UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Empirical study of novice spreadsheet usage: Comparing two spreadsheet calculation paradigms

Markku Tukiainen

Report A-2000-1

ACM H.1.2, H.5.2, D.m
UDK 681.3.02, 519.68
ISSN 0789-7316
ISBN 951-708-906-6

Empirical study of novice spreadsheet usage: Comparing two spreadsheet calculation paradigms

Markku Tukiainen
E-mail: Markku.Tukiainen@Joensuu.FI

Abstract

Empirical studies of spreadsheet programming have commonly shown high over all error rates but much less attention has been paid to reasons for these errors. One often mentioned cause for errors is the low conceptual level of spreadsheet systems. By changing the conceptual level of spreadsheet system, we wanted to study whether this will produce different type of errors compared to traditional spreadsheet systems. In this paper we present an empirical study comparing the traditional spreadsheet calculation paradigm and the structured spreadsheet calculation paradigm that utilizes goals, plans and spreadsheet data structures in computation. The results show that the error behavior of novice spreadsheet users is systematically different between paradigms.

1. Introduction

Traditional spreadsheet calculation systems provide a fast method to express data and to make calculations, provided that there is some natural way to lay all the data on one or more two-dimensional sheets. Spreadsheet calculation is the most widely used end-user programming language. The success of spreadsheet calculation systems has been attributed to their ease of use, users can see what is to be done and do it (Kay, 1984) and to the metaphorical and visual nature of the user interface (Norman, 1986). Despite this ease of use (for more balanced view, see (Lewis and Olson, 1987)), spreadsheet applications tend to have a lot of faults (Panko, 1997): Even experienced users made errors in 44% of the cases reported by Brown and Gould (1987). Their study also showed that the errors were mostly made in entering calculations, not simply in typing constants: Seventy percent of reported errors were errors in formulae.

There are several reasons for errors in using the traditional spreadsheet systems. First, there are errors due to the characteristics of the spreadsheets systems themselves. For example, formulae are not visible all the time; they are represented linearly, whereas the spreadsheet itself is presented spatially, and the physical distance on the screen between a formula and the cells referred to by the formula is large. All these reasons hide the true structure of a spreadsheet from the user, leading to errors (Brown and Gould, 1987). Second, there are errors due to the physical limits of current computer systems. For example, if the amount of data entered is substantial, the physical limits of a computer screen force the user to see only a part of the application at a time. Usually, the user divides logically grouped data into smaller physical clusters in order to see and understand the workings of the application. Third, there are errors due to limitations of the current spreadsheet calculation model. For example, naming the references is the only way to structure computations (Ronen et al., 1989), the specification of absolute and relative cell referencing is vague (Doyle, 1990, Sajaniemi, 1999), and the connection between the computation and the data is established by referencing to locations in a sheet, not the actual data structures in the model. Usually, these have been attributed to the low conceptual level of current spreadsheet systems (Lewis and Olson, 1987, Ronen et al., 1989).

The common way to construct computations within a spreadsheet system is to first enter some data and then make a formula once and copy it to other locations as needed (with relative cell addresses changed automatically by the spreadsheet program). This type of action has been demonstrated in many studies (Brown and Gould, 1987, Napier et al., 1992, Olson and Nilsen, 1987-1988), for example Brown and Gould (1987) call this working "locally" i.e. nearly all of the copying operations took place immediately following the corresponding data or formula entry. Naming ranges for referring into in formulae, copying formulae and data, and moving the data and formula ranges are the normal means to structure the computation and appearance of a spreadsheet (Nardi, 1993). They are also the only means in traditional spreadsheet systems.

The low conceptual level of current spreadsheet systems manifests itself especially in the absence of the connection of the computation in-the-large and the data areas. The spreadsheet user enters formulae once and is supposed to remember to make changes to the referenced addresses if the input data area changes in size. For example, if a user creates a table of values and then writes an expression using the addresses of the corner cells in the table or a named reference to the table and later adds a new row (or column) to the end of the table, the calculation (i.e. the formula) is not updated automatically.

Traditional spreadsheet systems have not changed much computationally since VisiCalc in 1979. The complexity of the interface and the amount of new features like easy graphing and helping agents have increased, but the formula specification remains relatively unchanged. One example of slow change in specifying the calculation can be found in recent versions of Microsoft Excel: the AutoSum feature. The user selects an area and then presses the AutoSum button and gets, by default, columnwise sums. Of course, the user can point to a row and press AutoSum and get a rowwise sum and then copy it across the needed areas. But the problem remains the same; if the user adds a new row of values to the area, the AutoSum-formula is not updated to reflect this change in data.

Although traditional spreadsheet systems offer no other data structuring tools than range naming the users of a spreadsheet system do not see spreadsheets as collections of individual cells, but rather as logical collections of cells (Saariluoma and Sajaniemi, 1989). This has led to a new approach to spreadsheeting called the structured spreadsheet calculation (Hassinen et al., 1988). One of the main ideas of the structured spreadsheet calculation is to make these implicit structures explicit so that the users can refer to these structures as a whole and connect these structures to computation so that the changes in structures will be reflected automatically to the computation. By offering this kind of abstraction system, the conceptual level of the structured spreadsheet calculation becomes higher than the traditional spreadsheet calculation.

We wanted to examine whether this change of the conceptual level will have an impact on the number or the type of errors novice spreadsheet users will produce. Especially, if the types of errors are different this will have important implications to the further development of spreadsheet calculation systems. In this paper we present an empirical study comparing the traditional spreadsheet calculation paradigm and the structured spreadsheet calculation paradigm. The study was conducted using Microsoft Excel (a traditional spreadsheet system) (Microsoft, 1994) and Basset (a structured spreadsheet system) (Tukiainen, 1996).

The purpose of the study is to compare two spreadsheet paradigms, traditional and structured, having different conceptual levels. The subjects are, however, using actual spreadsheet systems (Microsoft Excel and Basset) that represent these paradigms. Some of the errors that subjects will make may be due to the specific system features while others may be affected by the paradigm used. Therefore, a clear distinction must be made between errors evoked by problems inherent to the paradigm and errors evoked by problems inherent to the spreadsheet system.

In the next section we define the main concepts behind the structured spreadsheet calculation paradigm. In section 3, we give an example of Basset usage. These two sections describe the conceptual level of the structured spreadsheet paradigm and make it possible to differentiate between paradigm evoked errors and spreadsheet system evoked errors in the later analysis. In section 4, we present the experiment and discuss the results. This section gives a classification of error types and discusses about reasons for these error types. We conclude in section 5 by supporting the view that raising the conceptual level of spreadsheet systems causes less paradigm evoked computational errors.

2. Structured Spreadsheet Calculation

Structured spreadsheet calculation (Hassinen et al., 1988) is a paradigm entirely different from the traditional spreadsheet calculation paradigm. It is based on the separation of logical and physical data structures for data representation, and on the separation of goals and plans for computation definition. *Physical data structures* refer to the surface layout of a spreadsheet and *logical data structures* refer to the conceptual data in the application and in the application domain. For example, the logical sequence of twelve elements (e.g., monthly sales) can be physically laid out as two lists containing six cells (two halves), or as four lists containing three cells (four quarters), or as two lists containing five cells and one list containing two cells (e.g., because of the physical width of the screen), or in some other way.

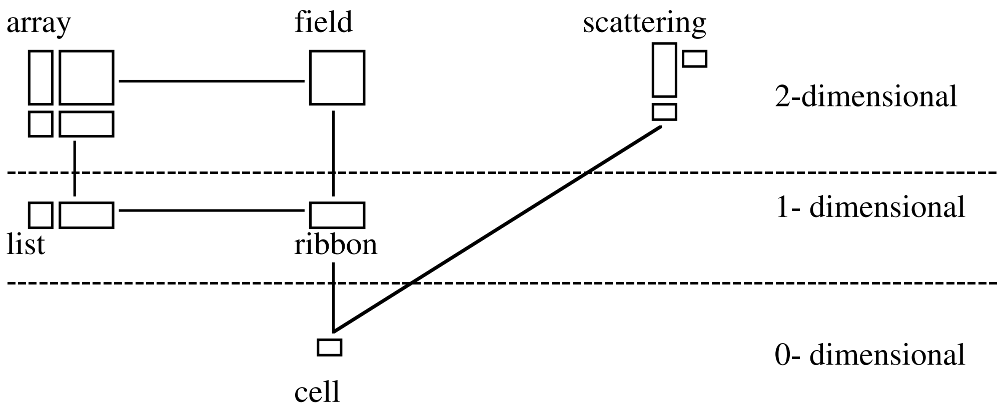

Figure 1: The hierarchy of physical data structures.

Computations in the structured spreadsheet calculation paradigm are defined as *goals*, which describe what the calculation will produce (like a total sum) and implemented by *plans*, which describe how the calculation is actually done. A user creates and uses logical data structures only in computations while physical data structures are used for external representation. The user can modify the representations, for example to add titles, split the structure to many pieces and locate the pieces in various parts of the sheet.

