Visual Specifications of Correct Spreadsheets[*]

Robin Abraham Martin Erwig Steve Kollmansberger Ethan Seifert

School of EECS
Oregon State University
[abraharo, erwig, kollmast, seiferet]@eecs.oregonstate.edu

January 24, 2005

Abstract

We introduce a visual specification language for spreadsheets that allows the definition of spreadsheet templates. These templates are used by a spreadsheet generator to create Excel spreadsheets that are provably free from a large class of errors, such as reference, omission, and type errors.

We demonstrate how spreadsheets can be categorized into classes of spreadsheets that are related through well defined update operations. For each class we can identify a template that can be expressed in our visual language. We present a formal definition of the visual language for templates and describe the process of generating spreadsheets from templates. In addition, we present an editor for these templates and analyze the editor using the Cognitive Dimensions framework.

Keywords: Visual Language, Spreadsheet, Specification, End-User Software Engineering, Program Generation, Software Correctness

1 Introduction

Visualization is often used to analyze and solve software engineering tasks [19, 25]. Especially in large software engineering projects, visualization provides a means to display varying levels of detail in easily understood formats. This helps to keep software engineering projects manageable and consistent. UML is the standard for visualizing and modeling object-oriented software projects [1]. UML diagrams and other visualizations do not exist in a vacuum—there are numerous tools which allow visualizations to be manipulated [23]. Some systems, such as Fujaba [15], even convert diagrams automatically into framework code. Others, such as VIATRA [10], allow diagrams to be checked for correctness and adherence to specification.

While a spreadsheet environment, such as Excel, is a software program, the spreadsheets users create in it may also be considered as software programs. If a program is an executable specification

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

of computations, then a spreadsheet certainly is a program. Spreadsheets also have input (data cells), processing (formulas) and output ("bottom-line" cells).

Spreadsheets are also widely used: It is estimated that each year tens of millions of professionals and managers create hundreds of millions of spreadsheets [21]. These spreadsheet users are performing programming tasks everyday, even if they are not consciously aware of it. In many cases, these spreadsheets contain errors. Numerous studies have shown that existing spreadsheets contain errors at an alarmingly high rate [5, 20, 21]. Some studies even report that 90% or more of real-world spreadsheets contain errors [24].

These errors can cause significant real-world losses. In one case, a spreadsheet to calculate a bid was incorrect, which caused TransAlta Corp. to underbid a contract and, bound to stand by the terms of the contract, to take a $2.4 million loss [11]. In another spreadsheet error, two Shurgard executives were overpaid by $700,000 each [28]. While the executives surely enjoyed that particular error, these and similar errors can be financially devastating to companies and individuals.

Since spreadsheets are themselves programs, spreadsheet design, modification and use can also be considered a software engineering task. The importance of applying software engineering methodologies to spreadsheets lies in the high error rates and potential (and occasionally actual) million dollar impacts of spreadsheet errors. In many cases, these errors are a result of the ad-hoc, undocumented operation of spreadsheets where a sheet may be created by one person, then used and modified by another who does not fully understand its functionality. Since spreadsheet systems offer no abstractions and do not even separate data from computation, attempts to reuse spreadsheets can easily result in errors.

This paper introduces 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 introduced in this paper, ViTSL (an acronym for **vi**sual **t**emplate **s**pecification **l**anguage), provides a method for modeling the template of a spreadsheet and the ways it can evolve.

ViTSL specifications are constructed with an editor and are loaded into Gencel [13], which is an Excel extension providing an environment where update operations behave dependent on the given specification. With the introduction of abstraction provided by ViTSL, spreadsheets become not only safer, but also more re-usable. The Gencel environment manages the evolution of a spreadsheet from a ViTSL specification. 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. Since the computational function and structure of a spreadsheet is abstracted from the data, a spreadsheet modeled with ViTSL can be confidently reused. Specifications also act as a form of documentation to describe the functionality of the spreadsheet without reference to particular instances.

The architecture of the ViTSL/Gencel system is depicted in Figure 1 and reflects the observation that the creation of spreadsheets can be conceptually split into two phases: First, a computational schema, or template, is developed, which defines headers, data cells, and computations. Second, this schema/template is filled with data, which includes the possible insertion and deletion of rows and columns. The ViTSL editor serves the first task, whereas the Gencel system is responsible for the second. In Excel these two different phases are not separated, which is the source of many errors in Excel spreadsheets as will be illustrated in Section 2.

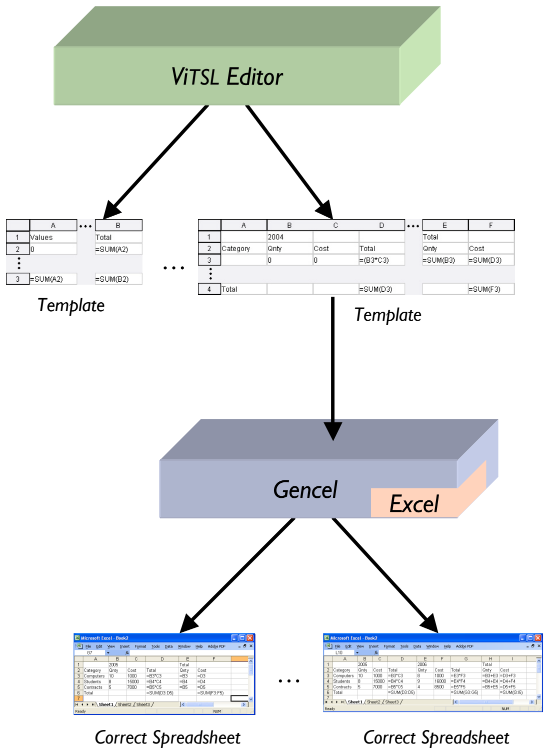


Figure 1: Architecture of the ViTSL/Gencel system

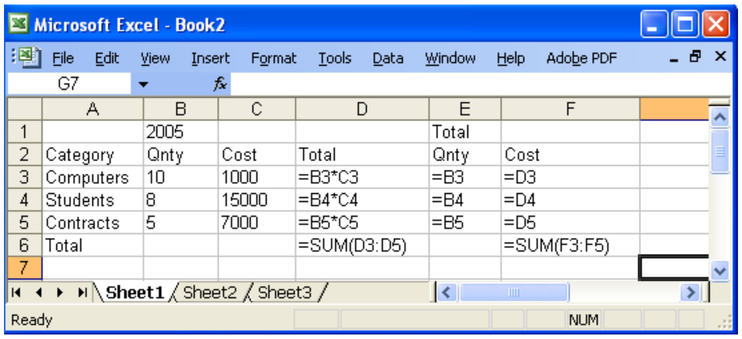
The benefit of the chosen two-level approach to creating spreadsheets is that users can still apply update operations to their spreadsheets (such as row/column insertions or deletions), but only those that keep the spreadsheet within the specified evolution and do not introduce any reference, type, or omission errors.

