Inferring Templates from Spreadsheets*

Robin Abraham
School of EECS
Oregon State University
abraharo@eecs.oregonstate.edu

Martin Erwig
School of EECS
Oregon State University
erwig@eecs.oregonstate.edu

ABSTRACT

We present a study investigating the performance of a system for automatically inferring spreadsheet templates. These templates allow users to safely edit spreadsheets, that is, certain kinds of errors such as range, reference, and type errors can be provably prevented. Since the inference of templates is inherently ambiguous, such a study is required to demonstrate the effectiveness of any such automatic system. The study results show that the system considered performs significantly better than subjects with intermediate to expert level programming expertise. These results are important because the translation of the huge body of existing spreadsheets into a system based on safety-guaranteeing templates cannot be performed without automatic support. We also carried out post-hoc analyses of the video recordings of the subjects' interactions with the spreadsheets and found that although expert-level subjects needed less time and developed more accurate templates than less experienced subjects, they did not inspect fewer cells in the spreadsheet.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement; H.4.1 [**Information Systems Applications**]: Office Automation—spreadsheets

Keywords

Spreadsheet Specification, Template Inference, End-User Software Engineering

1. INTRODUCTION

A study conducted this year based on data from the U.S. Bureau of Labor Statistics shows that there are currently as many as 11 million end-user programmers in the United States, compared to only 2.5 million professional programmers [32]. Many of these end-user programmers develop spreadsheets. Moreover, the number of American workers who use spreadsheets is even higher, about 23 million workers, which amounts to 30% of the workforce. Numerous studies have shown that existing spreadsheets contain errors at an alarmingly high rate [6, 19, 23, 33]. Some studies report that up to 90% of real-world spreadsheets contain errors [27]. These errors impact people directly because they use spreadsheet systems, and indirectly by the decisions that are based on spreadsheet calculations.

Spreadsheet systems offer users a high level of flexibility. This aspect makes it easier for people to get started working with spreadsheets. The downside is that this freedom also offers ample opportunity to create erroneous spreadsheets. Errors during creation of a spreadsheet are made as well as when modified by other users. The problem gets further exacerbated when the people who use or modify the spreadsheet do not fully understand its functionality. This situation arises because spreadsheet systems do not offer any higher-level abstractions. Moreover, data and computation are not separated in spreadsheets, and the immediate visual feedback mechanism makes traditional coding and program compilation/execution steps indistinguishable from each other. These factors make widespread reuse of spreadsheets difficult and prone to errors.

Since a spreadsheet is essentially a program, we address the problem along the lines of traditional Software Engineering approaches to software development. The key aspect of our approach is that we separate the modeling and data-entry aspects of spreadsheet development. We have developed a visual language called ViTSL (an acronym for **vi**sual **t**emplate **s**pecification **l**anguage) [3] for modeling spreadsheet templates. The user can import a ViTSL template into Gencel [11, 12], a spreadsheet system we have developed as an add-on to Excel, and create and edit spreadsheets that are guaranteed to conform to the template.

In Figure 1, on the left, we show how spreadsheets are usually developed. In this case, the application-level and data-level updates are both performed on the spreadsheet directly. On the right we show the ViTSL/Gencel model of spreadsheet development. In this case, the application-level updates are performed on the ViTSL template, while the *safe* data updates are performed on the spreadsheet. The updates are safe in the sense that they are customized according to the template and the user is only allowed to change data values. The system prohibits direct changes to the spreadsheet formulas. Formulas will be automatically updated whenever rows or columns are inserted or deleted. The spreadsheet generator component of the framework al-

*This work is partially supported by the National Science Foundation under the grant ITR-0325273 and by the EUSES Consortium (`http://EUSESconsortium.org`).

International Conference on Software Engineering 2006, Shanghai, China

lows the user to generate the spreadsheets from the ViTSL template.

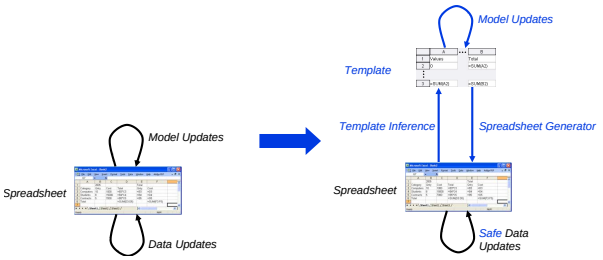


Figure 1: Vitsl/Gencel model of spreadsheets.

In the original scenario, once a template was created and loaded in Gencel, it was not possible to change the template and have the changes propagate to the already created spreadsheet data. Moreover, templates had to be developed from scratch. That is, there was no way of inferring a template from an already existing spreadsheet, which limits the applicability of this approach and makes a transition very costly.

In this paper we address this problem and describe a method for inferring templates from spreadsheets. The template inference component shown highlighted in Figure 1 complements the spreadsheet generator and enables a broader and more flexible use of the ViTSL/Gencel approach. However, a challenge is presented by the fact that the template inference process is inherently ambiguous. Therefore, in order to judge the effectiveness of the developed method, we have performed a study to assess the reliability of template inference.

The rest of this paper is structured as follows. In the next section we describe our template-based approach that protects spreadsheet users from a large class of errors. In Section 3 we describe a method that we have developed to extract templates from spreadsheets and discuss the implementation and working of the system with a couple of examples. In Section 4 we describe a study we carried out to evaluate the system. Related work is described in Section 5, and we present future work and conclusions in Section 6.

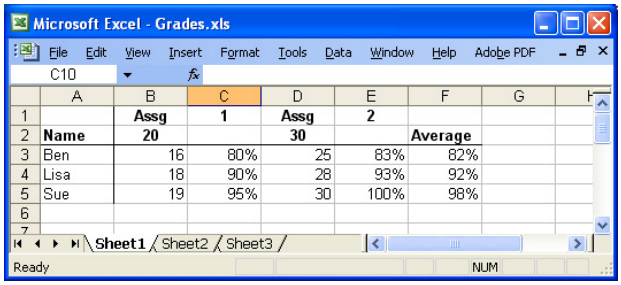
2. TOWARDS SAFER SPREADSHEETS

In this section, we illustrate some common problems in existing spreadsheet systems through an example. In particular, we illustrate how errors can be introduced into spreadsheets. We then show how the errors can be avoided by using the ViTSL/Gencel system.