Figure 1 shows the hierarchy of physical data structures. A *cell* is the smallest unit of all data structures. It is an individual element containing a text, a number or a formula, and occupying a certain location in the spreadsheet. A *ribbon* is a consecutive line of one or more cells located in one row or in one column. A *list* is a line of one or more ribbons, all located in the same row (in case of row-wise ribbons) or in the same column (in case of column-wise ribbons). The lengths of the individual ribbons in a list may be different. A *field* is a rectangular collection of one or more cells and can be considered as a line of row-wise or column-wise ribbons. An *array* is a collection of two or more fields that could be thought to be obtained from a larger field by leaving out some rows and/or columns. Thus an array can also be seen as a list of lists. Finally, a *scattering* is any (unstructured) collection of individual cells not covered by any of the other forms.

Logical data structures reveal the conceptual relations of individual data items on the logical level. The hierarchy of logical data structures is shown in Figure 2. A *chamber* is the simplest unit. It is an individual data element describing text, a number or a value obtained through computations from other data. The only properties of a chamber are its identity (i.e., name) and contents. It is important to note that a chamber has no location information because, from the logical point of view, the position of a piece of information in a spreadsheet is unimportant per se. A *sequence* is an ordered, non-empty set of chambers. Thus, the elements in a sequence can be denoted by their index within the set. A sequence has no information of its position or direction (vertical vs. horizontal). A sequence may be *split* into two or more parts. The lengths of the parts may vary within a sequence. Sequences describe 1-dimensional application domain data, and the splitting within a sequence models the internal structure of application domain qualifiers. For example, the sales figures within a year can be modeled as a sequence split into 12 parts consisting of 31, 28, 31, ... elements.

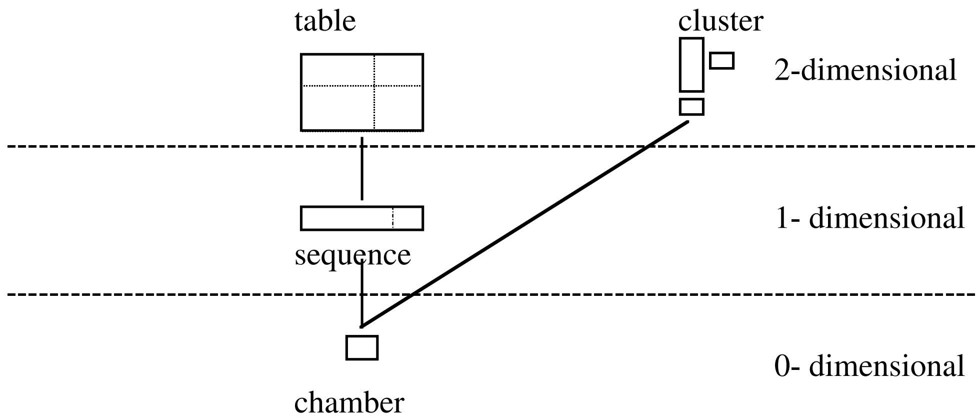

Figure 2: The logical data structures hierarchy.

A *table* is an ordered, non-empty set of similar sequences. Two sequences are similar if they have the same number of elements, or they are split into the same number of parts and the lengths of these parts are pair-wise equal. A table may be *split* into two or more parts of consecutive sequences. Similar to sequences, tables have no position or direction. Finally, a *cluster* is a logical collection of elements that cannot be structured in any of the previous ways.

Goals and plans provide a *schema-based approach* to expert programming knowledge that emphasizes the role of semantic structures more than the role of syntactic structures (Detienne, 1990). This approach can be used to account for problem solving and understanding activities in many general programming tasks, e.g., for coding and debugging. The schema-based approach has been developed mainly by Elliot Soloway and his colleaques who conducted a series of studies of novice Pascal programmers' bugs and misconceptions. According to them, expert Pascal programmers have and use high-level, plan knowledge that directs their programming activities (Soloway et al., 1982).

The notion of Soloway's programming plan comes from the text comprehension research notion of schema (Soloway and Ehrlich, 1984). A plan in Soloway's theory is a procedure or strategy in which the key elements of the process have been abstracted and represented explicitly. Soloway's group identified a set of plans which they thought to be used in simple Pascal looping programs. Figure 3 presents a set of such basic plans: running-total-variable-plan that captures the notion of a variable responsible for gathering the total sum of individual items, division-by-zero-guard-plan that unifies the actions needed to prevent a division by zero error to occur, etc.

```
PROGRAM alpha(Input, Output);
VAR Count, Sum, Number: INTEGER;
        Average : REAL;
BEGIN
Count := 0;
Sum := 0;
READLN (Number);
WHILE Number <> 99999 DO
        BEGIN
        Sum := Sum + Number;
        Count := Count + 1;
        READLN (Number);
        END;
IF Count > 0 THEN
        BEGIN
        Average := Sum / Count;
        WRITELN (Average);
        END
ELSE
WRITELN ('No valid inputs');
END.
```

Running total loop plan

Running total variable plan

Counter variable plan

Division by zero guard plan

Print plan

Figure 3: Soloway's basic programming plans (adapted from (Curtis, 1988)).

The notion of goals and plans in the structured spreadsheet calculation follows these ideas. As an example, Figure 4 defines two goals and three plans in a notation developed for spreadsheet knowledge representation (Sajaniemi et al., 1999). The goal TotalSum can be implemented by many plans, one of which is TotalSum_direct. This plan has as its subgoal the SumClause which can be implemented by the operator + or the function SUM.

```
G: TotalSum
        D: c1 = TotalSum(Input)
           TotalSum sums up all the elements in input
           structure to the output chamber
        P: _direct
           _splitted
           _cumulative

P: TotalSum_direct
        D: Description: c1 = TotalSum_direct(Input)
           TotalSum_direct sums up all the values of elements
           in the input structure directly
        T: c1: SumClause(Input)

G: SumClause
        D: c1 = SumClause(Input)
           SumClause sums up all the elements in input
           structure to the output chamber with a
           single addition
        P: _addition
           _sumFunction

P: SumClause_addition
        D: Description: c1 = SumClause_addition(Input)
           SumClause_addition sums up all the values
           of elements in the input structure
           using the addition operator
        T: c1: +Input.c1+...+Input.cn

P: SumClause_sumFuntion
        D: Description: c1 = SumClause_sumFuntion(Input)
           SumClause_sumFuntion sums up all the values
           of elements in the input structure
           using the sum function
        T: c1: SUM(Input)
```

Figure 4: An example of goals and plans (D = description, T = formula template).

Logical data structures correspond to application domain data, and physical data structures correspond to programming domain data representations of logical structures. Thus, computational goals are usually related to logical data structures while their implementations, i.e., plans, are related to physical data structures. These relationships are presented in Figure 5.

In spreadsheets, application domain goals are, however, simple, and hence practically all goals have a direct application level interpretation. Further, spreadsheet users tend to use subgoals that deal with data that has a clear meaning in the application domain, rather than operating with tricky computations using artificial data structures. As a consequence, goals deal with application domain data much more frequently than in procedural programming. For further discussion of the goals and plans approach to the programming knowledge, see (Davies, 1990, Davies, 1993, Detienne, 1990, Ehrlich and Soloway, 1984, Rist, 1989, Rist, 1995, Soloway and Ehrlich, 1984).

The conceptual level of the structured spreadsheet paradigm is higher than in traditional spreadsheets because of its more abstract data structuring mechanism and its approach to apply computations to structured data. Furthermore, the distinction between application domain (i.e., logical structures and goals) and programming domain (i.e., physical structures and plans) is made far more clear than in the traditional spreadsheet paradigm.

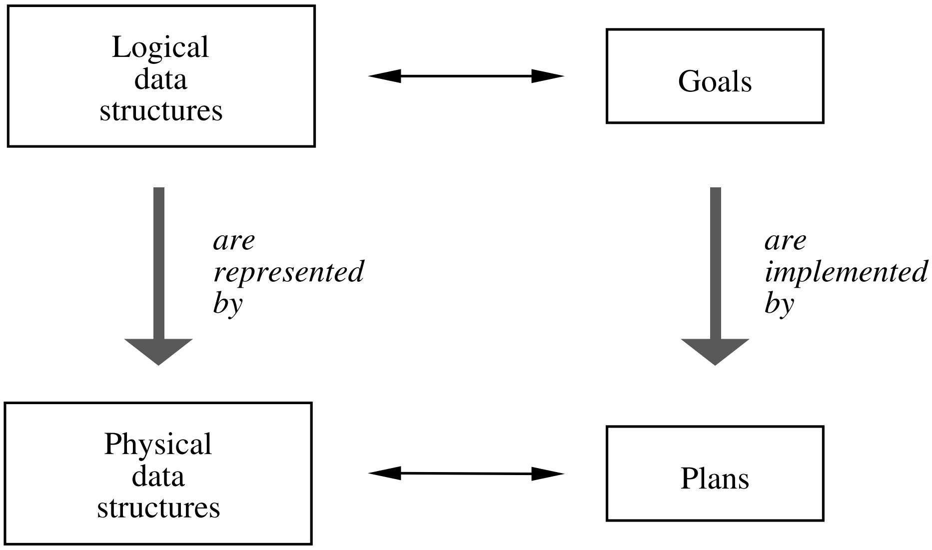

Figure 5: The relationships between data structures, goals and plans (Sajaniemi, 1989).