Just as software engineering practices allow separation of modeling and implementation duties as well as standards and quality control, the ViTSL specification language provides these benefits to spreadsheets. An organization could hire domain experts to construct ViTSL specifications for the domains they wish to operate in. These specifications can be confidently used by other employees without a precise understanding of all aspects of the how the spreadsheets work. Furthermore, by enforcing the structure and formulas, Gencel acts as an automatic quality control system, ensuring that spreadsheets stay within the bounds prescribed by the domain experts and approved by management.

In Section 2 we first present an example of how Gencel can protect a spreadsheet from errors and how ViTSL can be used to design a specification to that end. In Section 3, we describe a formal model for ViTSL specifications. We briefly discuss the process of spreadsheet generation from specifications in Section 4. In Section 5, we show the design of the ViTSL editor and describe how it can be used to create specifications. An analysis of the ViTSL editor using the Cognitive Dimensions framework [16, 4] follows in Section 6. Finally, we discuss related work in Section 7 and present conclusions in Section 8.

2 Editing Spreadsheets

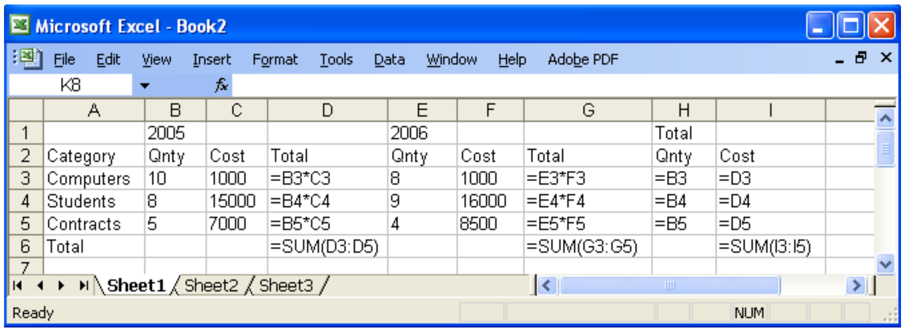
Consider the Excel spreadsheet shown in Figure 2.


Microsoft Excel - Book2
File Edit View Insert Format Tools Data Window Help Adobe PDF
G7
Sheet1 Sheet2 Sheet3
Ready
NUM

	A	B	C	D	E	F
1		2005			Total	
2	Category	Qnty	Cost	Total	Qnty	Cost
3	Computers	10	1000	=B3*C3	=B3	=D3
4	Students	8	15000	=B4*C4	=B4	=D4
5	Contracts	5	7000	=B5*C5	=B5	=D5
6	Total			=SUM(D3:D5)		=SUM(F3:F5)
7						

Figure 2: Excel budget spreadsheet

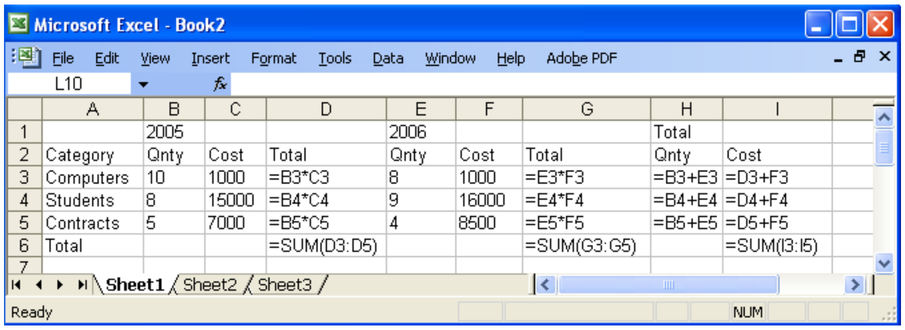
This spreadsheet allows the user to maintain quantity and costs for a variety of items within a single year. Note that the individual year consists of three actual Excel columns. If the user wanted to add another year to the spreadsheet, a series of operations would be required. First, three new columns would need to be inserted and the appropriate headers copied. Next, the formula for the total of that year could be copied from another year. At this point, the table looks correct and is shown in Figure 3.


Microsoft Excel - Book2
File Edit View Insert Format Tools Data Window Help Adobe PDF
K8
Sheet1 Sheet2 Sheet3
Ready
NUM

	A	B	C	D	E	F	G	H	I
1		2005			2006			Total	
2	Category	Qnty	Cost	Total	Qnty	Cost	Total	Qnty	Cost
3	Computers	10	1000	=B3*C3	8	1000	=E3*F3	=B3	=D3
4	Students	8	15000	=B4*C4	9	16000	=E4*F4	=B4	=D4
5	Contracts	5	7000	=B5*C5	4	8500	=E5*F5	=B5	=D5
6	Total			=SUM(D3:D5)			=SUM(G3:G5)		=SUM(I3:I5)
7									

Figure 3: Excel budget spreadsheet, updated

However, critical updates have not yet been made, and could easily be overlooked: In addition to inserting and filling the new columns, the user must also remember to update the grand totals at the far right. In this case, no copy/paste operation will suffice—references to the new year's quantity and total cost must be added manually. This error-prone process must be repeated for each year the user wishes to add. Mistakes in this operation will cause the total values to be incorrect, but no feedback is given by Excel to indicate a mistake may have been made. In addition, if the updates are forgotten in some cells, they will still appear to have reasonable values, and Excel will grant no warning to the user. A corrected version of the spreadsheet is shown in Figure 4.


