Quality Control in Spreadsheets: A Software Engineering-Based Approach to Spreadsheet Development

Kamalasen Rajalingham, David Chadwick, Brian Knight, Dilwyn Edwards
Information Integrity Research Centre
School of Computing & Mathematical Sciences
University of Greenwich, United Kingdom
k.rajalingham@gre.ac.uk

Abstract

This paper presents work conducted towards the development of an effective software engineering-based methodology for integrity control in the process of spreadsheet development. Various approaches and concepts within the discipline of software engineering are investigated. The proposed methodology consists of a set of coherent stages incorporating relevant software engineering techniques and principles. The framework for spreadsheet quality control is mainly aimed at addressing the widespread problem of spreadsheet errors. This paper elaborately discusses the application of relevant software engineering techniques and principles in the construction of spreadsheet models, accompanied and supported by appropriate examples. The principles and techniques of tree-based formula representation, unique definition of spreadsheet elements and separation of data and operations, are among the main features explored..

1. Introduction

Numerous publications over the years have described the seriousness of the problem of spreadsheet errors as well as the extent to which it has adversely affected businesses. There are three different perspectives from which the phenomenon of spreadsheet errors can be investigated [1]. They are as follows:

- the frequency of the occurrence of the errors
- the real-life consequences of spreadsheet errors
- the actual errors themselves

1.1. Frequency of spreadsheet errors

This area has attracted much attention and research. This is evident from the fact that most of the literature on the spreadsheet integrity problem concern the frequency of the occurrence of spreadsheet errors. Numerous related experiments and studies have been conducted in the past and adequate information is available.

Panko and Halverson [2] have provided an excellent compilation of such information from a host of different sources. Apart from that, large and well-known auditing firms such as KPMG Management Consulting and Coopers & Lybrand have reported that spreadsheet errors are occurring at appalling rates. According to a financial model review by KPMG Management Consulting, London [3], at least 5 errors were found in 95% of the financial models reviewed. In addition to that, an article in *New Scientist* [4] has reported that a study conducted by the British accounting firm Coopers & Lybrand found errors in 90% of the spreadsheets audited. Therefore, it is beyond any doubt that the rate of occurrence of these errors is in fact significantly high.

1.2. The real-life consequences

The second aspect of the spreadsheet integrity phenomenon concerns the real-life impact and consequences of these errors. Investigating the adverse effect of spreadsheet errors on businesses leads to a better insight into the seriousness of the situation. Such information has been presented in numerous publications. It must however be noted that these are just reported cases. There must be many other similar cases that have not been brought to public attention due to fear that it might affect the reputation of the company involved. Some of the reported cases are given in Chadwick et al [5], Rajalingham & Chadwick [6] Rajalingham et al [1].

An analysis of a large collection of reported cases over a decade, on the problem of spreadsheet errors and its negative consequences, clearly shows the extent of the damage that can be caused. Spreadsheet errors adversely affect the integrity and reliability of spreadsheet models. Consequently, numerous problems are encountered as a result of poor decisions made based on the unreliable or incorrect figures on spreadsheets.

1.3. The specific errors and flaws committed

This is an issue that has not at all been adequately explored or discussed in publications on the problem of spreadsheet errors. Four research papers that have addressed this issue and analysed specific types of spreadsheet errors from business and academia are Panko and Halverson [2], Chadwick et al [5], Rajalingham & Chadwick [6], and Rajalingham et al [1]. The outcome of research into specific types of spreadsheet errors is the provision of a more comprehensive classification or taxonomy of spreadsheet errors, presented and elaborately discussed by Rajalingham & Chadwick [6] and Rajalingham et al [1].

Figure 1 shows a marginally improved version of the model of classification of spreadsheet errors given by Rajalingham & Chadwick [6] and Rajalingham et al [1]. These papers also contain an elaborate discussion of the errors with examples.

2. Applying software engineering principles

2.1. A spreadsheet is a computer program

The increasing complexity and importance of spreadsheet models means that they have to be viewed differently. Contrary to the traditional view that a spreadsheet is merely a flexible electronic worksheet, it should be viewed as a computer program. A close examination of spreadsheet structure would reveal that a spreadsheet is fundamentally similar to a computer program.