3. Basset – A Structured Spreadsheet System

Basset (Tukiainen, 1996) is a spreadsheet tool that uses the structured spreadsheet calculation paradigm. The objects in Basset consist of goals, plans and data objects that hold either input or output values for computations. The calculations are done by selecting system-implemented goals (like a cumulative sum) or by specifying one's own goals using the specification language within the system. All the data objects in Basset are collections of one or more cells. A user manipulates only these structures, not the individual cells. Input data objects for user data are called *input structures*. Output data objects holding the output values of a calculation are called *output structures*. The user can manipulate the physical layout of these structures on the screen.

The data structure types in Basset are chamber, sequence, and table. The simplest of these structures is a *chamber*, which consists of one cell. Chambers, like all structures, have names, which can be used in computations. A *sequence* is a one-dimensional collection of cells. A *table* is a two-dimensional collection of cells. In Basset the distinction between logical and physical data structures is implicit to the user. Logical data structures are used in specifying computations and physical data structures are their representations on the screen. For example, a logical data structure table can be split into a collection of sequences or to a collection of smaller tables on the screen. To the user this is just a representational feature. Calculations are always connected to logical data structures.

The system-implemented goals in Basset are based on an analysis of a set of spreadsheet applications (Sajaniemi and Pekkanen, 1988, Tukiainen and Sajaniemi, 1996). There are currently about 40 goals in Basset and each goal can have several plans. Computations can use both input structures and the output structures of other goals. Examples of Basset's goals are various sums such as row-wise sums of a table or cumulative sum of a sequence (accumulation). The user applies a computation by selecting a goal from the goal menu or by writing her own goal description if she does not find an appropriate system-implemented goal for her purposes. Basset has two user levels. On the *novice level*, the system tries to make decisions for the user and some of the features like plan implementation can be hidden. On the novice level, the system furthermore chooses the default plan for a goal automatically, while on the *expert level*, the system permits the user to select the plan or to write a plan of her own.

Plans in Basset define how the actual calculation will be done. One attribute in plan description is a calculation expression written in Basset's own specification language. A calculation expression can contain references to arguments of the goal, specific structures in the spreadsheet, and subgoals to be fulfilled with other plans. For example, Figure 6 shows a plan for the goal Cumulative Difference, which computes cumulative sums of two sequences and subtracts the elements in these two sequences pair-wise producing an output sequence of cumulative difference. Income and Expenses are names for actual physical structures in the sheet.

```
P: CumulativeDifference_UsingCumulativeSums
        D: sequence3 = DifferencePairwise(sequence1, sequence2)
        R: sequence1 = CumulativeSum(Income)
           sequence2 = CumulativeSum(Expenses)
        T: DifferencePairwise(sequence1, sequence2)
```

Figure 6: User-defined plan for the goal Cumulative Difference.

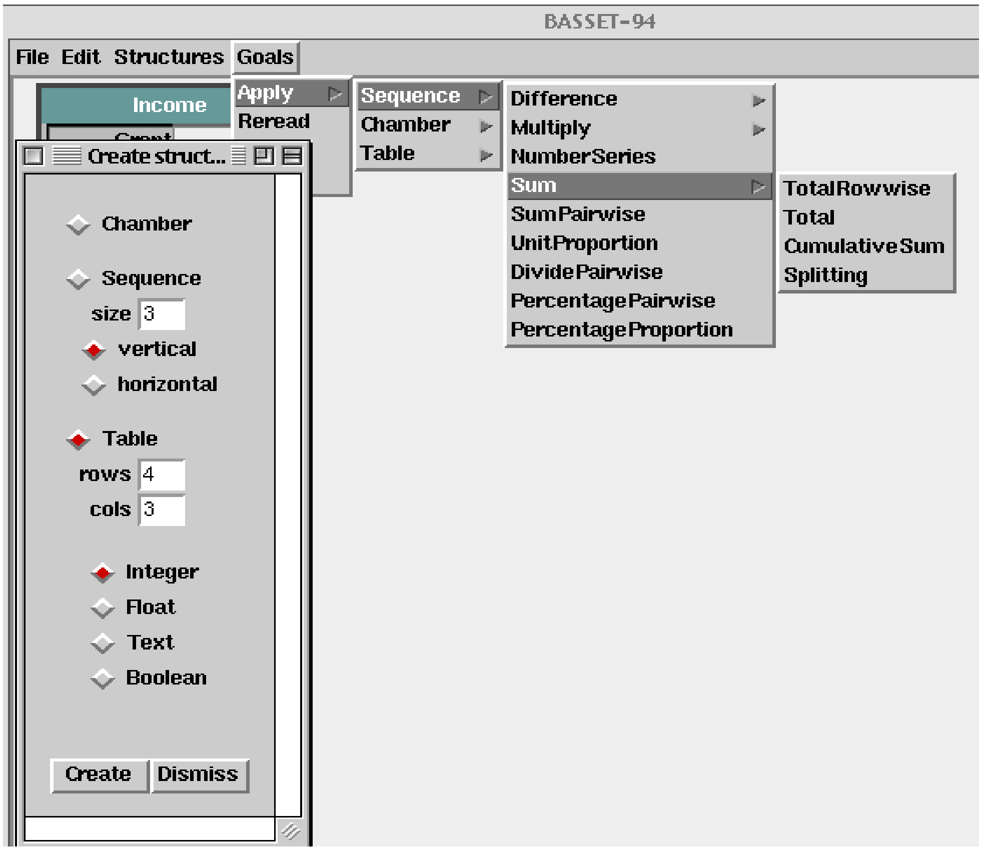

Figure 7: Create Structure dialog window.

As an example, consider a student creating a budget for her income and expenditure. The student has a government grant, a bank loan and other income, which have to cover for food, rent, leisure time and other expenses. First she will create sequences for her income and expenditure data and calculate the total sums of the sequences and the difference of these sums. Later she gets a scholarship from the university and she has to modify the income sequence.

The process begins by creating two sequences for the input data. The user creates the input structures using a dialog window (see Figure 7), where she determines the type of the structure (chamber, sequence, table) and the size of the structure. In the case of sequence the user also selects the layout (vertical or horizontal). She also selects the cell type of input structure, in this case integer. After the structure is created, the user enters the headings and numbers into the structure.