Microsoft Excel - Book2
File Edit View Insert Format Tools Data Window Help Adobe PDF
L10
Sheet1 Sheet2 Sheet3
Ready
NUM

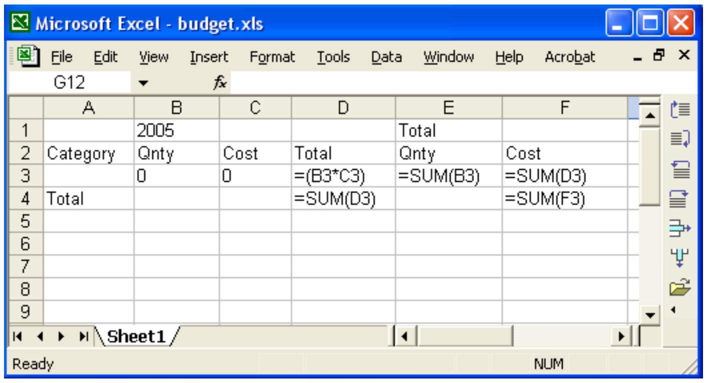
	A	B	C	D	E	F	G	H	I
1		2005			2006			Total	
2	Category	Qnty	Cost	Total	Qnty	Cost	Total	Qnty	Cost
3	Computers	10	1000	=B3*C3	8	1000	=E3*F3	=B3+E3	=D3+F3
4	Students	8	15000	=B4*C4	9	16000	=E4*F4	=B4+E4	=D4+F4
5	Contracts	5	7000	=B5*C5	4	8500	=E5*F5	=B5+E5	=D5+F5
6	Total			=SUM(D3:D5)			=SUM(G3:G5)		=SUM(I3:I5)
7									

Figure 4: Excel budget spreadsheet, corrected

Likewise, if the user wishes to add additional items to their spreadsheet, they may insert rows. However, once the rows have been inserted, the formulas for each year must be copied down from the previous year. Moreover, the formulas for the grand totals must also be copied. In case a row is added at the bottom, the aggregation formulas along the bottom must all be updated, all by hand. Again, if the user fails to update a formula or inadvertantly selects the incorrect cell, Excel will happily present them with the erroneous value. Any single forgotten or incorrect update will likely lead to an error on the bottom line, which could be substantial in magnitude and difficult to detect, isolate, and fix.

As the user progresses with this spreadsheet there are basically three updates that the user will perform: add another item (row), add another year (three columns), or update quantities and header labels. The user may also choose to delete years or categories, although this is probably less common. In addition, each of these operations can be broken down into a fixed set of necessary steps. In this way, the initial spreadsheet with one year, a spreadsheet with two years, and a spreadsheet with twenty years are all related. In this sense, the spreadsheets from Figures 2 and 4 can be thought of as deriving from the one shown in Figure 5.

From this template, any number of spreadsheets may be derived using the operations provided by Gencel [13]. These operations, which consist of row or column insert, value update, and row or column delete, are specialized for the template to ensure that updates occur correctly with all necessary changes. For example, if the user presses the insert column button (see right panel in Figure 5) when the cursor is within a year group, three new columns representing a new year will be inserted at once and all formulas will be correctly updated instantly. The Gencel system provides these specialized updates to ensure correctness. For example, a spreadsheet with twenty years and forty items could be reached using the specialized, correct update operations. Since a template is generic with respect to the actual categories, years 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.

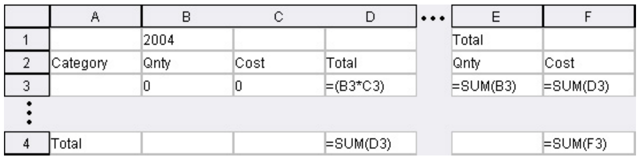

Microsoft Excel - budget.xls
File Edit View Insert Format Tools Data Window Help Acrobat
G12
Sheet1
Ready
NUM

	A	B	C	D	E	F
1		2005			Total	
2	Category	Qnty	Cost	Total	Qnty	Cost
3		0	0	=(B3*C3)	=SUM(B3)	=SUM(D3)
4	Total			=SUM(D3)		=SUM(F3)
5						
6						
7						
8						
9						

Figure 5: Gencel budget spreadsheet

From the template shown in Figure 5 it is not clear which columns and rows are fixed and which are expandable. However, it is possible to represent these templates in some more abstract way than simply a Gencel spreadsheet. It is possible to analyze the spreadsheet and abstract the basic building blocks which make it up—a three column repeating group which forms a year, an aggregation formula (for example, the grand total of quantity) which sums all the left-most cells in each year, as well as headers and values. 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 formal visual specification language for these templates.

The ViTSL template for the above Gencel spreadsheet looks as follows.



	A	B	C	D	⋯	E	F
1		2004				Total	
2	Category	Qnty	Cost	Total		Qnty	Cost
3		0	0	=(B3*C3)		=SUM(B3)	=SUM(D3)
⋮							
4	Total			=SUM(D3)			=SUM(F3)

This specification describes how the three columns repeat and how the total formulas on the right sum up the quantity and total cost. In particular, we group the three columns for a year together and make them horizontally repeat, showing that any number of years are allowed. The relative references, B3 and D3, in the grand total columns refer three cells to the left for Qnty and two cells to the left for Cost, respectively. As these cells are repeated (for additional years) the formulas for the grand totals are automatically updated by Gencel to include the Qnty and Cost, respectively, for each year. Similarly, the third row is specified as vertically repeating.

In this way, spreadsheets can be derived mechanically, ensuring that formulas are always correct and that the correct types of data are always used. This provides a substantial amount of safety against errors which may occur in a manual update process, especially as a spreadsheet becomes large with many references.

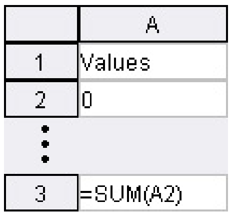
3 Visual Spreadsheet Specifications

ViTSL is a visual language to formally describe spreadsheet structure and evolution. A ViTSL figure is a template for a class of tables that all share essentially the same structure.

The structure of tables is determined by different kinds of cells and their relationships. For example, the cells of a table can be distinguished according to their content into header, data, and computation cells. Moreover, some rows or columns of a table are fixed, like header and footer rows and columns, while other rows and columns are duplicated if new data is to be added.

ViTSL offers constructs to define a table as a horizontal sequence of fixed and extendable columns where a column is constructed as a vertical sequence of fixed and extendable blocks, which are rectangular collections of cells containing values and formulas.