2.2. Unique Definition of Spreadsheet Elements

Various concepts and principles from structured analysis and design can be used in spreadsheet development as well. *Uniqueness* is a concept that is present in many parts and aspects of structured

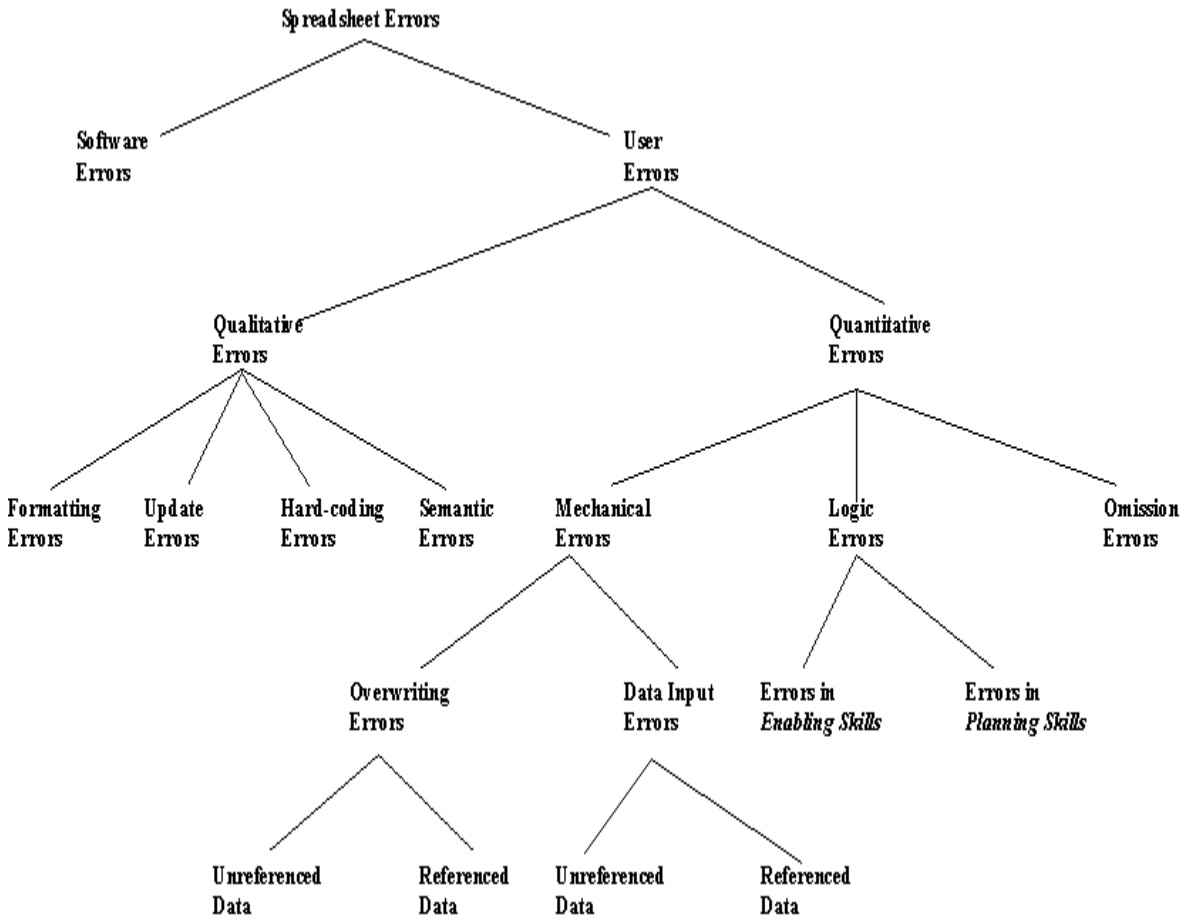


Figure 1. Types of spreadsheet errors

approaches. The main purpose of this is to prevent ambiguity and confusion. In relational database design, there has to be uniqueness in the following aspects of design:

- Every relation or entity type has a distinct or **unique** name.
- Every attribute within the same relation/entity type must be **unique**.
- Every relation must have a primary key, which is an attribute or set of attributes capable of **unique**ly identifying a particular occurrence of the relation or entity type.