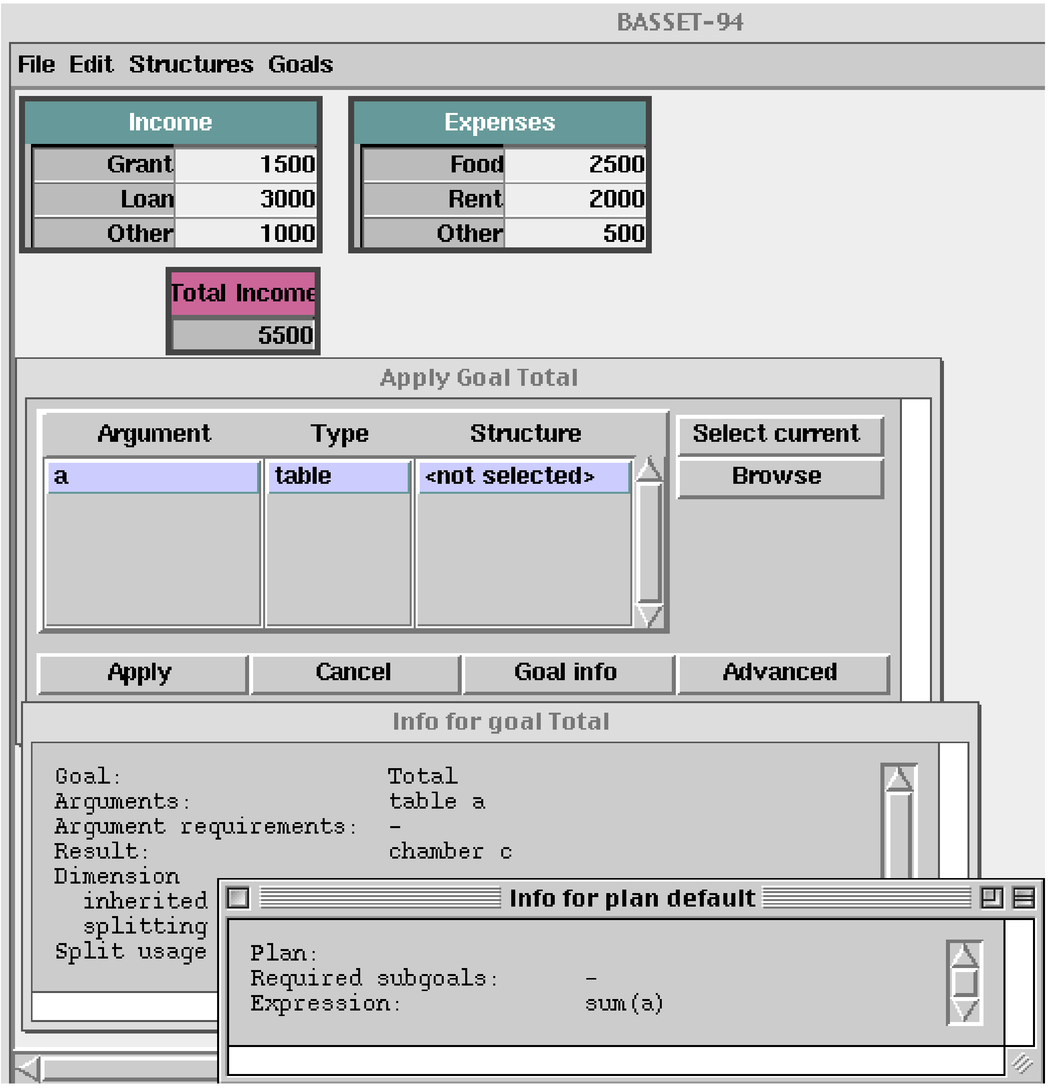

Figure 8: Applying goals.

Having created the input structures the user specifies the computation. This is done by selecting suitable goals from the *Goals* menu. The *Goals* menu is hierarchical, with categories such as Sums, Percentages, and so forth. The uppermost level in the *Goals* menu contains the data types (see Figure 7) limiting the number of appropriate goals by the type of the input structure. If a goal is applicable to many input data types, it will be found in all appropriate data type categories.

Figure 8 shows one goal already applied: the Total Sum of the Income sequence, with a chamber as the resulting output structure. The input structures and the output structures have different colors for better perception. The figure also shows an *Apply Goal* dialogue for the goal Total Sum. The user must select the sequence Expenses as an argument for the goal. Arguments are chosen either by selecting the currently active structure from the sheet (*Select current* button) or by selecting a structure from a list of existing structures (*Browse* button). The figure also has two information windows in which the goal and the plan specifications are shown.

Having applied the goal Total Sum to the sequence Expenses, the user applies the goal Pairwise Difference to the total sums of Income and Expenses and obtains a working application for her budget. She can modify the input structures, e.g., add or delete items, rows and columns. These changes will automatically be reflected in the output structures, i.e., output structures' sizes will change accordingly. Figure 9 shows a *Resize current structure* dialog window that can be used to add a new item into an input structure. In our example, the user wants to add an item Scholarship to the input sequence Income and she wants this item to be located after the item Other (see figure 9).

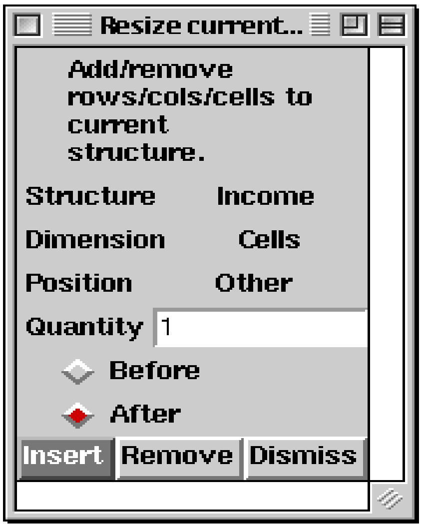

Figure 9: Resize current structure dialog window.

Figure 10 shows the final application in our example. The user has added the item Scholarship and the Total Sum goal, which has Income as its input structure, has been automatically updated to include Scholarship in its computation.

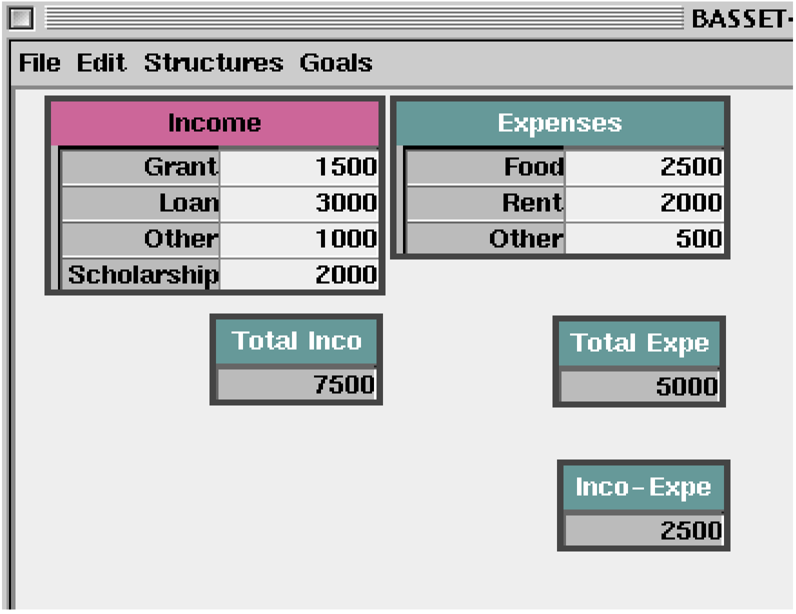

Figure 10: The final application in the budget task.

4. Experiment

The traditional spreadsheet calculation paradigm and the structured spreadsheet calculation paradigm differ quite much in their concepts and approaches to the spreadsheet calculation. This raises the following question:

> *Do programmers of the structured spreadsheet paradigm produce more or less calculation errors and do they produce different kinds of errors than programmers of the traditional spreadsheet paradigm?*

In the present experiment, we chose Microsoft Excel to be a representative system of the traditional spreadsheet calculation paradigm and Basset to be a representative system of the structured spreadsheet calculation paradigm. Since there are no expert programmers in Basset and the structured spreadsheet calculation paradigm, we chose to use novice spreadsheet programmers as subjects. We taught them four hours of spreadsheet calculation in general and then they practiced the use of one spreadsheet tool (Excel or Basset) for four hours. All subjects did the same tasks but using the tool assigned to the group. We measured the number of error-free and erroneous solutions and classified the errors. We were only interested in errors in calculations, not in layout or entering numbers and headings. The data collected consisted of the files containing the saved spreadsheets and the pre-test questionnaires of subject’s background. We also videotaped a few random subjects.

4.1 Method

4.1.1. Subjects

The subjects were first-year undergraduate students at the University of Joensuu majoring in various areas of study ranging from theology to mathematics. They participated in the experiment as a course requirement for an Introduction to Spreadsheets course. A total of 154 students participated in the experiment. They were randomly assigned to Basset (74 subjects) and Excel (80 subjects) groups. The course prerequisites include introduction to computer systems, Internet and other network facilities, university's computing facilities, the use of the university's library (e.g. database searching), and word processing. The main operating system used is Windows NT, so the students know a good deal of using different application programs in the Windows environment before they come to the Introduction to Spreadsheets course.

4.1.2. Materials

The experimental materials consisted of four spreadsheet applications. The application domain was simple business-type accounting. This was chosen because it is a well defined domain, students usually have a fair understanding of the domain, and it is easy to design simple problems in this domain. Also, spreadsheets are heavily used in business accounting. The tasks chosen for the experiment were quite elementary, but they are representative of real-life spreadsheet tasks.

We used two types of tasks: construction and modification tasks. In the construction tasks the subjects had to implement new spreadsheet applications, and in the modification tasks they were given existing spreadsheet applications to modify. The applications varied in size from 22 cells to 33 cells and the number of formulae in applications varied from 3 to 5. The tasks involved data entry for numbers and headings, composing formulae with relative and absolute cell referencing, copying formulae and saving the application. All task instructions contained the headings and the constant numbers needed in the spreadsheets.

The tasks were:

Task 1A and 1B: The budget task and updating the budget task.
Construction of a new application. The subjects were asked to create a budget for their income and expenditure. The income items (3) and expenditure items (4) were given. The solution was to first enter the data for the income and the expenditure. Then they had to calculate the total sums of the income and expenditure and to calculate the difference between these sums. At this point the instructions asked them to save the application (Task 1A). After saving, the instructions asked the subjects to modify the income by adding one item to the end of the income data (Task 1B). The Basset solution to the task 1B is almost the same as the spreadsheet of Figure 10. The only difference is that the input sequence Expenses does not include the item Fun.

Task 2.: The cumulative sum of credit units.
Construction of a new application. The subjects were asked to create an accounting application for their credit units using accumulation. The instruction showed an example application without the formulae. The subjects were instructed to create an application with

the same information, no demands for spatial layout were made. The credit units for 6 semesters were given. The solution was to enter the data and to create a cumulative sum.