Consider, for example, all tables that consist of a plain column of numbers with a header at the top and a summation formula at the bottom. A corresponding table template can be specified by the following ViTSL expression.



	A
1	Values
2	0
⋮	
3	=SUM(A2)

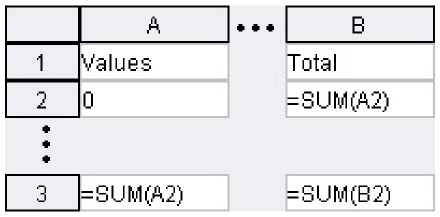
The specification consists of three elements: two cells, the header containing the label and the footer containing the summation formula, and a vertically expandable group (also called *vex group* for short) that consists of a single cell containing the value 0. The argument of the summation formula, A2, is shown as an absolute address in the ViTSL specification. However, it actually represents a relative address that is translated and expanded by Gencel into a range of addresses according to the number of times the second row is expanded.

The above template describes a class of tables that all consist of one column with the shown (or another) header and footer and that have one or more number cells in between.

ViTSL offers the following visual elements for describing templates:

- *Cells*, represented by rectangles and containing data and formulas.
- *References*, represented by cell addresses.
- *Vex groups*, represented by vertical dots that indicate the possible expansion of a (group of) cell(s) in vertical direction.
- *Hex groups*, represented by horizontal dots that indicate the possible expansion of one or more columns in horizontal direction.

An example for a horizontally expandable group (*hex group*) is given in the following summation table.



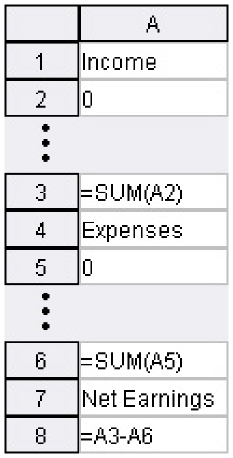
	A	⋯	B
1	Values		Total
2	0		=SUM(A2)
⋮			
3	=SUM(A2)		=SUM(B2)

This template is obtained by taking the previously shown summation column, making it horizontally expandable, and joining it by a column that also contains a header and a summation footer, but whose vex group contains a summation formula whose argument references the number cell of the hex group.

The hex group in the last example illustrates that expandable groups may consist of groups of cells and not just single cells. Moreover, one column can also contain multiple vex groups. Similarly, a table can contain multiple hex groups. However, vex groups and hex groups cannot be arbitrarily nested. The only possible nesting is indicated by the example: hex groups may contain vex groups.

In addition, a couple of structural constraints are needed to ensure that a reasonable definition for the update operations of Gencel exists. For example, all columns in a table have to *align* vertically. To explain the idea of alignment, consider a column as a sequence of fixed and expandable blocks (cell groups), say $c = [b_1, \ldots, b_k]$. Now c matches another column $c' = [b'_1, \ldots, b'_k]$ only if (a) b_i has the same height as b'_i and (b) b_i is an expandable group iff b'_i is. This constraint ensures that all existing vex groups are horizontally aligned, which allows the insert-row command to be defined to insert a number of rows according to the common height of the vex groups. Similarly, we require that all blocks in a column have the same width. For columns in hex groups, this constraint ensures that the insert-column command can be defined to create a number of columns according to the common width of the blocks of the hex group.

An example for a table containing multiple vex groups is the following accounting sheet.



	A
1	Income
2	0
⋮	
3	=SUM(A2)
4	Expenses
5	0
⋮	
6	=SUM(A5)
7	Net Earnings
8	=A3-A6

The formula `A3-A6` computes the difference between the two summation cells.

The relative references used in templates are very expressive: First, unlike absolute addresses, relative references are compositional, that is, they need not be adjusted when cells or blocks are composed with other blocks. Second, depending on their origin and target, relative references can express single-cell addresses as well as ranges. For example, the references from the summation formulas point into a vex group and refer to all the cells that will be generated within that group, in contrast to the references A6 and A3, which point to nonexpandable cells and refer always to single values.

In the following we provide a formal definition of the ViTSL syntax. A *template* t is given by a horizontal composition ($|$) of fixed (c) or expandable ($c^{\rightarrow}$) columns, where a column is given by a vertical composition ($\hat{}$) of fixed (b) or expandable ($b^{\downarrow}$) blocks. A block is given by a composition of formulas (f). Formulas consist of basic values (ϕ), references (ρ), and expressions that are built by applying functions to a varying number of arguments given by formulas ($\phi(f, \ldots, f)$). Functions can generally be applied to an arbitrary number of arguments of the same type, like addition SUM and multiplication PROD.

Formally, references are given by pairs of integers and represent relative references in the form of offsets. We use the following abbreviations for cell offsets: $\ell = (-1, 0)$, $r = (1, 0)$, $u = (0, -1)$, and $d = (0, 1)$. We sometimes use sequences of abbreviated offsets to represent larger offsets, for example, $\ell\,\ell = \ell^2 = (-2, 0)$. The syntax of templates is summarized in Figure 6.

$f \in \mathit{Fml}$	$::=$	$\phi \mid \rho \mid \phi(f, \ldots, f)$	(*formulas*)
$b \in \mathit{Block}$	$::=$	$f \mid b \,\vert\, b \mid b\hat{}b$	(*blocks, tables*)
$c \in \mathit{Col}$	$::=$	$b \mid b^{\downarrow} \mid c\hat{}c$	(*columns*)
$t \in \mathit{Template}$	$::=$	$c \mid c^{\rightarrow} \mid t \,\vert\, t$	(*templates*)

Figure 6: Abstract Syntax of ViTSL.

The constructs correspond directly to the visual notation. As an example, consider the summation column, which was shown at the beginning of this section. This column is represented by the following template.

$$\mathsf{Values}\hat{}0^{\downarrow}\hat{}\mathsf{SUM}(u)$$

The summation table is represented by the following expression.

$$(\mathsf{Values}\hat{}0^{\downarrow}\hat{}\mathsf{SUM}(u))^{\rightarrow} \mid \mathsf{Total}\hat{}\mathsf{SUM}(\ell)^{\downarrow}\hat{}\mathsf{SUM}(u)$$

The abstract syntax representation of ViTSL templates is an important prerequisite to facilitate a precise formal definition of the spreadsheet generation process and the derivation of safety properties.