2.1 A Scenario

Sharon is an elementary school teacher who has created a grading spreadsheet for her class, in which she records points for students on individual assignments, see Figure 2. This spreadsheet contains one row for each student and two columns for each assignment. Since different assignments have different total number of points in general, the spreadsheet stores for each student and assignment, the number of points earned by the student as well as the percentage of that number with respect to the total number of points for that assignment. The overall performance of each student is computed in the rightmost column by an average of the percentages over all the assignments.

After having added several students, Sharon notices that her formula for computing percentages, =B3/B2, was not

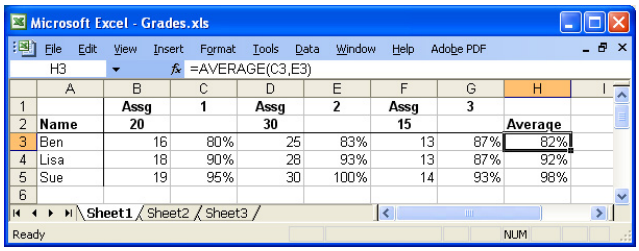


	A	B	C	D	E	F	G
1		Assg	1	Assg	2		
2	Name	20		30		Average	
3	Ben	16	80%	25	83%	82%	
4	Lisa	18	90%	28	93%	92%	
5	Sue	19	95%	30	100%	98%	
6							

Figure 2: Grade sheet.

properly propagated to the newly inserted rows. After some time she figures out that the column number of the cell containing the total number of points must not be relative, but an absolute address. Therefore, she changes the formula to =B3/B$2.[1]

After she has graded a new assignment, Sharon adds the results into the spreadsheet. She inserts two columns and fills in the data. However, she notices that the average for the first student seems to be too low, see Figure 3. Inspecting the formula in cell H3, she learns that Excel has not automatically updated the formula, which is still =AVERAGE(C3,E3), representing an average over the non-contiguous range.[2] Therefore she has to update all the formulas in column H by hand, that is, she changes the formula in H3 to =AVERAGE(C3,E3,G3), and similarly for cells H4, H5, etc. The procedure is time consuming and prone to errors. Even worse, she realizes that she has to repeat this update ordeal again and again for every new assignment she wants to add.



	A	B	C	D	E	F	G	H
1		Assg	1	Assg	2	Assg	3	
2	Name	20		30		15		Average
3	Ben	16	80%	25	83%	13	87%	82%
4	Lisa	18	90%	28	93%	13	87%	92%
5	Sue	19	95%	30	100%	14	93%	98%
6								

Figure 3: Grade sheet after updates.

This example demonstrates that update operations offered by existing spreadsheet systems are weak and ill-defined in the sense that they do not provide adequate safety guarantees and make it easy to introduce errors. What makes errors like the one shown particularly harmful is that they are generally not introduced in a single cell, but can invalidate many cells at once. The study reported in [6] found that 65% of all spreadsheet errors are contained in formulas.

The fact that a semantic update operation, the insertion of a new assignment, has to be implemented by Sharon in terms of a number of low-level operations (namely, two column insertions, copying of formulas, and adjusting multiple formulas) is problematic since it is not enforced by the spreadsheet

[1]At this point, many non-professional spreadsheet users would have probably not gone all the way to figure out the correct referencing mode, which would have caused the spreadsheet to be already incredibly difficult to maintain.

[2]If the range was contiguous, Excel would update the formula automatically and include the newly-inserted cell.

system that all the required steps be performed. Therefore, any omission might leave the spreadsheet in an inconsistent state. Moreover, each individual step presents another opportunity to introduce errors into the spreadsheet.

One reason for this situation is that existing spreadsheet systems work with a simple programming model of a flat collection of cells that do not contain any structure other than their arrangement on a grid. This lack of modularity and abstractions has been reported as a major weakness of spreadsheet systems [20]. One particular problem is that cells are identified by global row and column numbers (letters) so that references to cells or subareas of a spreadsheet have to be expressed using these global addresses. The global cell addressing schema has been blamed for complicating the comprehension of spreadsheets and for location errors [58].

The lack of structure and abstraction puts current spreadsheet systems into the category of assembly languages when compared to the state of the art in other programming languages. This situation is peculiar because spreadsheet systems are equipped with very sophisticated user interfaces offering many fancy features, which can distract from their intrinsic language limitations. The rigid, global addressing scheme makes computations vulnerable to changes in the structure of the spreadsheet—much like in the old days of assembly language programming where the introduction of a new item into the memory could cause some references to become invalid. Related is the problem of viscosity, which means the difficulty of changing one part of a program without changing other parts [16]. In the presented example, high viscosity can be observed, for example, when the total number of points per assignment is moved one cell to the right. In that case, it is necessary to change all percentage formulas in that column afterwards. Studies have shown that users try to exploit the surface structure of spreadsheets [30] and that spreadsheets should therefore make their inherent structure visible.

Next we will outline an approach for explicitly representing and enforcing structure in spreadsheet applications that follows these insights. By separating model and data updates into two layers, many of the described problems can be avoided. In particular, a large class of spreadsheet errors can be exterminated from spreadsheets altogether.

2.2 Safer Spreadsheets with ViTSL/Gencel

The model layer of a spreadsheet application can be described by a visual language for structuring spreadsheets, allowing reuse and preventing errors. The idea originates by noticing that a given spreadsheet may evolve in a number of predictable ways, and various instances of a spreadsheet could emerge from a common template. The visual language ViTSL provides a method for modeling the template of a spreadsheet and the ways it can evolve [3].

ViTSL templates are constructed with an editor and are loaded into Gencel [11], which is an Excel extension that manages the evolution of a spreadsheet from a ViTSL template. This environment automatically handles all formula generation and spreadsheet structure modification, ensuring that all spreadsheet formulas are correct and allowing the user to focus on data entry and analysis. Templates also act as a documentation to describe the functionality of the spreadsheet without reference to particular instances.

From the example presented in Section 2.1 we can observe that once the structure of the spreadsheet application has been fixed, the teacher progresses by performing basically three kinds of updates: add another student (row), add another assignment (two columns), or update points and labels (for assignments or student names). The teacher may also choose to delete an assignment or a student, although this is probably less common.