Observing spreadsheets, we find that there are no mechanisms for enforcing such constraints. Consequently, there are many spreadsheet models designed without due attention paid to the uniqueness of the definition of elements within the model. The example [7] in figure 2 is used to illustrate this.

Two important design flaws can be identified from the spreadsheet model in figure 2 [1]. They are as follows:

- A data value is not associated with any label e.g. **cells B9 (114,290)** and **C23 (36,019)**. Although the meaning can be understood by carefully studying the model, it is not immediately obvious, especially to someone who is not familiar with such models.
- A label is placed on the same row as two different values, not distinguished by a column label. This results is some ambiguity as to exactly which value the label refers to e.g. **cells B27** and **C27**, both being on the same row and labelled *foreign exchange*. Only after carefully studying the model it is known that **cell C27** actually represents the 'total of appropriations'.

These problems can be overcome by ensuring that each element in the spreadsheet model is uniquely and unambiguously defined. The model in figure 3 shows the improved or corrected version of the model in figure 2.

2.3. Hierarchical decomposition in *tree* form

Hierarchical decomposition is a technique commonly used in software engineering to gradually break-down the complexity of programs. A similar approach can be adopted in spreadsheet design and development. However, prior to attempting to break-down the complexity of a spreadsheet model, an analysis of the basic structure and components of a spreadsheet model is essential.

2.3.1. Analysis of spreadsheet structure. Generally, the principal elements of a spreadsheet model are *labels*, *data values* and *formulae*. There are many ways in which these terms can be defined. For our purposes, data values refer to the bits of data directly entered or fed into the model by users. A formula uses these data values and even other formulae to perform a particular operations. Labels are associated with data values and formulae to provide them with a meaning. Some labels describe the entire model.

Referring to figure 3, examples of data values would be **40,360 (cell B6)**, **4,515 (cell B14)** and **800 (cell B27)**. An example of a formula is **40,360 (cell B6) + 72,360 (cell B7) + 1,570 (cell B8)**, the result of which is **114,290 (cell B9)**. An example of a label is **Gross profit (cell A11)**, which offers a meaning to the formula result **73,556 (cell C11)**.

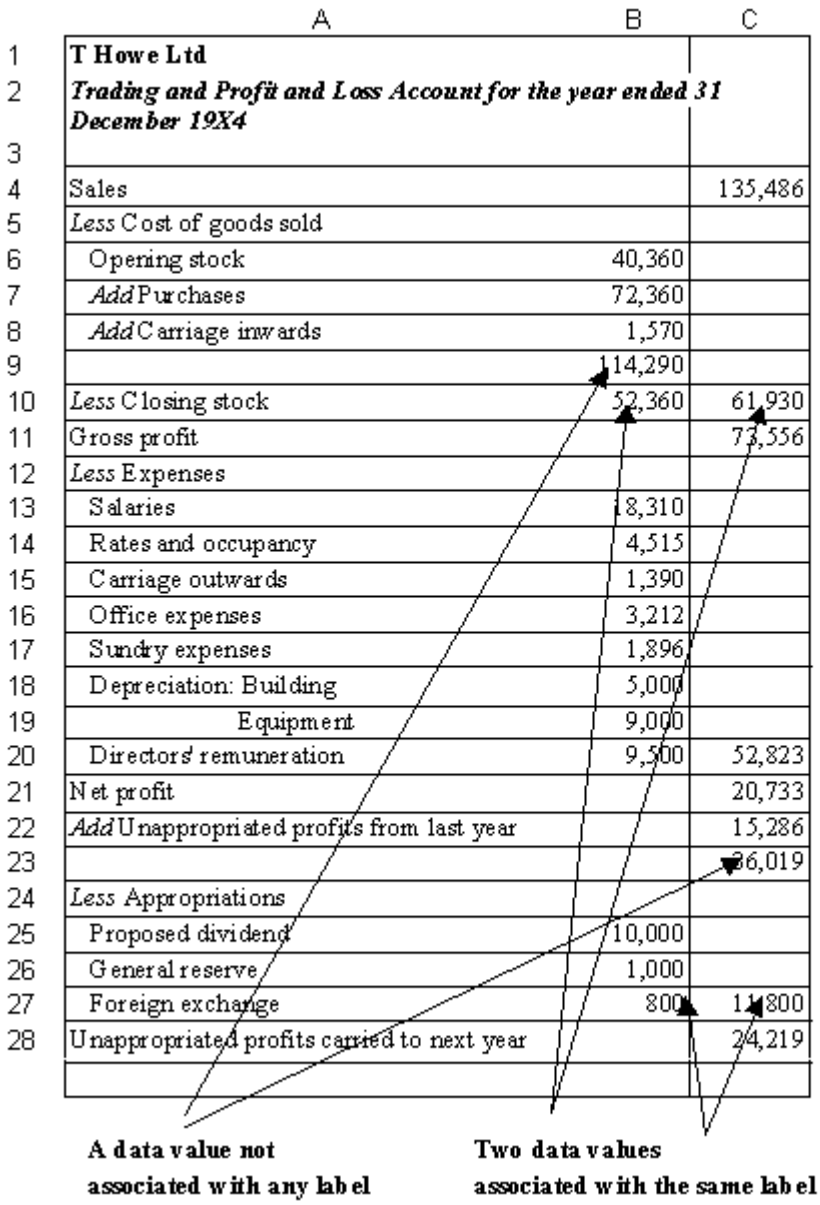
Research undertaken into spreadsheet errors has revealed that a majority of errors committed in spreadsheets are formula-based errors. In most cases, the formula is incorrectly constructed either due to a lack of understanding of the underlying algorithm or carelessness in entering the right cell addresses. Formulae are the most important elements of the spreadsheet model as they are the ones that actually calculate and provide useful results based on raw, user-entered data values. The process of formula construction in spreadsheets is basically a form of computer programming. The model in figure 4 shows the basic components of a spreadsheet formula.