Task 3. Adding new items to existing data structures.
Modification of an existing application. The application had two input data areas, one consisting of prices for 3 products and the other containing the numbers of sold items for each of the products. The titles for the areas and the items were taken from real-life settings, i.e., the titles were Prices, Sold Items, Hammers, Saws and so on. The application calculated sales values for the products using pair-wise multiplication of the prices and the numbers of items sold. The subjects were asked to add two new products into the application. The products were to be located as the second and the last item of the products. The formulae had to be updated to reflect these changes.

Task 4. Adding new calculations to an existing computation.
Modification of an existing application. The application to modify was the same as in the task 3. Subjects were asked to add a computation to calculate the sales values in Euros in addition to Finnish marks. The instruction showed an example application with the results of the formulae. The subjects were instructed to create an application with the same information; no demands for spatial layout were made. The solution was to add a cell containing a constant value for the exchange rate (given in the instructions), and to calculate the sales in Euros using multiplication of the sales in Finnish marks and the exchange rate.

4.1.3. Design

All subjects did all of the four tasks using either Excel or Basset. The tasks were ordered into 24 task sets, which contained all of the possible orders of the tasks. The task sets were distributed randomly to the subjects.

4.1.4. Procedure

The experiment took place in the same computer laboratories that the subjects had used for practicing sessions (4 hours). The subjects were tested in groups of three to thirteen. The groups were the same as used in the practicing sessions. First the subjects were given a background questionnaire to answer. When all were ready, (in about five to ten minutes) the task sets with a blank page on the top were distributed to the subjects and oral instructions about the experiment were given.

The subjects were told that there was going to be five tasks similar to those they had done in the practicing sessions. The first task would be the same training task for all. Then the four experiment tasks would follow, being the same for all but in a random order. This was told so that the subjects would not wonder if the subject next to him would stop working earlier than himself. The subjects were told that there was a time limit of ten minutes per task. If they would complete the task before the time limit, they were instructed to open a new blank spreadsheet on the screen and wait until the instructor would give permission to start working on the next task. If they could not complete the task in the time limit, they were instructed to save the work as it was, with all the incomplete or erroneous calculations.

4.2. Results

4.2.1. Data Analysis

The background of the subjects was controlled in the pre-test questionnaire where we asked their overall computer experience (variable A: ordinal scale), experience with spreadsheet calculation (B: years and C: number of applications done, ordinal scale) and experience in programming (D: years and E: size of largest application done, ordinal scale). Similarity of the groups was tested using Mann-Whitney Test (variables A, C and E) and t-test (variables B and D). There were no significant background differences between groups.

Table 1 gives the overall results of correctness. There are five tasks because the first task was divided into two subtasks, which were saved in two files. For various reasons 93 spreadsheets were missing, the largest number of missing spreadsheets being with Basset group in task 1B. Often the subjects did not finish the task 1A correctly, so they spend all the time doing that task. This happened in 20 cases of total 22 erroneous tasks 1A.

	Correct		Incorrect		Missing
	n	**%**	**n**	**%**	**n**
Task 1A: Excel	(58)	85.3	(10)	14.7	12
Task 1A: Basset	*(42)*	*65.6*	*(22)*	*34.4*	*10*
Task 1B: Excel	(57)	80.3	(14)	19.7	9
Task 1B: Basset	*(30)*	*76.9*	*(9)*	*23.1*	*35*
Task 2: Excel	(46)	60.5	(30)	39.5	4
Task 2: Basset	*(60)*	*84.5*	*(11)*	*15.5*	*3*
Task 3: Excel	(57)	75.0	(19)	25.0	4
Task 3: Basset	*(68)*	*95.8*	*(3)*	*4.2*	*3*
Task 4: Excel	(59)	77.6	(17)	22.4	4
Task 4: Basset	*(53)*	*76.8*	*(16)*	*23.2*	*5*
Total	531		146		93

Table 1: Task correctness and numbers of missing spreadsheets.

The Chi square test was used to test the interaction between the groups and the tasks. Table 2 shows the $\chi 2$–values and p-values. Interactions were significant in three tasks: Task 1A ($\chi 2=6.945$, df=1, $p=0.008$), task 2 ($\chi 2=10.496$, df=1, $p=0.001$) and task 3 ($\chi 2=12.449$, df=1, $p<0.001$).

Task	$\chi 2$	df	p	Sig.
1A	6.945	1	0.008	*
1B	0.172	1	0.679	
2	10.496	1	0.001	*
3	12.449	1	<0.001	*
4	0.014	1	0.906	

Table 2: The interactions between the group (Excel vs. Basset) and the tasks.

4.2.2. Error Analysis

We classified only errors that constituted distinctive groups. There were few errors that happened only once or twice and these were not classified. Also logic errors, i.e., errors in understanding the task description, were unclassified. As a consequence, the error percentage counts for each task do not sum up to a hundred percent.

There were four researchers independently devising the causes for errors and all reasons are listed without taking up for consideration of their probabilities. Each error type is given a number starting with the letter E denoting Excel errors or B denoting Basset errors. The error type name is followed by the error percentage for the task in present. This is followed by five numbers in parentheses giving error type percentages for tasks 1A, 1B, 2, 3 and 4 respectively. This does not tell anything about the overall commonality of the error types, but is just incidental dependency between the task characteristics and the error types. These error type percentages are followed by a list of possible causes for the error type. When a cause is specific to one task, i.e., it refers directly to certain calculation characteristics of the task, the task number is mentioned in parentheses before the cause description.

We do not include in the possible error types any cognitive factors like losing attention during the task, because our experimental method was not planned for that type of observation and because such analysis is not interesting for the purposes of this experiment. We try to explain the errors in terms of the computation doctrine, i.e., spreadsheet-based computation (as opposed, e.g., to procedural programming), the spreadsheet calculation paradigm used and the spreadsheet calculation tool's usability issues (see Figure 11). Thus in the possible causes for the error type, classes doctrine, paradigm, and tool are used.

Doctrine class includes error causes that could be due to the expected content or workings of the user's mental model of spreadsheet-based computation. This class contains also errors in mathematical concepts underlying the spreadsheet formulae. It has been noted that novice users' mental models in programming tend to be incomplete and erroneous, see for example Bonar's (1985a) study on novice Pascal programmers' mental models. Doctrine class errors can arise in both spreadsheet calculation paradigms discussed in this paper.

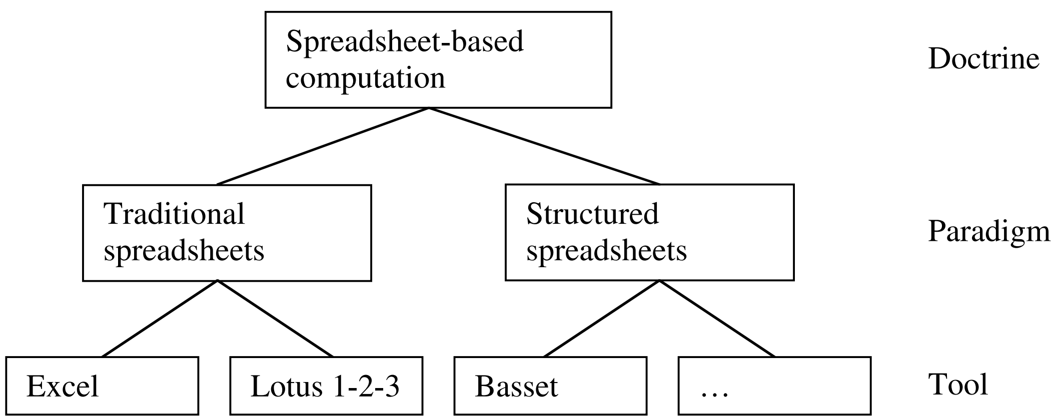

Figure 11: The hierarchy of error origins.

Paradigm class includes error causes that could be brought about the reasons of spreadsheet calculation paradigm used. These errors can arise only in one of the paradigms discussed. This class is the most important for the current study because it reveals the error causes connected directly to calculation paradigms.

Tool class includes error causes that could be brought about the usability problems in the specific spreadsheet tool used. This class is less important because we are not trying to study the usability of the tools but the differences in the paradigms behind the tools.

Possible error causes are described shortly in the following classification and further elaborated in the next part.

Task 1A: The budget task.

Excel Error type E1. Referenced sum area too large (sum includes empty cells at end) 50 % (50-0-0-0-0).

Possible causes:

Paradigm:	Subject perceives empty cells as white space and includes them into the calculation, because there is nothing in the cells that could affect the sum calculation. (This does not result an error in computation, but can be a potential error cause in future use of the application)
Paradigm:	Wrong symmetry demand. The referenced area for the sum of the first input sequence is enlarged to the same size as in the case of the second input sequence (i.e., four cells), although the size of the first input sequence is only three cells.
Paradigm:	Misunderstanding of Copy-operation. The sum formula of the second input sequence (four cells), is copied as the sum of the first input sequence, although the size of the first input sequence is only three cells.
Tool:	Subject separates the data and the calculations with empty cells and then uses Excel's AutoSum-feature, which automatically includes empty cells below the number area.