On a closer look, we can observe that each of these operations can be broken down into a fixed set of necessary steps, in particular, adding rows or columns and updating formulas and data. In this way, an initial spreadsheet with one assignment, a spreadsheet with two assignments, and a spreadsheet with seven assignments are all related. In this sense, the spreadsheets from Figures 2 and 3 (once corrected) can be thought of as deriving from the one shown in Figure 4 (shown in formula view).

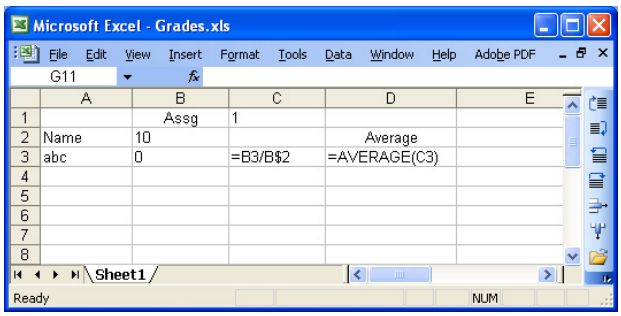


Figure 4: Grade sheet in Gencel.

From this sample sheet, any number of spreadsheets may be derived using the operations provided by Gencel. These operations, which consist of row or column insert, value update, and row or column delete, are specialized for this particular sample sheet to ensure that updates occur correctly with all necessary changes. For example, if Sharon presses the insert column button (see right panel in Figure 4) when the cursor is within an assignment group, two new columns representing a new assignment will be inserted at once and all the formulas (the percentage formulas as well as the average formula at the far right) will be correctly updated instantly.

The Gencel system provides these specialized updates to ensure the correctness of formulas. Since the sample sheet is generic with respect to the actual students and assignments, and other labels and values, it may be reused by various users at different times. In all cases, the safety and correctness of the formulas and structure within the Gencel system is assured.

From the sample sheet shown in Figure 4 it is not immediately clear which columns and rows are fixed and which are expandable, which makes the inference process challenging. However, the creator of the grading spreadsheet application would know about the intended behavior and could specify the corresponding information, in this case a two-column horizontal expanding group, called *hex* group, which forms an assignment, and a single-row vertical expanding group, called *vex* group, for each student. In addition, an aggregation formula that computes the average of the percentages for each student is contained in the hex group to the right of the vex group, and so on. By abstracting out the building blocks from the concrete Gencel spreadsheet in this way, we can fully and formally describe the operations required

to create a spreadsheet. This is the purpose of ViTSL—to provide a visual specification language for spreadsheets and their evolutions. The ViTSL template for the above Gencel spreadsheet is shown in Figure 5. The vex group is represented by the ellipsis ⋮ following row 3, which can be expanded. Similarly, the hex group is represented by the ellipsis ⋯. The fact that the hex group consists of two columns is represented by the absence of the separating line between the column headers B and C. In addition to the formulas, the template consists of labels, such as Assg and Name, that will generally not be edited in the generated Gencel spreadsheet and the sample values, such as 10, abc, and 0, that will be edited.

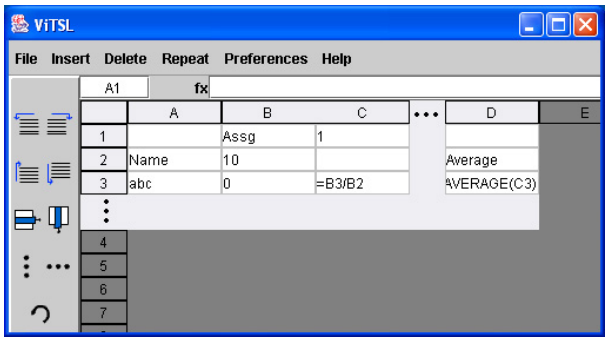


Figure 5: Grade sheet template in Vitsl.

Using Gencel, Sharon simply has to load the ViTSL template and then press the insert column button two times to create the assignments. All formulas are updated correctly and automatically and are protected against unintended changes. Similarly, for adding a new student, pressing the insert row button is all that is needed to update the formulas in the spreadsheet. Therefore, Sharon can concentrate on entering data and does not have to worry about formulas. In particular, the errors illustrated in Section 2.1 would have been prevented using Gencel. In general, Gencel provably eliminates the following kinds of errors from spreadsheets [12].

- *Range errors* (for example, omitted or additional cells in aggregations)
- *Reference errors* (for example, references to wrong cells or circular references)
- *Type errors* (for example, using strings in numeric computations)

The impact of these errors have been extensively documented. For example, a range error has caused a Florida construction company to underbid a project by a quarter of a million dollars [17]. An example of a type error is the illegal interpretation of a date as a numeric value, which caused an operating fund of the Colorado Student Loan Program to be understated by $36,131 [34]. Finally, a reference error caused a hospital's records to overstate its Medicaid/Medicare crossover log by $38,240 [35]. The use of Gencel would have prevented all these errors.

3. EXTRACTING TEMPLATES FROM SPREADSHEETS

We anticipate that Gencel will be used by spreadsheet users working with ViTSL templates developed by domain experts who have some programming experience. In the case of legacy spreadsheets, it would be vital (from an adoption point of view) to have tools that extract the templates automatically. In this section we discuss algorithms for extracting ViTSL templates from spreadsheets. This effort is a first step towards reverse engineering spreadsheets. In related work, we have developed *ClassSheets* [10], which is a more expressive form of spreadsheet specifications. ClassSheets could potentially also be the target of future reverse-engineering efforts.

There is a high level of ambiguity associated with spreadsheet template inference since spreadsheets are the result of a mapping of higher-level abstract models in the user's mind to a simple two-dimensional grid structure. Moreover, spreadsheets do not impose any restrictions on how the users map their mental models to the two-dimensional grid (flexibility is one of the main reasons for the popularity of spreadsheets). Therefore the relationship between the model and the spreadsheet is essentially many-to-many, and we suspect that template inference of spreadsheets will generally require user input to resolve ambiguities. The current version of the system only displays one (the first) template it comes up with. In future versions we plan to incorporate interaction mechanisms by which the user can pick from a list of possible templates. Another problem is that, in some cases, the spreadsheet being considered might not have enough information for the correct template to be inferred. For example, in the spreadsheet shown in Figure 2, if data for only one student was present, the template inference system should be able to identify the hex group but it simply does not have information to identify the vex group (for the student data).

While developing the algorithms for the system, we were guided by two principles.