There are two main aspects to a formula, namely the *formula structure* and the *arguments/terms*. The structure of a formula refers to the organisation of *binary operators* (+, -, /, * or ^) and/or *built-in spreadsheet functions* in the formula. An argument/term in the formula is either represented in the form of a *cell address* or a *constant*. When a cell address is used the formula reads the value entered in the corresponding location on the spreadsheet. If this value changes, the result value of the formula changes accordingly, although the actual formula itself remains unchanged.

2.3.2. Hierarchical representation of a spreadsheet formula in *tree* form. In the same way that a computer program can be broken down into smaller parts and represented in the form of a tree, elements of a spreadsheet formula (data values and other formulae) can also be modelled diagrammatically in the form of a tree. This is important as it can be used to confirm our understanding of the structure and components of a particular formula as well as show its relationship with other formulae in the model. It can also be useful as a means of documenting the design of a spreadsheet model.

All types of formulae can be represented in the form of a tree, including the spreadsheet (e.g. *MS Excel*) built-in functions. The general format of a function is as follows [8]:

= name (argument1, argument2 ...)



	A	B	C
1	**T Howe Ltd**		
2	***Trading and Profit and Loss Account for the year ended 31 December 19X4***		
3			
4	Sales		135,486
5	*Less* Cost of goods sold		
6	Opening stock	40,360	
7	*Add* Purchases	72,360	
8	*Add* Carriage inwards	1,570	
9		114,290	
10	*Less* Closing stock	52,360	61,930
11	Gross profit		73,556
12	*Less* Expenses		
13	Salaries	18,310	
14	Rates and occupancy	4,515	
15	Carriage outwards	1,390	
16	Office expenses	3,212	
17	Sundry expenses	1,896	
18	Depreciation: Building	5,000	
19	Equipment	9,000	
20	Directors' remuneration	9,500	52,823
21	Net profit		20,733
22	*Add* Unappropriated profits from last year		15,286
23			36,019
24	*Less* Appropriations		
25	Proposed dividend	10,000	
26	General reserve	1,000	
27	Foreign exchange	800	11,800
28	Unappropriated profits carried to next year		24,219

Figure 2. Model with an *ambiguity* problem

`name' is the function name, and `argument1', 'argument2', etc., are the arguments required for the evaluation of the function. Arguments must appear in a parenthesised list as shown above and their exact number depends on the function being used. However, some functions do not require arguments and are used without parentheses [1].

The tree represents all the elements of a particular formula (data values, hard-coded constants, cells referenced, binary operators and built-in functions). Based on the spreadsheet model shown in figure 3, it can be seen how formulae can be represented in the form of a tree to facilitate comprehension, analysis and documentation. This is shown in figure 5.