4 Automatic Generation of Spreadsheets

We present here only an informal description of the spreadsheet generation process in this paper. Details of how tables and updates operations are created from specifications can be found in [13] and in [12].

The definition of update operations relies on an intermediate structure, called a *template instance*, which is a slight generalization of a table specification in which vertical and horizontal repeating groups are marked with numbers instead of arrows to represent the number of expansions of the corresponding groups, that is, $b^{|k}$ instead of $b^{\downarrow}$ and $c^{\underline{k}}$ instead of $c^{\rightarrow}$. These group sizes are, in particular, needed to correctly generate references in formulas. In the initial template instance all $\rightarrow$ and $\downarrow$ exponents are replaced by ones. Then each application of an insert-column command to a hex group increases its exponent by one, whereas each application of an insert-row command increases the exponents of all vex groups in one row by one.

Since a template instance contains all the required information to generate all formulas with references for a table, update operations, such as insert column, simply create an updated template instance by increasing the repeating group index and then derive the changed formulas from the new instance. All values from the old table are copied, whereas values for possibly inserted rows or columns are taken from the template.

The translation of a template instance into a spreadsheet works by recursively traversing the template instance. During this traversal, four different kinds of actions are performed. (The formal definition of the translation is indicated by equations of the form $T(t) = t'$; the actual definition is a bit more involved since it maintains information about the position of a block or column within the template to facilitate the proper translation of references.)

1. Constants and functions are copied unchanged (that is, $T(\phi) = \phi$).

2. Horizontally and vertically joined blocks and columns are translated separately, and the results are joined as in the template instance (that is, $T(t_1 \mid t_2) = T(t_1) \mid T(t_2)$ and $T(c_1 \,\hat{}\, c_2) = T(c_1) \,\hat{}\, T(c_2)$).

3. A horizontal (vertical) repeating group that is marked with a number k will be translated k times, and all results will be horizontally (vertically) joined together (that is, $T(c^{\underline{k}}) = T(c) \mid T(c) \mid \ldots \mid T(c)$ and $T(b^{|k}) = T(b) \,\hat{}\, T(b) \,\hat{}\, \ldots \,\hat{}\, T(b)$).

4. Relative references will be translated into absolute references or ranges, depending on the target cell of the reference. The absolute addresses are computed based on the current position of the cell that contains the reference, the relative offset, and possible exponents of vex and hex groups that are "crossed" by the offset.

The translation of relative into absolute references requires the consideration of many different cases. The main idea is described in the following. Suppose (x, y) is the absolute address of a cell that contains a formula with a relative reference $\rho = (i, j)$. If (i, j) points to a cell that is contained in a horizontally and/or vertically repeating group, the reference might denote a horizontal, vertical, or two-dimensional range, but only if the current cell (x, y) that contains the reference (i, j) is *not* contained in a repeating group that is expanded in parallel with the referenced one. This information can be obtained for the horizontal dimension by checking whether $x + i$ lies outside the horizontal range of a possible repeating group containing (x, y), in which case the cells are expanded independently of one another. Similarly, only if $y + j$ lies outside the vertical range of the group containing the current cell, it is not expanded together with the group containing (i, j).

If the relative address is to be translated into a range, this range is computed by selecting from the already expanded table area (see item 3. above) that corresponds to the repeating group containing the referenced cell $(x + i, y + j)$ all addresses (x', y') whose relative offset from the beginning of the repeating group is the same as the offset of the referenced cell from the first repeating group. The ViTSL budget sheet specification presented in Section 2 illustrates this case where the generated range for the summation formula under Qnty consists only of two cells, which both start at the beginning of each repeating-group instance, that is, columns B and E. A similar example is the summation formula under Cost.

Apart from the translation of references the table generation process is rather straightforward, and the described process yields Gencel tables like the one shown in Figure 4 for the template instance derived from the template from Figure 5.

We have designed a type system for ViTSL that guarantees a very high level of spreadsheet correctness, namely, any spreadsheet that is created with Gencel from a ViTSL template will never contain any omission, reference, or type errors. This is possible because type- and reference-correct ViTSL templates will be translated into specialized update operations, such as insert row, used by Gencel to modify spreadsheets. For details of the type system we refer to [13, 12].

We believe that this safety property can have a huge positive impact on the correctness of spreadsheets, because once a particular ViTSL template has been created that is appropriate for a particular application, all spreadsheets that will ever be instantiated from it will be always correct.

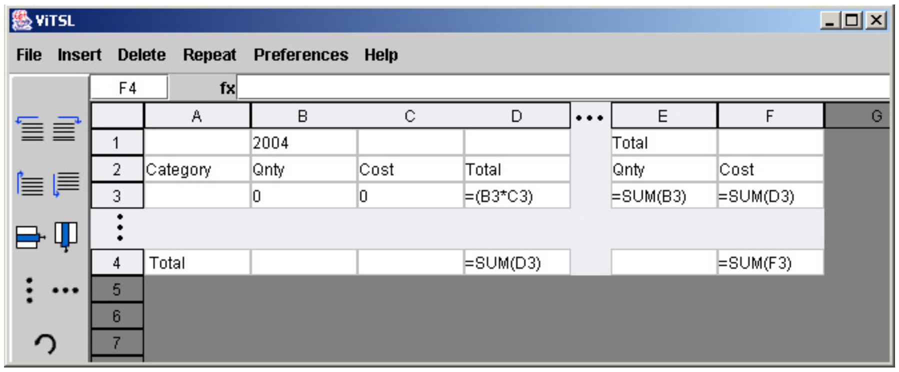
5 The ViTSL Editor

The ViTSL editor is a tool used to create ViTSL specifications, that is, templates. To create these specifications the ViTSL editor uses a GUI which has intentional similarities to Microsoft Excel. This design goal intends to make a user familiar with Excel "feel at home" when using this editor. When working with the editor a user will create a ViTSL specification much like they would create a spreadsheet. Both ViTSL sheets and spreadsheets are expanded by the insertion of rows and columns, describing values for cells, and the application of operations on those cells.

Consider the following scenario. Suppose a user is interested in creating the template for budget spreadsheets discussed in some detail in Section 2. Creating the basic structure for one year will require the user to insert four rows and three columns. The user will also need to describe the header cells and provide definitions of the formula cells. Defining a formula cell starts by selecting the target cell and typing = into the cell and then providing the formula and the cells it references. A user can reference a cell either by typing in the cell address or by clicking on the target cell when in a formula, just like in Excel.