1. The generated template should be the smallest possible, starting from which the user should be able to generate the target spreadsheet using only Gencel insert/delete row/column commands and changes to data cells.
2. The system should be tolerant to errors within the spreadsheet. The user should be able to control the tolerance threshold.

In the following subsections we discuss the steps involved in extracting ViTSL templates from spreadsheets. We use the corrected version of the grade sheet shown in Figure 3 as a running example to explain the steps involved in template inference.

3.1 Identifying Tables in Spreadsheets

We have observed in some cases that end users put unrelated information in the same spreadsheet (maybe so they have all their data in the same sheet). We define a *table* as (part of) a spreadsheet that is an instance of a ViTSL template. In case the user has unrelated information in the same spreadsheet, we are faced with the scenario of a single spreadsheet containing multiple tables. It is therefore important to identify the different tables within a spreadsheet since inferring a common template for unrelated data that just happens to be in the same sheet would be a mistake. We have reused some spatial analysis algorithms from the UCheck tool [1] to break up the spreadsheet into connected cell areas we treat as tables. In the grade sheet shown in Figure 3, the cell area from A1 to H5 is a single table.

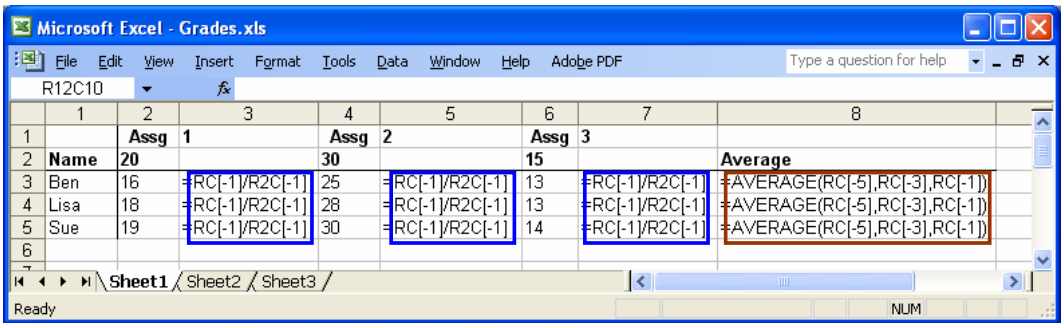


Figure 6: CP-similar regions in grade sheet.

3.2 Identifying "Similarity" Regions Within Tables

Once areas containing different tables have been found, the next step is to identify regions within each table area containing *similar* formulas. The idea is to reduce sets of similar formulas to hex and vex groups. We follow a strategy of identifying *maximal* sets of similar formulas which maximizes the number of instances of repeating groups and thus minimizes the size of the inferred template. Since the described approach hinges on the notion of cell similarity, we will discuss this notion next.

Two formulas are similar if they satisfy the *cp-similarity* criterion described in [8]. Two cells are cp-similar if their formulas could have resulted from a copy/paste action from one of the cells to the other. An *absolute* reference points to a particular cell in the spreadsheet and will point to the same cell even if the reference is copied to another cell in the sheet. A *relative* reference refers to a cell based on its position relative to the cell containing the reference. If a relative reference is copied to another cell, it will point to a cell at the same relative position with respect to the new location. Excel allows two reference schemes in cells.

1. In the *A1-style* referencing scheme, relative references are of the form A2 (both the row and column change when the reference is copied to a new cell) and absolute references are of the form A$3 (the row number remains unchanged if the reference is copied to a new location), $A3 (the column number remains unchanged if the reference is copied to a new location), or A3 (both the column and rows remain unchanged if the reference is copied to a new location.
2. In the *R1C1-style*, a reference B3 in cell C3, for example, would be represented as RC[-1]—reference the cell in this row and one column to the left of this one. Along similar lines, a formula =B3/B$2 in cell C3 could be represented as =RC[-1]/R2C[-1] in the *R1C1* style.

We follow the approach described in [8] and decide two formulas are cp-similar by comparing their *R1C1*-style representations.

The cp-similar formula cells in the grade sheet have been marked in Figure 6. Note that column headers are numbered in *R1C1*-style in Excel. The cells enclosed by the blue rectangles all have the formula =RC[-1]/R2C[-1]. All the cells within the brown rectangle (in column 8) have the formula =AVERAGE(RC[-5],RC[-3],RC[-1]). Simply by comparing the *R1C1*-style representations of the formulas, the system can infer the two cp-similar regions (the one enclosed by the blue rectangles and the one enclosed by the brown rectangle) within the spreadsheet.

The cells whose formulas have been found to be cp-similar are grouped on the basis of rows and columns. The cp-similar blocks are indicators for repeating groups. For example if the formula cells in one row are cp-simlar to cells in the same columns in another row, the two rows could be instances of the same vex group. The system does a column-wise and then a row-wise partitioning of the cp-similar cells. This sequence is followed simply because ViTSL only allows nesting of vex groups within hex groups. Note that this representation is as expressive as only allowing nesting of hex groups within vex groups. The column-wise partitioning generates the lists [C3,C4,C5], [E3,E4,E5], and [G3,G4,G5] as potentially belonging to the same hex group. Similarly, the row-wise partitioning generates the lists [C3,E3,G3,H3], [C4,E4,G4,H4], and [C5,E5,G5,H5] as (parts of) potential expansions of the same vex group.

3.3 Inferring Templates

Once the cells within a table area have been partitioned into regions containing cp-similar formulas, the system tries to overlay them (along with the regions they refer to) to generate the templates. In addition to the formula cells, we also compare the referenced data cells in the two rows to check if they have the same type. If the corresponding formula cells are cp-similar and the corresponding data cells are of the same type, we have a perfect match. For example, based on the column-wise partitioning of the cp-similar cells, the system tries to overlay the cells in the lists [C3,C4,C5] and [E3,E4,E5]. The cells in the first list have references to the cells B2, B3, B4, and B5, and the cells in the second list have references to the cells D2, D3, D4, and D5. The system compares the corresponding referenced cells to check that they have the same types. If this condition is satisfied, we have strong indication that columns D and E together come from the same hex group as columns B and C. The same reasoning is applicable to columns F and G as well, and they too can be considered to be instances of the same hex group as columns B and C. Along similar lines, rows 3, 4, and 5 are inferred to be the instances of the same vex group.