Figure 5 represents the logical aspect of the spreadsheet model, independent of physical location on the spreadsheet. Tree structures can also be constructed on formulae to show the physical model using cell addresses. Examples of this [1] are given in figure 6.

As all functions are of the same form, = **name (argument1, argument2 ...)**, we can represent each in the form of a tree (not necessarily a binary tree). The root

would now contain the function name while each argument would form a node.

to the fact that data values and their referencing formulae are placed close to each other and users are sometimes unable to distinguish between the two. This means that the cause

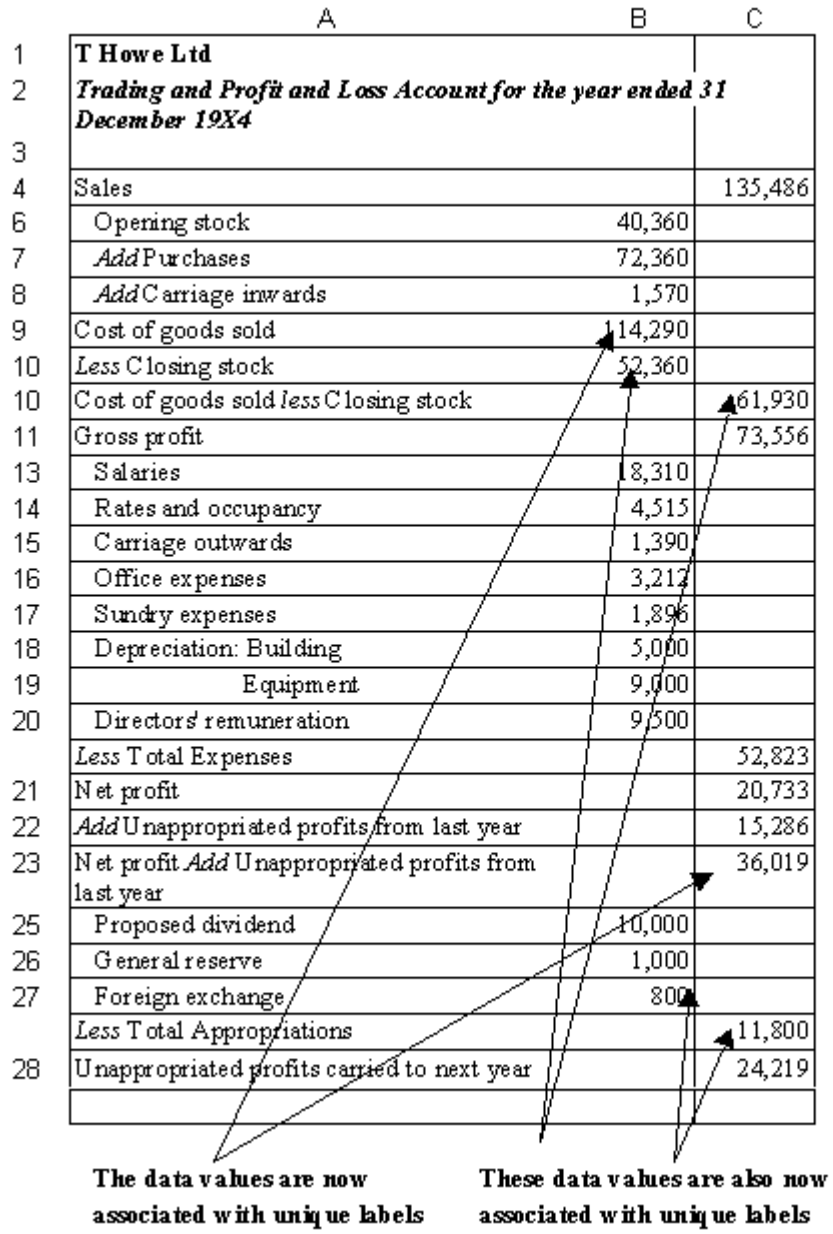


Figure 3. The *ambiguity* problem resolved

2.4. Separation of data and operations

It is important to control data integrity and maintain the consistency of data in information systems. A frequent error committed by users is the accidental overwriting of formulae. Formulae are important and must be protected from such undesirable events. This is mainly attributable is the physical proximity of data values and formulae performing operations.