Excel Error type E2. Referenced sum area too small (not all appropriate cells included) 30 % (30-8-0-0-0).

Possible causes:

Paradigm:	Misusage of Copy-operation. The formula for the sum of the first (smaller) input sequence is copied for the second sequence, but the referenced area is not enlarged. The subject perceives the input sequences as objects that the computation should use, so copying a formula referring to one structure should work for another structure.

Basset Error type B1. Missing goal 68 % (68-25-64-0-31).

Possible causes:

Doctrine:	Subject does not succeed in constructing the right goal for the task.
Paradigm:	Distinction between goals is too delicate. Appropriate goal is not identified among related goals.
Tool:	Subject looks for the goal in the wrong submenu. In Task 1 the Pairwise Difference goal cannot be found in the Table submenu.

Basset Error type B2. Wrong goal applied 14 % (14-13-0-0-19).
Possible causes:

Paradigm:	The distinction between available goals is so small that a wrong goal is selected.
Tool:	The goals are listed in a single long list without any order (e.g., alphabetic) and the goal Pairwise Difference is at the end of the list of over 10 goals, so the subject can forget what goal she is trying to select.

Basset Error type B3. Wrong arguments for a goal 9 % (9-25-0-0-19).
Possible causes:

Doctrine:	Subject's mental model for mathematical operation does not differentiate which of the arguments is the subtrahend and which one is the minuend.
Tool:	The argument selection is a tricky operation. It is easy to make the selection in an erroneous way.
Tool:	The wording used in the implementation of computation selection is not natural for the subject. People do not usually "select arguments for subtraction operation", they say "a-b".

Task 1B: Updating the budget task.

Excel Error type E2. Referenced sum area too small 8 % (30-8-0-0-0). See Task 1A.

Excel Error type E3. A new item added to both input structures 23 % (0-23-0-0-0).
Possible causes:

Paradigm:	Adding a new item to the first input sequence using Insert rows-operation accidentally adds a new item to the middle of the second input sequence because the sequences were laid out horizontally next to each other. This new empty cell is automatically added to the referenced area of the sum of the second input sequence. Subject comprehends adjacent cell areas as independent objects and thinks that a spreadsheet system operation affects the individual object only.
Paradigm:	Subject does not know any other way to insert new items to an input structure than using Insert rows-operation. Possible work around is to lay out the structures vertically, as was done in almost all of the error-free application.
Paradigm:	Subject does not care of extra empty cells in the reference area of a function as they do not affect the result.

Excel Error type E4. Missing formula update 62 % (0-62-0-0-0).
Possible causes:

Paradigm:	Some insertion operations are automatically reflected in the computation (i.e., if the insertion is at the middle of the area), so a subject may think that all insertions are reflected in the computations, although adding a new item to the end of an input sequence requires changing the formulae referencing this area.

Paradigm: Subject perceives the data areas as whole objects that the computation uses, so adding a new item to a object should be reflected in the computations accordingly.

Basset Error type B1. Missing goal 25 % (68-25-64-0-31). See Task 1A.

Basset Error type B3. Wrong arguments for a goal 25 % (9-25-0-0-19). See Task 1A.

Basset Error type B4. No new item added into the input structure 38 % (0-38-0-0-0).
Possible causes:

Tool: Subject does not know how to add a new item into the existing structure.

Tool: Subject does not save the file after the change. Basset does not remind users to save the changes when quitting or opening a new sheet.

Task 2: The cumulative sum of credit units.

Excel Error type E5. First item in a cumulative sum is a constant 50 % (0-0-50-0-0).
Possible causes:

Doctrine: Subject does not comprehend the difference of a reference to a cell and a constant with the same value. It is easier to write a constant number 12 than a reference =A7.

Doctrine: Subject comprehends spreadsheets to consist of constants and calculations (formulas) where the distinctive feature of calculations is that they include operators or function calls. The formula =A7 does not have operators, so it may be perceived as not being calculation and so the subject uses the constant number 12 instead of the formula.

Paradigm: Misusage of the Copy-operation. Subject might think that she has made a dynamic copy of the constant value cell using Copy-operation (so that if the cell content changes, the copy will change also).

Excel Error type E6. Malformed formula 20 % (0-0-20-0-13).
Possible causes:

Doctrine: Subject has not enough knowledge to implement the right formula. Especially in task 2 the cumulative sum is a cognitively quite hard plan. We saw a total of seven different ways to produce the right result and six incorrect versions of these ideas.

Doctrine: Subject has no mathematical model for the computation needed, so she cannot implement it with a spreadsheet.

Excel Error type E9. Missing formula 23 % (0-0-23-53-44). See Task 4.

Basset Error type B1. Missing goal 64 % (68-25-64-0-31). See Task 1A.

Task 3: Adding new items to existing data structures.

Excel Error type E7. Wrong reference in a new formula 30 % (0-0-0-30-19).
Possible causes:

Paradigm: The subject does not use mouse pointing when inserting references to a formula and makes a visual slip when determining the cell reference for typing.

Paradigm: (Task 3) There were 3 items in both input sequences and a computation for their pair-wise multiplication. The places for new items occurred in the middle of the input sequences. The Insert rows-operation was applied to different places in the input structures, because the structures were laid out horizontally next to each other but not starting from the same row. This caused the cell references in some of the following formulae to change so that they did not refer to successive rows anymore. Thus the successive formulae in output structures did not have a similar visual form anymore. This may have confused subjects to select a wrong formula to copy.

Excel Error type E8. Incomplete but so far correct computation 25 % (0-0-0-25-0).
Possible causes:

Paradigm: The subject gets confused when the references in formulae updated are automatically after inserting new data items into input structures. The references in adjacent cells cross each other. This can confuse the novice subject to believe that the formulae are somehow wrong. The subject chooses a strategy to write over the old formulae by hand so that they will be “right” again. This takes longer time and they have not enough time to complete all formulae.

Excel Error type E9. Missing formula 53 % (0-0-23-53-44). See Task 4.

In Task 3 the Basset group had only three erroneous applications. In all cases, the ready-made calculations were deleted by the subjects. The reasons are hard to imagine, one possible explanation is that in Basset the output structure shrinks to a chamber whose value is ERR, if there are any problems in calculation. When a subject adds a new element to the first input structure, the numbers of elements in input structures differ and the pair-wise multiplication cannot be computed, so the temporary result is ERR. The subjects may have deleted the output structure at this point because it seemed to be wrong. Because of the small number of errors, no error types are listed for Basset in this task.

Task 4: Adding new calculations to an existing computation.

Excel Error type E6. Malformed formula 13 % (0-0-20-0-13). See Task 2.

Excel Error type E7. Wrong reference in a new formula 19 % (0-0-0-30-19). See Task 3.

Excel Error type E9. Missing formula 44 % (0-0-23-53-44).
Possible causes:

Doctrine:	Subject does not have the right mathematical model to calculate the currency exchange, so she cannot implement the computation.
Paradigm:	Subject does not understand the concept of absolute references. The calculation should refer to the multiplier cell with an absolute reference and the subject cannot enter it.
Paradigm:	(Task 3) Subject comprehends homogeneous areas as whole objects. In almost all cases the input areas were constructed correctly. Either this took all the time or adding new items to the input structures changed the references in formulae and this caused subjects extra mental load and prevented them from devising the new formulae.

Excel Error type E10. Malformed decimal number 24 % (0-0-0-0-24).
Possible causes:

Tool:	Subject uses period instead of comma as decimal number separator in currency values and Excel misinterpreted the content of the cell as text.

Basset Error type B1. Missing goal 31 % (68-25-64-0-31). See Task 1A.

Basset Error type B2. Wrong goal applied 19 % (14-13-0-0-19). See Task 1A.

Basset Error type B3. Wrong arguments for a goal 19 % (9-25-0-0-19). See Task 1A.

4. 3. Discussion

The error analysis above defined the types of errors the subjects produced in each of the tasks using the two tools. There were total of 10 different Excel and 4 Basset error types. We will now discuss about the possible explanations behind these error types.

4.3.1. Doctrine Evoked Errors

Three Excel error types (E5, E6, E9) and two Basset error types (B1, B3) have possible causes that could be explained by the structure and content of the subjects' mental model of spreadsheet computation.

There seemed to be at least four types of possible reasons for subjects' doctrine evoked errors. First there were errors concerning misunderstanding of spreadsheet concepts like the difference between a reference to a cell containing a constant value and using a constant with the same value in computation. Second there were errors due to fragmentary knowledge of spreadsheet operations and usage. Examples of this could be when the subjects could not change the layout of the sheet to compensate the effects of inserting areas. Another example could be a misconception of Copy-operation. The same Copy-operation is used to copy constants and formulae and the subjects seemed to think that copying constant values is a kind of dynamic copy so that if the original constant value will change, the copy of that value will also change.