In some cases, the data cells might not agree, for example, if the data in a cell has been omitted. Figure 7 shows part of a grade sheet that was used in the study. The rows that store information for each of the students are all part of the same vex group. The data in row 10 differs from the others since the student dropped the course in week 2. Because of this, the lab and quiz score entries for this student are all blank from E10 onwards in the row. The system is tolerant to such minor deviations (integer values for the scores in the other rows and blanks in the corresponding cells in row 10) and can nevertheless distill the template for the spreadsheet.

	A	B	C	D	E	F	G	H
1	**Topics:**		**Quiz 1**	**Lab 1**	**Quiz 2**	**Lab 2**	**Quiz 3**	**Lab 3**
2			Gen C++	Triangle	Copy ctor	Fn Objs	Simple	Recursion
3	**Last**	**First**	knowledge	Abstraction	implement		Recursion	Analysis
4	Student	1	90	90	70	95	85	
5	Student	2	50	100	70	70	100	
6	Student	3	100	98	100	85	0	
7	Student	4	90	100	100	95	85	
8	Student	5	40	95	100	60	75	
9	Student	6	83	100	100	100	60	
10	Student	7	40	Dropped in Wk #2				
11	Student	8	50	98	90	90	80	
12	Student	9	100	100	100	95	100	1

Figure 7: Deviations from template.

3.4 Template Inference in Action

In our system, the user can open an Excel spreadsheet and then click the button labeled "Template" (on the right toolbar in Figure 5). The system carries out the automatic extraction of the spreadsheet template as described above (for the grade sheet in the example shown in Figure 5) and displays it in a new worksheet with "-Templ" appended to the name of the original worksheet.

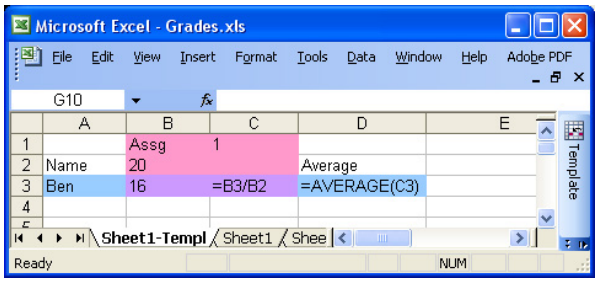


Figure 8: Automatically inferred grade sheet template.

The system shades vex groups light blue and hex groups pink. Cells in the template that are part of vex and hex groups are shaded purple. In case you are reading a black and white printout of this paper, A3 and D3 have been shaded blue, B1, B2, C1, and C2 have been shaded pink, and B3 and C3 have been shaded purple by the system. The system retains some of the values from the spreadsheet as default values in the templates. We made this design choice under the assumption that the default values would serve as an example and help the user get started with the task of modifying the spreadsheet. The default values might also serve as documentation and remind the users of the original spreadsheet from which the template is inferred. Besides the default values, the template shown in Figure 8 is the exact same one shown in Figure 5 in the ViTSL editor. The inferred templates can be saved as ViTSL templates and can be further edited in the ViTSL editor or be directly loaded into Gencel.

The system described above allows the user to adopt a very flexible approach to developing safe spreadsheets within the ViTSL/Gencel framework. The user could start with a ViTSL template and then work with the spreadsheet in Gencel or the user could start with an Excel spreadsheet directly and then infer the ViTSL template using the tool and then continue using Gencel. The user might also start creating a spreadsheet with a ViTSL template loaded in Gencel. At some point, if the user wants to deviate from the initial template, she could turn off Gencel, work in Excel (in an *unrestricted* mode so to speak), invoke the template inference system to generate a ViTSL template for the new spreadsheet, reactivate Gencel and continue working with the spreadsheet. The template inference system puts the safety features of Gencel within the grasp of people and organizations who have spreadsheets they might have invested considerable time and effort in developing.

4. EVALUATION

One particular spreadsheet could potentially be generated from many different templates. This precludes the possibility of automatically validating the correctness of the templates generated by our system by an oracle. The creator of the spreadsheet would be the one in the best position to decide if the spreadsheet and the template generated by the automatic extractor match up. We assume this judgment would become more accurate with increasing experience with spreadsheet systems and the domain. For example, an accountant with considerable experience with spreadsheets would be in a better position to judge the correctness of a template for an accounting sheet than a person without any background in accounting.

To judge the performance of our system, we compare templates generated by the system against those generated by novice and expert subjects. The main goal is to assess the effectiveness/performance of a system that automates the task of extracting templates from spreadsheets. We are also interested in how experts and novices go about the task of inferring templates from spreadsheets. This information can be used for improving the inference tool and its interaction with the users. More formally, we seek to answer the following research questions.

RQ1: How well does the system perform compared to expert and novice test subjects in extracting templates from spreadsheets?

RQ2: Are there any patterns of behavior exhibited by novice and expert subjects when they are trying to understand spreadsheets in order to develop their templates?

4.1 Participants

Nineteen students from a 300-level course on Software Engineering at Oregon State University participated in the study. We refer to this group of subjects as Group N. The course primarily dealt with the specification and design of software. UML was presented as the *de facto* standard modeling language for software, and ViTSL was presented as a language for modeling spreadsheets. Prior programming experience ranged from two to ten years (in two to four languages) and all the participants had between two and eight years of experience using spreadsheets. We chose students from this course as the test subjects because the target audi-

ence for ViTSL are people with a beginning to intermediate level of programming and spreadsheet expertise.

We also enlisted help from four doctoral students working in the area of Programming Languages to serve as expert subjects. These subjects had five to ten years of programming experience (in two to five programming languages) and many years of experience with spreadsheets. They all also have experience with specification languages as part of their Ph.D. studies. We refer to this group of subjects as Group E.

4.2 Study Tasks

For the study, we decided to use spreadsheets from the EUSES spreadsheet corpus [14]. The corpus has 4498 spreadsheets collected from various sources. Since Gencel is not useful for spreadsheets that do not contain formulas, we first isolated the 1977 spreadsheets in the corpus that had formulas in them. We then randomly selected 29 spreadsheets from this set for the purpose of the study.

The 29 spreadsheets were then randomly assigned to the participants in Group N such that each participant was working with 5 or 6 spreadsheets. The participants were asked to look at the spreadsheets assigned to them and develop the ViTSL templates that could be used to generate those spreadsheets. They were asked to sketch the ViTSL template they had come up with on paper and also provide short descriptions for their templates. We were hoping the descriptions would be useful in cases in which the ViTSL templates developed by the participants were ambiguous or in cases in which the participants were not comfortable with ViTSL. We made video recordings of the participants' interactions with the spreadsheets and later used the videos for some of our analyses.