In order to create a hex group the user selects the columns that will be expanded together (Qnty, Cost, Total) and chooses to make a hex group. This hex group is a representation of each year. The ⋯ button is then pressed. The editor provides feedback to let the user know the effect of the operation. First, because the hex group consists of more than one column, the header division between each member of the expanding group will be removed. Also, ellipses will be placed in the header trailing each repeating group. Finally, the user will need to insert two more columns and create formulas referencing the newly created hex group to get total cost and total quantity of all possible years.

To finish the budget sheet specification the user will select the row 3 and will choose to make it a vex group by pressing the ⋮ button. This vex group represents a category and its costs within a year. The user will have to insert one more row and create formula references to total cost for



	A	B	C	D	E	F	G
1		2004			Total		
2	Category	Qnty	Cost	Total	Qnty	Cost	
3		0	0	=(B3*C3)	=SUM(B3)	=SUM(D3)	
4	Total			=SUM(D3)		=SUM(F3)	
5							
6							
7							

Figure 7: Budget template created in the ViTSL editor

each year and another formula to reference total cost for all years. This will complete the budget specification; the final result is pictured in Figure 7.

6 Analysis of the ViTSL Design

In this section we describe how our choices in designing the system have been guided by the *Cognitive Dimensions of Notations* framework [16, 4]. For the sake of brevity, we discuss, with suitable examples, the ones that are of primary interest in the context of the ViTSL editor.

Viscosity The viscosity of a system is an indicator of how much effort is required to accomplish user goals using the system. Two kinds of viscosity might be present in a system—*repetition viscosity* which refers to the system forcing the user to carry out many actions of the same kind individually (for example, replacing all occurrences of one string by another), and *knock-on viscosity* which refers to the system requiring the user to perform many actions to restore consistency after the user has performed some action to achieve a certain goal (for example, variable renaming in a program using a replace-all feature of a simple text editor).

To facilitate easy editing of specifications, we have been especially careful in providing facilities to reverse any action performed by the user. The insert-row and insert-column operations have corresponding delete-row and delete-column operations. The grouping operations that specify if cells are in horizontally or vertically repeating groups have corresponding ungroup operations. The system supports the deletion of repeating or non-repeating rows or columns. We also provide the facility to undo an unlimited number of operations.

Viscosity can be lowered, in general, when the system provides suitable abstractions. The current version of our system does not provide facilities to lower repetition viscosity. It does aim to lower knock-on viscosity relating to specification-edit actions. For example, affected cell references in formulas get automatically updated when a row or column is deleted.

Error proneness The error proneness of a system is an indicator of how the notation causes the user to commit errors. In situations where this problem might arise, the system should have built-in mechanisms to protect the user from committing errors.

We minimize *editing errors* that can occur during the creation of the specification by only allowing logically valid operations at every step during the edit process. For example, if any cell within a vertically repeating group is selected, the button for creating vertically repeating groups is disabled (this situation is shown in Figure 8).

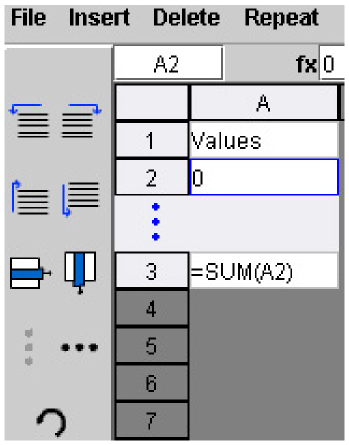


Figure 8: Within a vertically repeating group

A similar situation arises when any cell within a horizontally repeating group is selected. Another example is everything else is disabled while the user is in edit mode for formulas (this situation is shown in Figure 9).

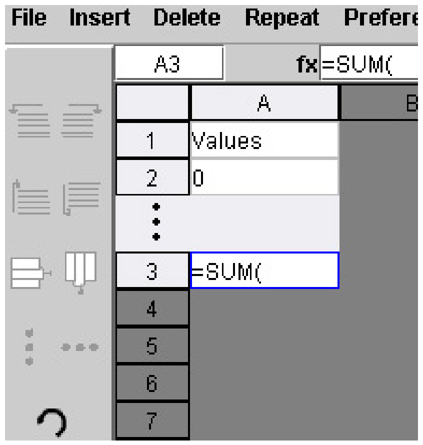


Figure 9: In formula-edit mode

Provisionality Users might view spreadsheet specification design as an exploratory activity. Provisionality refers to the support provided by the system to encourage the user to "play around" with possible alternatives and to carry out “what-if experiments” with their tasks.

The ViTSL system is very flexible as far as update and undo operations are concerned. Even though the system does not impose any constraints on how the user goes about creating the specifications, it limits the errors the user can make. The limitations discussed under *progressive evaluation* might restrict a “trial and error” approach to creation of specifications.

Closeness of mapping The closeness of mapping is an indicator of how closely related the notation is to the result it is describing.

Since the specifications developed in ViTSL will eventually be used with Gencel, we have adopted a structure similar to Excel to maintain closeness of mapping. Moreover, operations like insertion or deletion of rows or columns work the same way as in Excel. We deviate from the Excel model in that we provide separate buttons to insert rows above or below or to insert columns to the left or to the right of the currently selected cell. This design choice was made to provide greater flexibility to the user and is also consistent with the editing model of Gencel. We allow selection of ranges with shift-click to specify horizontally or vertically repeating groups and the cell addresses also work the same way they do in Excel.

Consistency When similar semantics are expressed in similar syntactic forms, users find it easier to infer the structure of the presented information. If this consistency is not maintained and there are many representations for the same thing in the notation, usability might be severely compromised.

Since ViTSL is targeted at expert users of Excel, the basic update operations in ViTSL have been designed to be consistent with those in Excel. The main deviation from Excel is that ViTSL has additional operations for forming repeating groups. In this case, the visual notation for vertically and horizontally repeating groups are similar as shown in Figures 10 and 11.

Secondary notation and escape from formalism Oftentimes the user might need to record additional information without using the formal syntax of the notation. To cater to this need, many systems support secondary notations that can be used any way the user likes. For example, programming languages allow users to include documentation within the source code using comments.