Third there were errors resulting from cognitively hard tasks, e.g., in the error type E6 subjects could not devise a plan for cumulative sum. Another example of this could be the errors concerning the identification of all correct and needed goals for the solution. The error type B1 (missing goals) could be explained by subject not succeeding in constructing the right goal for the task. Fourth there were errors in mathematical models underlying the spreadsheet implementations of formulae. These could be classified to a different (higher) class than spreadsheet-based computation doctrine because they could manifest themselves in other computation doctrines also.

4.3.2. Paradigm Evoked Errors

For the purposes of this paper, paradigm evoked errors are most interesting as they may reveal fundamental differences between structured and traditional spreadsheet paradigms. Eight Excel error types (E1, E2, E3, E4, E5, E7, E8, E9) and two Basset error types (B1, B2) have possible causes that may be explained by the paradigm specific features of the two tools used in the experiment.

The conceptual levels of the two paradigms differ. The paradigm errors we found in this study can be explained by this difference in conceptual levels. The Excel errors deal mostly with the low level details of formulae. This can be seen as a manifestation of the lack of data structuring tools and poor connection between data areas and calculations. On the other hand, the Basset errors occur at the data structure level of computation. We will first elaborate upon Excel errors and then upon Basset errors.

Typical errors in Excel are connected to copying. In the error type E1 the referenced area is too large and one probable cause is the misunderstanding of the Copy-operation. There are two input data areas, which are of different size. A subject adds empty cells to the smaller area probably because she may think that the referenced areas have to be of the same size in order for a Copy-operation to succeed.

Another misunderstanding of the Copy-operation can be found in error type E5. The same Copy-operation is used to copy constants and formulae, which may confuse a novice subject to over generalize the concept of Copy-operation. Furthermore, the traditional spreadsheet calculation paradigm does not separate constant value cells from formula cells in the cell surface appearance. This may make it hard for a novice subject to distinguish between the concepts of a copy of the cell content and a reference to the cell.

Changing the input data area, e.g., adding new elements to the data, seemed to be a hard operation for novice subjects. The traditional spreadsheet paradigm utilizes a fixed grid of cells in its user interface. Adding new cells to the grid moves old areas to right or down and the new area looks good. But all the other input data or calculation areas that were stretching over the same rows and columns as the new added area, are mutilated.

In the experiment, this kind of addition of new cells to a particular input structure was done by most of the Excel subjects in such a way that the result looked like new empty cells were added in the middle of the other input structures also. This seemed to confuse the novice subjects, because they did not probably comprehend what did happen. Also, if the added cells were at the end of an old area the formulae was not updated to contain the new cells automatically. The novice subjects often forgot to update the referenced areas. Error types E3, E4, E7 and E9 could be explained by this type of action.

Subjects can comprehend homogeneous areas as objects. The error type E7 (adding new items to the input structures) changed the references in formulae, the formulae differed perceptually and the subjects did not perceive the formula area as a homogenous area although the cells were laid out uniformly. This probably caused them to switch their strategy from copying formulae (used in homogenous areas) to writing formulae one by one. It seems that in Task 3 subjects selected this strategy to correct and add new multiplication. This caused a lot of errors in formula referencing.

Overall the Excel subjects in this study seemed to interpret some type of objects consisting of multiple cells and sometimes they seemed to think that these objects could be used in calculations. This type of interpretation has been verified in empirical studies of spreadsheets earlier (Saariluoma and Sajaniemi, 1989, Sajaniemi et al., 1999). Other empirical work on spreadsheet calculation describes also similar cases, where user feels that the spreadsheet system should calculate using task specific larger structures than one cell (e.g., see subject Sue in Hendry and Green study (Hendry and Green, 1994)).

The Basset error type B1 (missing goals) could be explained by the amount of details in goal descriptions. The distinction between available goals can be small and that could be why a subject could not select the appropriate goal. The higher conceptual level of the structured spreadsheet calculation paradigm could also explain B1. The paradigm forces a user to think globally of the solution, i.e., to plan and design the solution before starting the implementation. This causes a lot of mental load for novices and is against the novices' preferred strategy to think locally of the solution (e.g., see Spohrer et al. (1985) for a discussion of Pascal novice strategies). This error manifests itself in the tasks where there is a chain of goals, i.e., tasks 1 and 4.

The number of concepts that is required to successfully use Basset is quite high. The concepts of the structures, the types of the structures, the modification of structure sizes, the ready-made goals and the concepts of calculation have to be mastered. The subjects did perform quite well, but in some tasks they seemed to make decisions based on fragmentary knowledge of the structured spreadsheet calculation and some surface level details of the goals in Basset. Studies of novice Pascal programming have demonstrated similar behavior (Bonar and Soloway, 1985b).

The error type B2, the selection of a wrong goal could also be explained by the larger amount of details to look for in Basset than in Excel even at the beginning level of spreadsheet calculation. The novice user has to master the goals of Basset in order to know what kind of computations there are in the system. The goal descriptions have to be read carefully in order to distinct the goals from each other. Some of the subjects did not pay attention to this and selected wrong goals for the task. In budget task most of the wrong goals were Pairwise Sums instead of Pairwise Difference. It could be argued that the time to learn Basset was short for mastering all of the goals in the system.

4.3.3. Tool Evoked Errors

Two Excel error types (E1, E10) and four Basset error type (B1, B2, B3, B4) have possible causes that may be explained by the usability features of the two tools used in the experiment.

Both tools have some strange features at their user interface and operations. For example, Excel shows the area to be copied surrounded by a flashing line, which is not a normal feature of Ms Windows standard. This seems to be distracting to novice users. Also Basset for example does not ask the user whether she wants to save the spreadsheet if it is not saved and the user opens a new sheet.

This category explains errors that were most likely caused by the low-level details of usage. One usability problem not addressed directly in the experiment was that Basset operates in the X Window environment. The look and the feel of the X Window environment was not familiar to the subjects in the experiment. This was remedied by not requiring the subjects to do any other actions in the X environment than using Basset. The counselor of the experiment session started the X environment and Basset beforehand for all the workstations used in the experiment.

There were six tool type error causes found in the experiment. The first was in Excel error type E1 (usage of AutoSum-operation). Excel's AutoSum-feature automatically includes empty cells between the referenced number area and the cell in which the formula is written. This does not produce an immediate error in computation but is a potential error cause for the future use of the sheet. The second tool type error cause was in E10 (malformed decimal number). This happened because some of the subjects confused the usage of comma and period. They used period as a currency value in task 4 and Excel misinterpreted the content of the cell as text. The text value could not be used in computation.

Basset is still a prototype implementation of the structured spreadsheet calculation paradigm. There are problems with the menu selections, the order of elements in menus and the division of goals into submenus. These problems cause extra mental load, which could be an explanation for Basset error types B1 and B2 in many cases. These Basset usability problems with menus were often mentioned at after the course questionnaires.

The fifth tool type error cause was in Basset error type B3 (wrong arguments for the goal). This was a serious error producing many erroneous spreadsheets in tasks one and four. The arguments of a goal are assigned when the goal is applied in Basset. This operation's usability in Basset is poor. A user has to first select the first argument and then select a structure using several user interface operations. After this the user has to select the next argument and so on. The arguments appear in the order they are originally written into the goal specification and this may not meet the requirements of the user. For example, in the task 4 we have MultiplyByConstant goal, which has a chamber (multiplier) and a table (multiplicand) as its arguments. A user can select this goal from the category Chamber (from Goals menu) and think that she is assigning the chamber as the first argument, but the first argument in the goal specification is the table (multiplicand).

The sixth tool type error cause was in Basset error type B4 (no new item added into the input structure). This error type happened in task 1B but the same subjects could perform the same operation correctly in task 3. One likely explanation is that the subjects performed the operation correctly in task 1B but forgot to save the file.

4.3.4. Results summary

The Excel subjects performed significantly better in task 1 than the Basset subjects. This could be explained by strategic differences of the tools. Basset forces a user to work with a more global approach to the solution. The user has to plan beforehand what kind of input structures to use, whether there is an appropriate goal existing etc. Excel allows a user to work at the local cell level, concentrating upon the details of the current cell entry at a time. The local cell level entry strategy seems to work for simple application at the novice level faster than Basset's approach. Also the task 1 required largest amount of cell entries. Entering the cells is slower with Basset than with Excel because Basset requires creating the input structures first and adding the titles require much extra work with multiple menus and dialog windows. This could have slowed Basset users down resulting the numerous incomplete applications.