The strategy to be adopted is to separate user-entered data values from the formulae that operate on them. The user-entered data values, along with their corresponding

labels are placed in a physically isolated matrix. The operations part of the model, consisting of formulae are

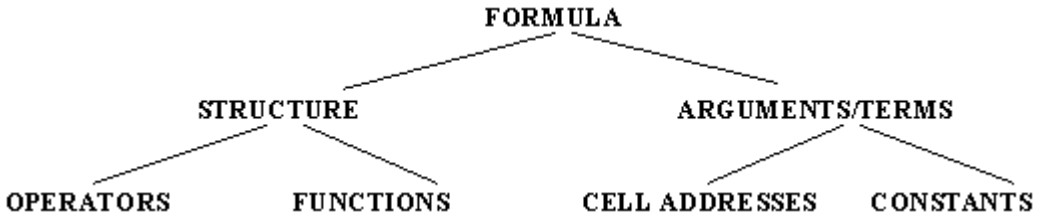


Figure 4. Components of a spreadsheet formula

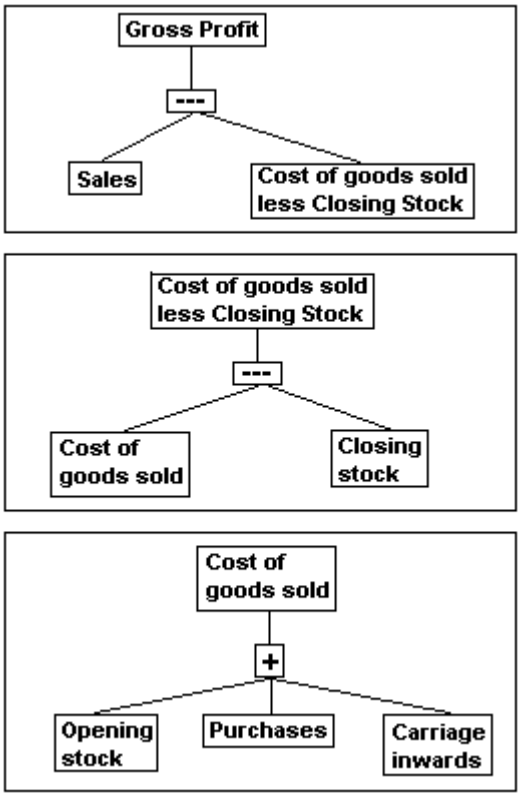


Figure 5. Selected formulae in *tree* form

placed in a another part of the spreadsheet. After they are programmed, the operations part is protected as a pre-caution against any overwriting of data.

This approach is very similar to the technique proposed by DiAntonio [9]. DiAntonio proposes the splitting of the spreadsheet model into a *facts matrix* and a *solution matrix*. The facts matrix is similar to our data component while the solution matrix is fundamentally similar to our operations component. In addition to that, the operations section is also strictly protected.

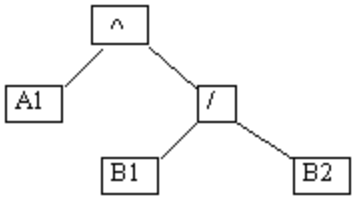
There is, however, one obvious and potentially serious problem with this method. The operations component is difficult to understand now. It is incomplete without the relevant data values put in the appropriate adjacent cells.

The strategy that can be used to solve this problem would be to include absolute copies of data values from the data component in appropriate places in the operations component. As such, the absolute copy is present as a formula in the operations component. There can even be multiple copies of a piece of data value provided they all reference the same original data value in the data component. The refined model is shown in figure 7. It is based on the spreadsheet model shown in figure 3.

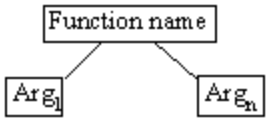
Examples

E.g. 1. = A1+B1 2. = A1/B1 i.e. A1÷B1

3. = A1^(B1/B2) i.e. $\sqrt[B1]{A1}_{B2}$



As all functions are of the same form, = **name (argument1 , argument2 ...)**, we can represent each in the form of a tree (not necessarily a binary tree). The root would now contain the function name while each argument would form a node.