We also asked the participants in Group E to go through the spreadsheets and develop the ViTSL templates for them. Each participant in Group E was randomly assigned the spreadsheets so that each spreadsheet would have two participants from Group E working on it. Again, we made video recordings of the participants' interactions with the spreadsheets for post-hoc analyses.

We ran the system on the 29 spreadsheets and inferred the ViTSL templates for the spreadsheets. One of the authors sketched the templates inferred by the system on paper so that the final output would look similar to the work done by participants from Group N and Group E.

We then randomly assigned all the templates to the experts (ensuring no expert graded their own template) and asked them to grade them on the basis of their *correctness*. The experts graded the templates on the five-point scale shown in Table 1.

Each template was graded by two experts who were not told whether the template was developed by a participant from Group N, Group E, or generated by the system. As a matter of fact, the graders were not even aware that some of the templates had been generated by a system.

4.3 Threats to Validity

The threat to external validity is that the subjects in Group E are not domain experts as far as the spreadsheets used in the study are concerned. Even so, we think it is relatively safe to assume that with their substantial programming and spreadsheet experience, they can be considered experts for the experiment tasks. Moreover, it would be difficult to assemble a group of domain experts for a set of spreadsheet chosen randomly from a large heterogeneous corpus.

5 points	Spreadsheet can be generated from the template by insert/delete row/column commands and data updates exclusively
4 points	Overall structure of the template is correct, and only data or references in formulas in the template are incorrect
3 points	Some parts of the template structure like a vex or hex group were missing
2 points	Subject showed some understanding of templates but misunderstood the spreadsheet and got the template wrong
1 point	Template does not make any sense

Table 1: Scoring Criteria for Templates

A threat to internal validity is the level of comfort of the subjects in groups N and E with templates and modeling languages (especially ViTSL). While the members of Group E have been exposed to ViTSL for over one year during research group meetings, presentations, and other discussions, the members of Group N were only exposed to ViTSL during the course. We have tried to minimize the impact of this factor by allowing the subjects to sketch, on paper, the templates they develop without being too weighed down with getting the ViTSL syntax right. We also made it clear to the expert graders during discussion of the grading criteria shown in Table 1 that the subjects were not to be docked points for not using correct ViTSL syntax.

4.4 Consistency of Raters

As mentioned earlier, each template was rated by two experts. To compare the experts (A, B, C, and D), we determined the Kappa (κ) values for the rating tasks on which different pairs of experts worked together to see how well the ratings agree. The κ values for the pairings of the graders are shown in Table 2, and all of them are greater than 0.6. Therefore, the agreement between the graders is good enough.

Graders	κ
A–B	0.76
A–D	0.71
B–C	0.70
C–D	0.74

Table 2: κ values for grader pairs.

4.5 Results

Figure 9 shows the boxplots of the scores of the different groups E and N and for the system (S).

To answer RQ1, which dealt with the performance of the system when compared to subjects in groups N and E, we carried out the following analyses of the data we collected.

A pairwise comparison of the scores using the Tukey method is shown in Figure 10. We see that none of the 95% confidence intervals include 0.

System versus Group N. The scores of the system-generated templates for the spreadsheets were significantly better than the scores of the templates developed by the subjects in Group N (ANOVA: $F(1,149)=51.69$, $p<0.001$). This result shows that the system is more reliable than the

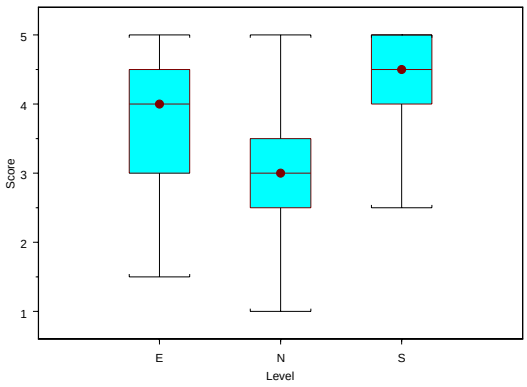


Figure 9: Task scores.

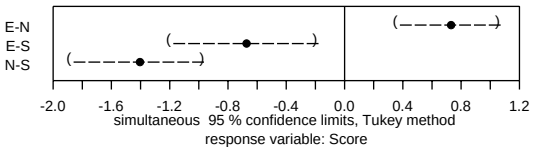


Figure 10: Three-way comparison of scores.

subjects with intermediate level of programming and spreadsheet experience.

System versus Group E. We also compared the scores of the system-generated templates against the scores of the templates developed by the subjects in Group E. We expected the subjects in Group E (the experts) to perform better than the system since they have considerable programming and spreadsheet experience. Instead, we were surprised to find that the system performed significantly better than the subjects in Group E (ANOVA: $F(1,85)=11.75$, $p<0.001$). One possible explanation for this result could be that the spreadsheets in the study were too simple for the experts to outperform the system.

Group N versus Group E. It is reasonable to assume that the expert subjects would outperform the novice ones on the assigned tasks. We compared the scores obtained by subjects in Group N against those obtained by the subjects in Group E just to confirm that this is the case. We see that that the subjects in Group E performed significantly better than those in Group N (ANOVA: $F(1,179)=22.17$, $p<0.001$).

4.6 Discussion

We carried out post-hoc analyses of the video recordings of the subjects' interactions with the spreadsheets to determine how much time they spent on the tasks. The box plots are shown in Figure 11, and we see that the subjects in Group N spent significantly more time on the tasks compared to the subjects in Group E (ANOVA: $F(1,132)=32.82$, $p<0.001$).

We also compared the *inspection profiles* (number of cells inspected by the subject while inferring the template for a spreadsheet) of the subjects in Group N against those of the

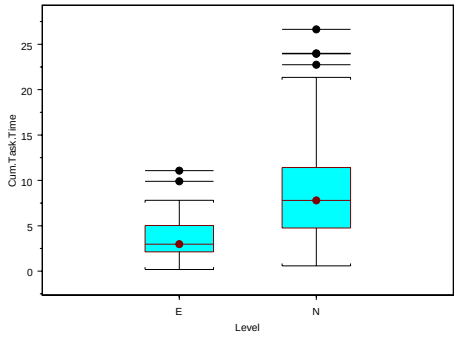


Figure 11: Time taken (per spreadsheet).

subjects in Group E and found that there is no significant difference (box plot shown in Figure 12). Our expectation was that the experts would need to inspect fewer cells to be able to infer the template for a given spreadsheet. In the experiment setting, however, we found no significant difference in the number of expected cells. This fact might be an indicator that the experts were extremely cautious while carrying out their assigned tasks.