The current version of the system does not support any secondary notation. In future versions of the system, we plan to incorporate mechanisms by which the user can include documentation, both at cell and specification level, within the ViTSL specifications.

Progressive evaluation This dimension refers to facilities within the system that allow the users to assess their progress, even with partially complete specifications.

The current version of the ViTSL system does not have a readily available *test drive* mechanism so users could preview their work to check their progress. The user has to build the specifications using the ViTSL interface, save to file, and load it into Gencel to evaluate the effect of the customized update operations. The lack of a more direct feedback mechanism is a problem we hope to rectify in future versions by integrating the ViTSL editor within Gencel. The users would then be able to more easily switch between the ViTSL specification-edit mode and the Gencel table-creation mode to check their progress.

Role expressiveness This dimension refers to the ease with which the role or purpose of an entity in the notation can be inferred. A role expressive notation is easier to "pick up" since the roles of the entities and their relationships can be easily discovered.

Mainly, there are two kinds of entities in the ViTSL system—buttons for edit operations and the cells used to build specifications. In a given specification, cells can have two possible roles—they can be singleton cells or they can be part of a repeating group. This information is conveyed to the user through the row and column address bars to show the rows or columns that are part of a repeating group and by shading the backgrounds of cells within repeating groups differently from those that are not part of repeating groups.

Figure 10 shows the horizontally repeating group from the template given in Figure 7. The fact that columns B, C, and D are part of the same group is depicted by the absence of separators. The three horizontally aligned dots in the visual notation show that the group to its left is a horizontally repeating group. We have three horizontally aligned dots as the icon on the corresponding grouping button to show this relation.

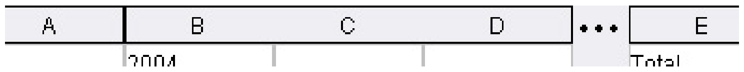


Figure 10: Horizontally repeating group.

An example of a vertically repeating group is shown in Figure 11. In this case, the three vertically aligned dots indicate that the group above it, row 3 in this case, repeats vertically. As in the case with the horizontally repeating cells, we have three vertically aligned dots as the icon on the button for the vertical grouping action to show the relation between the button and the action it performs.

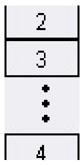


Figure 11: Vertically repeating group.

Hard mental operations This dimension is an indicator of the cognitive load the system places on the user. The notation would not be very usable if it made it complex or difficult for the user to figure things out by making high demands on the user's cognitive resources like attention and memory. Cell references in formulas within a template behave differently in generated tables depending on the cell the formula is in and the cell the reference is pointing to. The translation of references was briefly sketched in Section 4. The different cases are shown in Table 1.

Since this behavior is not very obvious from the visual notation, we have provided tooltip help

Source cell	Target cell	Behavior
non-repeating	non-repeating	reference
non-repeating	repeating	range
repeating	non-repeating	reference
repeating	repeating (same group)	reference
repeating	repeating (different group)	range

Table 1: Behavior of References

that indicates if a reference will get expanded to a range in the spreadsheet.

Premature commitment Ideally, the notation should not impose any *artificial* constraints on the order of doing things. Such constraints would force the user to make choices before they need to and with insufficient information. Therefore a notation which demands a high level of premature commitment would automatically be high on error proneness.

We had some initial concerns about the system forcing the user to think out the entire specification before starting the edit process. These concerns arose primarily from the fact that specifications start from a single cell and expand to the right and down depending on the update operations carried out by the user. To protect the user from imposition of any particular sequence of editing actions, we have provided separate buttons for inserting rows above and below and inserting columns to the left and to the right of the current cell. This feature allows the creation of a specification to proceed in all directions within the plane of the interface.

Visibility and juxtaposability The system should provide mechanisms that allow users to view components easily. For example, exploratory activities like debugging would require that reasonably-sized chunks of code can viewed in isolation and side by side with other code or execution windows.

Even in the case of very large spreadsheets (in terms of number of cells), the basic specification can be very small as can be seen in the budget sheet example shown in Figure 7. As a result of this higher level of abstraction, in most cases the specification will only require minimal screen real estate. In the rare case of very large specifications, we run into the same problem Excel users face with large spreadsheets—the need to scroll to view the full sheet. Another important aspect that reduces problems ViTSL might have with visibility and juxtaposability is that specifications are created at a table level. Therefore all the update operations are *local* to a table. On the other hand, an Excel spreadsheet might have more than one table and an update operation, an insert or delete column for example, might affect tables above or below the table the user is working on.

Summary The dimensions are not independent of each other and in many cases the language designer has to make trade-off decisions. In the case of ViTSL we have chosen to go with a representation similar to that of Excel since the target audience is assumed to be comfortable with Excel. Another factor that guides choices for trade-offs would be to aim at making common tasks easier to perform with respect to less common tasks. This concern has led us to provide easily accessible buttons for the common tasks. Cognitive Dimensions do not really help in this context since they are task neutral. However, they give us a good starting point and also help us focus on potential problem areas.

7 Related Work

Fujaba, short for "From UML to Java And Back Again", is a visual UML modeling tool which can automatically generate Java frameworks [15]. Fujaba can also reconstruct UML diagrams by analyzing Java code. Fujaba is similar to our system in that it may be considered a program generator from a specification. However, the Fujaba system does not provide any additional assurances on the safety or correctness of the Java code after it is generated, whereas we provide the Gencel environment to continue to ensure the specification is followed and the spreadsheet is correct.

Viatra, short for "VIsual Automated model TRAnsformations", is a UML-based tool which is designed to check consistency, completeness, and requirement satisfaction of code being designed [10]. This tool only performs these checks on the UML specification, thus ensuring that the specification is correct. However, once the specification is turned into code (either by hand or using a tool such as Fujaba), no assurance of the continued correctness is granted.

Gutwenger et al. have devised a system for visualizing complex UML diagrams in an attractive manner [17]. Their system takes a given abstraction (UML class network) and prepares a visualization for it. In contrast, our system uses visualization to define the abstraction (ViTSL specification) and establish a mapping between the visual elements, the Gencel spreadsheet, and the specification.

The pervasiveness of errors in spreadsheets has motivated some research into spreadsheet design [18, 29, 26], testing [27], and consistency checking [14, 7, 3, 2].

