blackwell, A. and Green, T. R. G. 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.

[3] Burnett, M., Atwood, J., Djang, R., Gottfried, H., Reichwein, J., and Yang, S. Forms/3: a first-order visual language to explore the boundaries of the spreadsheet paradigm, *J. Functional Programming*, Mar. 2001, 155-206.

[4] Burnett, M., Sheretov, A., Ren, B., and Rothermel, G. Testing homogeneous spreadsheet grids with the 'What You See Is What You Test' methodology, *IEEE Trans. Software Engineering*, June 2002 (to appear).

[5] Corritore, C., Kracher, B., and Wiedenbeck, S. Trust in the online environment, *HCI International, Vol, 1*, New Orleans, LA, Aug. 2001, 1548-1552.

[6] Ericsson, K. and Simon, H. *Protocol Analysis*, MIT Press, Cambridge, MA, 1984.

[7] Ernst, M., Cockrell, J., Griswold, W., and Notkin, D. Dynamically discovering likely program invariants to support program evolution, *Int'l. Conf. Software Engineering*, Los Angeles, CA, May 1999, 213-224.

[8] Gray, W. and Fu, W. 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, Mar. 2001, 112-119.

[9] Green, T. R. G. and Petre, M. Usability analysis of visual programming environments: a 'cognitive dimensions' framework, *J. Visual Languages and Computing* 7(2), June 1996, 131-174.

[10] Karam, M., Smedley T. A testing methodology for a dataflow based visual programming language, *IEEE Int'l. Conf. Human Centric Computing*, Stresa, Italy, Sept. 2001, 280-287.

[11] Krishna, V., Cook, C., Keller, D., Cantrell, J., Wallace, C., Burnett, M., and Rothermel, G. Incorporating incremental validation and impact analysis into spreadsheet maintenance: an empirical study, *IEEE Int'l. Conf. Software Maintenance*, Florence, Italy, Nov. 2001, 72-81.

[12] Landay J. and Myers, B. Sketching interfaces: toward more human interface design, *Computer* 34(3), Mar. 2001, 56-64.

[13] Miller, R., and Myers B. Outlier finding: focusing user attention on possible errors, *ACM Symp. User Interface Software and Technology*, Orlando, FL, Nov. 2001

[14] Nardi, B. *A Small Matter of Programming: Perspectives on End-User Computing*, MIT Press, Cambridge, MA, 1993.

[15] Panko, R. What we know about spreadsheet errors, *J. End User Computing*, Spring 1998.

[16] Reichwein, J. Rothermel, G., and Burnett, M. Slicing spreadsheets: an integrated methodology for spreadsheet testing and debugging, *Conf. Domain-Specific Languages*, Austin, TX, Oct. 1999, 25-38.

[17] Rosenblum, D. A practical approach to programming with assertions, *IEEE Trans. Software Engineering 21(1)*, Jan. 1995, 19-31.

[18] Rothermel, G., Li, L., DuPuis, C. and Burnett, M. What you see is what you test: a methodology for testing form-based visual programs, *Int'l. Conf. Software Engineering*, Kyoto, Japan, Apr. 1998, 198-207.

[19] Rothermel, G., Burnett, M., Li, L., DuPuis, C., and Sheretov, A. A methodology for testing spreadsheets, *ACM Trans. Software Engineering and Methodology*, Jan. 2001, 110-147.

[20] Sankar, S., and Mandal, M. Concurrent runtime monitoring of formally specified programs, *Computer* 26, Mar. 1993, 32-41.

[21] Wallace C., Cook, C., Summet, J., and Burnett, M. Assertions in end-user software engineering: a think-aloud study, *IEEE Symp. Human-Centric Computing Languages and Environments*, Arlington, VA, Sept. 2002 (tech note, to appear).