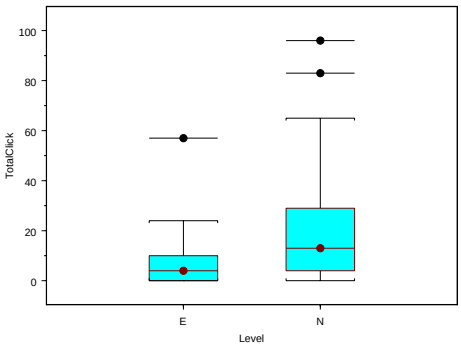


Figure 12: Sheet inspection profile (per spreadsheet).

To verify if the subjects found it more difficult to infer templates for bigger spreadsheets than for smaller ones, we ran regression tests comparing their scores on the tasks against the size of the spreadsheets. We found no significant correlation between the scores obtained on the tasks and the size of the spreadsheets for the two groups. This result is not too surprising since the size of a spreadsheet is not a particularly good measure of its complexity. More reliable measures might be the number and complexity of the formulas in the spreadsheet. Moreover, very simple templates can be used to generate very large spreadsheets. In such situations, humans might be able to infer the templates very accurately through visual inspection of the spreadsheet.

We see from the data that the templates automatically generated by the system score significantly higher than the subjects in groups E and N. If the time taken by Excel to load each spreadsheet is ignored, the system takes less than a second per spreadsheet to automatically infer the template. The mean time taken by the subjects in groups E and N to infer a template are 3.8 minutes and 8.9 minutes, respectively.

We did not impose any time limit on the subjects for the

completion of the tasks. Two of the subjects from Group N stopped after an hour because of prior commitments and one stopped citing fatigue.

5. RELATED WORK

Some researchers have focussed their efforts on guidelines for designing better spreadsheets so errors can be avoided to some extent [28, 36, 18, 24, 26]. Such techniques are difficult to enforce and involve costs of training the user.

Most of the research that has been done in the area of spreadsheets has been targeted at removing errors from spreadsheets once they have been created. Following traditional Software Engineering approaches, some researchers have recommended code inspection for detection and removal of errors from spreadsheets [22, 31, 21]. However, such approaches cannot give any guarantees about the correctness of the spreadsheet once the inspection has been carried out. Code inspection of larger spreadsheets might prove tedious, error prone, and prohibitively expensive in terms of the effort required.

The "What You See Is What You Test" (WYSIWYT) testing methodology for spreadsheets has been developed and studied within the Forms/3 framework [29]. User studies have shown that it is very effective in helping detect errors in spreadsheets. User studies have also been conducted to evaluate fault localization strategies in the WYSIWYT system [25]. These studies have demonstrated that end users are more likely to use a feature if the benefits are made apparent.

We have developed a goal-directed debugger for spreadsheets that allows users to mark cells with incorrect values and then specify the expected value in the cell. The system then generates a list of suggested changes that would result in the expected value being computed in the marked cell. The generated suggestions are ranked on the basis of heuristics we have developed and the list is presented to the user. The user can then simply pick a suggestion and apply it to the spreadsheet [2].

Automatic consistency-checking approaches have also been explored to detect errors in spreadsheets. Most of the systems require the user to annotate the spreadsheet cells with extra information [4, 5, 7, 9, 13]. We have developed a system, called UCheck, that automatically infers the labels within the spreadsheet and uses this information to carry out consistency checking [1], thereby requiring minimal effort from the user.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a tool to infer the templates from spreadsheets. This tool is an essential component of the ViTSL/Gencel architecture because it enables a smooth migration from Excel to Gencel, which is indispensable for a widespread adoption of the ViTSL/Gencel approach.

We have demonstrated that the tool works remarkably well compared to human subjects. The templates that were automatically inferred by the system have been shown to be significantly better than those inferred by the human subjects when rated by experts. In future work we will extend the functionality of the template inference tool and also perform further user study.

As discussed in Section 3.3, the system is tolerant of deviations from an exact match. This task might get more complicated when the spreadsheet has logical errors in it. In such situations, there might be more than one "correct" template for the spreadsheet. The current version of the system only infers one template. We plan to extend the system so that it will infer all possible templates for a given spreadsheet, rank them on the basis of one or more heuristic, and present them to the user. From the list, the user can pick the template they think is the most adequate, and the system can then report the potential errors and other violations (if any) within the spreadsheet that prevent the template from being an exact match. Such an extended system can also be employed to detect errors in spreadsheets.

Even though the subjects in Group E have considerable experience with programming and spreadsheets, they are not domain experts as far as the spreadsheets are concerned. It would be informative to repeat the study in specific spreadsheet domains with people who work in the respective domains as subjects. We also plan to carry out studies aimed at finding out how factors like size of the spreadsheets, number and types of errors, and complexity and number of formulas impact the system and user performance.

Acknowledgements

We express our gratitude to Curtis Cook, Simone Stumpf, Deling Ren, Zhe Fu, Mansour Al-Mutairi, Steve Kollmansberger, Cory Kissinger, Joey Lawrence, Laura Beckwith, and the students of CS 361 of Oregon State University for helping with the study.

7. REFERENCES

[1] R. Abraham and M. Erwig. Header and Unit Inference for Spreadsheets Through Spatial Analyses. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, pages 165–172, 2004.

[2] R. Abraham and M. Erwig. Goal-Directed Debugging of Spreadsheets. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, 2005. To appear.

[3] R. Abraham, M. Erwig, S. Kollmansberger, and E. Seifert. Visual Specifications of Correct Spreadsheets. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, 2005. To appear.

[4] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi. A Type System for Statically Detecting Spreadsheet Errors. In *18th IEEE Int. Conf. on Automated Software Engineering*, pages 174–183, 2003.

[5] T. Antoniu, P. A. Steckler, S. Krishnamurthi, E. Neuwirth, and M. Felleisen. Validating the Unit Correctness of Spreadsheet Programs. In *26th IEEE Int. Conf. on Software Engineering*, pages 439–448, 2004.