However, only little research has been performed on creating new, safer spreadsheet systems. The spreadsheet language Forms/3 [6] extends the spreadsheet paradigm by a number of features found in other programming languages. The language contains many experimental features, such as time, generalizations, gestures, and a model of sequence I/O.

Whereas the Forms/3 system focuses on extending spreadsheet expressiveness, the ViTSL/Gencel system places a strong emphasis on correctness of spreadsheets. However, both approaches share a commitment to using the spreadsheet model as a vehicle to bring an enhanced experience to end user programmers.

Evaluation of end user products may be done experimentally or through analysis. In this paper, we have chosen the Cognitive Dimensions model of analysis to direct the usability of the ViTSL editor. The Cognitive Dimensions of Notations was created to help designers evaluate their systems with respect to its usability [16, 4]. In addition to our use here, Cognitive Dimensions have been broadly applied to other programming and softfware engineering environments, such as evaluating the C# programming language [8] or UML use-case diagrams [9].

The approach of Peyton Jones and others [22] to extend Excel by user-defined functions uses a strong basis in end-user usability. The authors' primary goal is to design a method for end users to easily create new functions in their spreadsheets. They use a familiar spreadsheet model and attempt to minimize the impact (learning curve) on users while maximizing the user's productivity. The authors note that traditional spreadsheets often compute very complex models, but do not include any user-defined functions, which makes the spreadsheets needlessly complicated and difficult to maintain. The authors introduce a system which allows users to define functions within the existing spreadsheet model. The user creates a function sheet (which appears the same as any ordinary sheet) and defines the function from input cell(s) to an output cell. They may then use the function like any other built-in function. The authors' approach differs from our own in that they add uncontrolled power to a spreadsheet, while we seek to structure the actions a spreadsheet user takes. However, the works have many similarities. In addition to the common application of Cognitive Dimensions, both the authors' and our approach start with a familiar spreadsheet model. Both seek to improve end-user experience specifically. Both are more concerned with usability than adding new features to the spreadsheet system.

8 Conclusions and Future Work

In this paper, we have introduced a new visual language for describing spreadsheet models. Although software visualization techniques have been applied to other forms of programming, no general purpose visual abstraction of spreadsheet models has previously been described. In conjunction with the Gencel environment, spreadsheets designed with our model, ViTSL, are guaranteed to be free of formula or reference errors. In addition to the visual language, we have presented an editor which allows end users to graphically design spreadsheet models, which may then be imported into Gencel for safe use.

We have also analyzed the design of the ViTSL editor according to the Cognitive Dimensions framework in order to make the editor as straightforward and usable as possible. With the ViTSL editor and the Gencel environment, we have created and presented both a formalized, visual language for specifying correct spreadsheets as well as an implementation of the language and tools to operate such spreadsheets.

There are several directions for future work. First of all, we plan to carry out user studies using the ViTSL interface. The feedback from such a study would help us in refining the interface. Second, one major obstacle to the adoption of this new system is the collection of spreadsheets individuals and companies might already have. The users might have invested a lot of time and money in the creation of their spreadsheets and would not be willing to port them to the new system. This problem would be even more serious in cases where the model behind the original spreadsheet is not clear (or documented). To solve this problem, we are also working on a spreadsheet structure parser that would support the user in extracting possible ViTSL specifications from a given spreadsheet.

References

[1] UML. `http://www.uml.org`.

[2] 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.

[3] 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.

[4] A. F. Blackwell and T. R. G. Green. Notational Systems - The Cognitive Dimensions of Notations Framework. *HCI Models, Theories, and Frameworks: Toward and Interdisciplinary Science*, pages 103–133, 2003.

[5] 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.

[6] M. M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang. Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm. *Journal of Functional Programming*, 11(2):155–206, 2001.

[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] Steven Clarke. Evaluating a New Programming Language. *13th Workshop of the Psychology of Programming Interest Group*, pages 275–289, 2001.

[9] K. Cox. Cognitive Dimensions of Use Cases—Feedback From a Student Questionnaire. In *12th Annual Psychology of Programming Interest Group Workshop*, pages 99–122, 2000.

[10] G. Csertán, G. Huszerl, I. Majzik, Z. Pap, A. Pataricza, and D. Varró. VIATRA: Visual Automated Transformations for Formal Verification and Validation of UML Models. In *17th IEEE Int. Conf. on Automated Software Engineering*, pages 267–270, 2002.

[11] D. Cullen. Excel snafu costs firm $24M. *The Register*, June 2003.

[12] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger. Gencel — A Program Generator for Correct Spreadsheets. Technical Report TR04-60-11, School of EECS, Oregon State University, 2004.

[13] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger. Automatic Generation and Maintenance of Correct Spreadsheets. In *27th IEEE Int. Conf. on Software Engineering*, 2005. To appear.

[14] 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.

[15] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language. In *6th Int. Workshop on Theory and Application of Graph Transformation*, LNCS 1764, pages 296–309, 1998.

[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] C. Gutwenger, M. Junger, K. Klein, J. Kupke, S. Leipert, and P. Mutzel. A new Approach for Visualizing UML Class Diagrams. In *ACM Symp. on Software visualization*, pages 179–188, 2003.

[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] R. Koschke. Software Visualization in Software Maintenance, Reverse Engineering, and Re-Engineering: A Research Survey. *Journal of Software Maintenance*, 15(2):87–109, 2003.

[20] 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.

[21] R. R. Panko. Spreadsheet Errors: What We Know. What We Think We Can Do. In *Symp. of the European Spreadsheet Risks Interest Group (EuSpRIG)*, 2000.

[22] S. L. Peyton Jones, A. Blackwell, and M. M. Burnett. A User-Centered Approach to Functions in Excel. In *ACM Int. Conf. on Functional Programming*, pages 165–176, 2003.

[23] T. Quatrani. *Visual Modeling with Rational Rose and UML.* Addison-Wesley, 1998.

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

[25] C. Roman and K. C. Cox. Program Visualization: The Art of Mapping Programs to Pictures. In *Int. Conf. on Software Engineering*, pages 412–420, 1992.

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

[27] 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.

[28] A. Scott. Shurgard Stock Dives After Auditor Quits Over Company's Accounting. *The Seattle Times*, November 2003.

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