The Basset subjects performed significantly better in tasks 2 and 3 than the Excel subjects. Accumulation seemed to be a hard calculation for novice spreadsheet users. The Excel subjects used the largest assortment of different plans in task 2. They also made the most errors in this task. The accumulation is a system-implemented goal in Basset. For the Basset subjects it was enough that they understood the goal and could apply it within the system. This explains the difference in groups' performance in task 2. In task 3, inserting new items to existing data and calculation areas in Excel was difficult for the novice subjects because the insertion mutilated other data structures and the cell references in existing formulae. In Basset insertion is an operation applied to an input data structure and the necessary changes in computations will be done automatically by the system. It seems that Basset's higher conceptual level helps the novice users in more complex spreadsheet tasks.

The error classification revealed the largest number of causes in the class paradigm evoked errors. There were 8 paradigm evoked error types in Excel and 2 in Basset. This result is interesting because it suggests that raising the conceptual level of spreadsheet system benefits the users. The overall error numbers were smaller in Basset than in Excel and the classification revealed less paradigm evoked errors than other errors. In the other classes, there were 3 Excel error types and 2 Basset error types in the doctrine evoked error class and 2 Excel and 4 Basset error types in the class tool evoked errors. These are not so interesting because their reasons cannot be remedied by changing the tools or are easy to fix.

5. Conclusions

The results show that the error behavior of novice spreadsheet programmers is systematically different between the traditional spreadsheet calculation paradigm and the structured spreadsheet calculation paradigm. The reasons for these error behaviors were considered in the context of the content of subject's mental model of the spreadsheet-based computation doctrine in general, the characteristic differences between the two spreadsheet calculation paradigms, and the usability of the two tools used. We classified the error types into these classes and discussed the probable causes for each error type.

There were 8 paradigm evoked error types in Excel and only 2 in Basset. To our opinion this shows that Basset has succeeded in raising the conceptual level of spreadsheet calculation preventing some of the problems of the traditional spreadsheet paradigm to occur in structured spreadsheets. The remarkably low error rate in Basset users group for paradigm evoked errors makes it evident that chancing the conceptual level of spreadsheet systems will have a significant effect on the number and the type of errors novice spreadsheet users will produce.

In doctrine evoked error class, there were 3 Excel error types and 2 Basset error types. These are errors that can only be remedied by learning and experience. This was a study of novice subjects and the difference between the paradigms in this error class is not great. There were only 2 Excel and 4 Basset error types in the class of tool evoked errors. These are the easiest error causes to fix because they require only minor changes in the user interface of the tools.

The traditional spreadsheet calculation paradigm has been criticized for it's low conceptual level. It has been argued that this low conceptual level causes many errors in spreadsheet development and use. Our study supports this view by finding a large number of traditional spreadsheet paradigm evoked errors. Yet there has not been many attempts to change the conceptual level of spreadsheet calculation (Napier et al., 1989). The structured spreadsheet calculation paradigm is one effort to help the users of spreadsheet calculation to perform more error freely by raising the conceptual level of spreadsheets. The current study clarified some error types in both the traditional and the structured spreadsheet paradigms. This provides a promising area for further investigation of the underlying reasons for these error types, the advancement of both paradigms and the development of better tools to support spreadsheet users.

Acknowledgments

The author would like to thank Jorma Sajaniemi for his deeply thought advice and warm encouragement during this work, Kari Hassinen, Marja Kuittinen and Jorma Sajaniemi for their insightful explanations of the spreadsheet errors, and Juha Hakkarainen for running one half of the experiment sessions.

References:

Bonar, J. & Soloway, E. (1985b). Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programming. *Human-Computer Interaction* **1**(2): 133-161.

Bonar, J. G. (1985a). *Understanding the Bugs of Novice Programmers.* Ph. D. Thesis, Computer and Information Science, University of Massachusetts.

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

Curtis, B. (1988). Five Paradigms in the Psychology of Programming. In Helander, M. *Handbook of Human-Computer Interaction.* Elsevier Science Publishers B. V.

Davies, S. P. (1990). The nature and development of programming plans. *International Journal of Man–Machine Studies* **32**: 461-481.

Davies, S. P. (1993). The Structure and Content of Programming Knowledge: Disentangling Training and Language Effects in Theories of Skill Development. *International Journal of Human–Computer Interaction* **5**(4): 325-346.

Detienne, F. (1990). Expert Programming Knowledge: A Schema-based Approach. In Hoc, J.-M., Green, T. R. G., Samurcay, R., Gilmore, D. J. *Psychology of Programming.* London, Academic Press.

Doyle, J. R. (1990). Naive users and the Lotus interface: a field study. *Behaviour & Information Technology* **9**(1): 81-89.

Ehrlich, K. & Soloway, E. (1984). An Empirical Investigation of the Tacit Plan Knowledge in Programming. In Thomas, J. C. & Schneider, M. L. *Human Factors in Computer Systems.* Norwood, New Jersey, Ablex Publishing Corporation. 113-133.

Hassinen, K., Sajaniemi, J. & Väisänen, J. (1988). *Structured Spreadsheet Calculation.* 1988 IEEE Workshop on Languages for Automation, Computer Society Press, 129-133.

Hendry, D. G. & Green, T. R. G. (1994). Creating, Comprehending and Explaining Spreadsheets: a Cognitive Interpretation of What Discretionary Users Think of Spreadsheet Model. *International Journal of Human-Computer Studies* **40**: 1033-1065.

Kay, A. (1984). Computer Software. *Scientific American* **251**(3): 41-47.

Lewis, C. & Olson, G. M. (1987). *Can Principles of Cognition Lower the Barriers to Programming*. Empirical Studies of Programmers: Second Workshop, Washington, D. C., Ablex Publishing Corporation, 248 - 263.

Microsoft (1994). Microsoft Excel User's Guide ver. 5.0. Ireland, *Microsoft Corporation*.

Napier, H. A., Batsell, R. R., Lane, D. M. & Guadagno, N. S. (1992). Knowledge of Command Usage In a Spreadsheet Program. *DATA BASE* **23**(1): 13-21.

Napier, H. A., Lane, D. M., Batsell, R. R. & Guadagno, N. S. (1989). Impact of a Restricted Natural Language Interface on Ease of Learning and Productivity. *Communications of the ACM* **32**(10): 1190-1198.

Nardi, B. A. (1993). *A Small Matter of Programming: Perspectives on End User Computing*. London: MIT Press.

Norman, D. A. (1986). Cognitive Engineering. In Norman, D. A. & Draper, S. W. *User-Centered System Design.* Hillsale, NJ, Lawrence Erlbaum. 31-61.

Olson, J. R. & Nilsen, E. (1987-1988). Analysis of the Cognition Involved in Spreadsheet Software Interaction. *Human-Computer Interaction* **3**(4): 309-349.

Panko, R. (1997). *Spreadsheet Research (SSR) http://panko.cba.hawaii.edu/ssr/.* (5.1.2000).

Rist, R. S. (1989). Schema Creation in Programming. *Cognitive Science* **13**: 389-414.

Rist, R. S. (1995). Program Structure and Design. *Cognitive Science* **19**: 507-562.

Ronen, B., Palley, M. A. & Lucas, H. C. J. (1989). Spreadsheet Analysis and Design. *Communications of the ACM* **32**(1): 84-93.

Saariluoma, P. & Sajaniemi, J. (1989). Visual information chunking in spreadsheet calculation. *International Journal of Man-Machine Studies* **30**: 475-488.

Sajaniemi, J. (1989). Goals and Plans as a Basis for User Interfaces in Spreadsheet Calculation. In Pulliainen, K. & Sihvo, H. *West of East.* University of Joensuu. 129-140.

Sajaniemi, J. (1999). Modeling Spreadsheet Audit: A Rigorous Approach to Automatic Visualization. *Accepted for Journal of Visual Languages and Computing* .

Sajaniemi, J. & Pekkanen, J. (1988). An Empirical Analysis of Spreadsheet Calculation. *Software–Practise and Experience* **18**(6): 583-596.

Sajaniemi, J., Tukiainen, M. & Väisänen, J. (1999). *Goals and Plans in Spreadsheet Calculation.* Dept. of Computer Science, University of Joensuu, Technical Report, A-1999-1.

Soloway, E. & Ehrlich, K. (1984). Empirical Studies of Programming Knowledge. *IEEE Transactions on Software Engineering* **10**: 595-609.

Soloway, E., Ehrlich, K., Bonar, J. & Greenspan, J. (1982). What Do Novices Know About Programming. In Badre, A., Shneiderman, B. *Directions in Human/Computer Interaction.* Ablex Publishing Co.

Spohrer, J. C., Soloway, E. & Pope, E. (1985). A Goal/Plan Analysis of Buggy Pascal Programs. *Human-Computer Interaction* **1**: 163-207.

Tukiainen, M. (1996). ASSET: A Structured Spreadsheet Calculation System. *Machine-Mediated Learning* **5**(2): 63-76.

Tukiainen, M. & Sajaniemi, J. (1996). *Spreadsheet Goal and Plan Catalog: Additive and Multiplicative Computational Goals and Plans in Spreadsheet Calculation.* Dept. of CS, University of Joensuu, Technical Report, A-1996-4.