[6] P. S. Brown and J. D. Gould. An Experimental Study of People Creating Spreadsheets. *ACM Transactions on Office Information Systems*, 5(3):258–272, 1987.

[7] M. M. Burnett, C. Cook, J. Summet, G. Rothermel, and C. Wallace. End-User Software Engineering with Assertions. In *25th IEEE Int. Conf. on Software Engineering*, pages 93–103, 2003.

[8] M. M. Burnett, A. Sheretov, B. Ren, and G. Rothermel. Testing Homogeneous Spreadsheet Grids with the "What You See Is What You Test" Methodology. *IEEE Transactions on Software Engineering*, 29(6):576–594, 2002.

[9] M. J. Coblenz, A .J. Ko, and B. A. Myers. Using Objects of Measurement to Detect Spreadsheet Errors. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, 2005. To appear.

[10] G. Engels and M. Erwig. ClassSheets: Automatic Generation of Spreadsheet Applications from Object-Oriented Specifications. In *20th IEEE/ACM Int. Conf. on Automated Software Engineering*, 2005. To appear.

[11] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger. Automatic Generation and Maintenance of Correct Spreadsheets. In *27th IEEE Int. Conf. on Software Engineering*, pages 136–145, 2005.

[12] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger. Gencel — A Program Generator for Correct Spreadsheets. *Journal of Functional Programming*, 2005. To appear.

[13] M. Erwig and M. M. Burnett. Adding Apples and Oranges. In *4th Int. Symp. on Practical Aspects of Declarative Languages*, LNCS 2257, pages 173–191, 2002.

[14] M. Fisher and G. Rothermel. The EUSES Spreadsheet Corpus: A Shared Resource for Supporting Experimentation with Spreadsheet Dependability Mechanism. In *1st Workshop on End-User Software Engineering*, pages 47–51, 2005.

[15] M. Fisher II, M. Cao, G. Rothermel, C. Cook, and M. M. Burnett. Automated Test Case Generation for Spreadsheets. In *24th IEEE Int. Conf. on Software Engineering*, pages 141–151, 2002.

[16] T. R. G. Green and M. Petre. Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages and Computing*, 7(2):131–174, 1996.

[17] R. L. Hayen and R. M. Peters. How to Ensure Spreadsheet Integrity. *Management Accounting*, 60(9):30–33, 1989.

[18] T. Isakowitz, S. Schocken, and H. C. Lucas, Jr. Toward a Logical/Physical Theory of Spreadsheet Modelling. *ACM Transactions on Information Systems*, 13(1):1–37, 1995.

[19] J. F. Lerch, M. M. Mantei, and J. R. Olson. Skilled Financial Planning: The Cost of Translating Ideas Into Action. *ACM Conf. on Human Factors in Computing Systems*, pages 121–126, 1989.

[20] C. Lewis and G. M. Olson. Can Principles of Cognition Lower the Barriers to Programming? In *2nd Workshop on Empirical Studies of Programmers*, pages 248–263, 1987.

[21] R. Mittermeir and M. Clermont. Finding High-Level Structures in Spreadsheet Programs. In *9th Working Conference on Reverse Engineering*, pages 221–232, 2002.

[22] R. R. Panko. Applying Code Inspection to Spreadsheet Testing. *Journal of Management Information Systems*, 16(2):159–176, 1999.

[23] R. R. Panko and R. P. Halverson, Jr. Spreadsheets on Trial: A Survey of Research on Spreadsheet Risks. In *29th Hawaii Int. Conf. on System Sciences*, 1996.

[24] S. G. Powell and K. R. Baker. *The Art of Modeling with Spreadsheets: Management Science, Spreadsheet Engineering, and Modeling Craft*. Wiley, 2004.

[25] S. Prabhakarao, C. Cook, J. Ruthruff, E. Creswick, M. Main, M. Durham, and M. Burnett. Strategies and Behaviors of End-User Programmers with Interactive Fault Localization. In *IEEE Int. Symp. on Human-Centric Computing Languages and Environments*, pages 203–210, 2003.

[26] K. Rajalingham, D. Chadwick, B. Knight, and D. Edwards. Quality Control in Spreadsheets: A Software Engineering-Based Approach to Spreadsheet Development. In *33rd Hawaii Int. Conf. on System Sciences*, pages 1–9, 2000.

[27] K. Rajalingham, D. R. Chadwick, and B. Knight. Classification of Spreadsheet Errors. *Symp. of the European Spreadsheet Risks Interest Group (EuSpRIG)*, 2001.

[28] B. Ronen, M. A. Palley, and H. C. Lucas, Jr. Spreadsheet Analysis and Design. *Communications of the ACM*, 32(1):84–93, 1989.

[29] G. Rothermel, M. M. Burnett, L. Li, C. DuPuis, and A. Sheretov. A Methodology for Testing Spreadsheets. *ACM Transactions on Software Engineering and Methodology*, pages 110–147, 2001.

[30] P. Saariluoma and J. Sajaniemi. Extracting Implicit Tree Structures in Spreadsheet Calculation. *Ergonomics*, 34(8):1027–1046, 1991.

[31] J. Sajaniemi. Modeling Spreadsheet Audit: A Rigorous Approach to Automatic Visualization. *Journal of Visual Languages and Computing*, 11:49–82, 2000.

[32] C. Scaffidi, M. Shaw, and B. Myers. Estimating the Numbers of End Users and End User Programmers. In *IEEE Symp. on Visual Languages and Human-Centric Computing*, 2005. To appear.

[33] Thompson SH. Teo and Margaret Tan. Quantitative and qualitative errors in spreadsheet development. *Proceedings of the Thirtieth Hawaii International Conference on System Sciences*, 3:149–156, 1997.

[34] U.S. Department of Education. Audit of the Colorado Student Loan Program's Establishment and Use of Federal and Operating Funds for the Federal Family Education Loan Program, July 2003. Report ED-OIG/A07-C0009.

[35] U.S. Department of Health and Human Services. Review of Medicare Bad Debts at Pitt County Memorial Hospital, January 2003. Report A-04-02-02016.

[36] A. G. Yoder and D. L. Cohn. Real Spreadsheets for Real Programmers. In *Int. Conf. on Computer Languages*, pages 20–30, 1994.