E.g. 4. A1=SUM(B3:D5, B4)

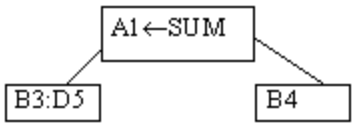


Figure 6. Examples of formulae in *tree* form

3. A coherent software engineering-based methodology

3.1. Rationale for the methodology

Many authors who have carried out research into the problem of spreadsheet errors have reached the conclusion that the main reason behind this prevailing phenomenon is the absence of a structured approach. Many of them have also proposed the adoption of a software engineering-based approach, employing traditional programming principles. This is evident from numerous publications on the subject [1].

According to Kavanagh [10], companies are being put at risk due to their failure to realise that the process of constructing spreadsheets requires the discipline of traditional programming. Spreadsheet applications are more vulnerable to poor design and to errors than conventional programs [11]. This means that a greater degree of discipline is required in the process of spreadsheet development. A software engineering-based methodology is capable of providing such discipline.

Panko and Halverson [2] have also indicated that when dealing with complex spreadsheets, there is a need to adopt strict programming disciplines. They have also said that a programming approach might be feasible due to the

Data Component **Operations Component (*protected*)**

	A	B	C
1	**T Howe Ltd**		
2	***Trading and Profit and Loss Account for the year ended 31 December 19X4***		
3	Sales	135,486	
4	Cost of goods sold: Opening stock	40,360	
5	Purchases	72,360	
6	Carriage inwards	1,570	
7	Closing stock	52,360	
8	Expenses: Salaries	18,310	
9	Rates and occupancy	4,515	
10	Carriage outwards	1,390	
11	Office expenses	3,212	
12	Sundry expenses	1,896	
13	Depreciation: Building	5,000	
14	Equipment	9,000	

	A	B	C
23	**T Howe Ltd**		
24	***Trading and Profit and Loss Account for the year ended 31 December 19X4***		
25			
26	Sales		135,486
27	Opening stock	40,360	
28	*Add* Purchases	72,360	
29	*Add* Carriage inwards	1,570	
30	Cost of goods sold	114,290	
31	*Less* Closing stock	52,360	
32	Cost of goods sold *less* Closing stock		61,930
33	Gross profit		73,556
34	Salaries	18,310	
35	Rates and occupancy	4,515	
36	Carriage outwards	1,390	
37	Office expenses	3,212	
38	Sundry expenses	1,896	
39	Depreciation: Building	5,000	
40	Equipment	9,000	
41	Directors' remuneration	9,500	

Figure 7. A segment of the refined model

similarity between spreadsheet errors and programming errors.

Hendry and Green [12] have suggested that instead of creating the whole spreadsheet first and then checking for errors, errors ought to be checked for at various stages of the development process. This will make it easier to trace and correct errors. This strategy of stage-by-stage component testing is a software engineering principle [6].

Based on these published reports, there is an obvious and urgent need to apply the principles of software engineering in spreadsheet development in order to enhance the integrity and reliability of the models produced. This will help address the currently major phenomenon of spreadsheet errors [1].

3.2. The methodology step-by-step

The methodology consists of a step-by-step process to the construction of spreadsheet models incorporating the software engineering concepts and principles discussed in the previous section.

Step 1:

The spreadsheet problem is analysed thoroughly, mainly in terms of the operations required to serve the purposes of the model. All the formulae performing these operations in the spreadsheet model are identified.

Step 2:

Based on the technique of *hierarchical decomposition*, each formula is organised in a hierarchical form. This hierarchy takes the form of a *tree*.

Step 3:

All user-entered data values are identified. Most of these data values are referenced by some formula or several formulae. The *formula trees* can also be helpful in determining the data values. These data values are then organised in the form of a matrix and physically separated on the spreadsheet. This is based on the technique of *separation of data and operations* on the spreadsheet.

Each element of the data matrix is uniquely described by a label or a pair of column and row labels. This is based on the technique of *unique definition of spreadsheet elements* discussed earlier.

Step 4:

The various formulae operating on the data values are organised into a separate *operations section* along with absolute copies of data values (where appropriate) in order to provide completeness and understandability to the model. As done previously in *step 3*, each element in the operations part is uniquely described by a label or a pair of column and row labels.

Step 5:

The operations part of the model is completely protected from users. Users are only allowed to alter the contents of the data matrix when necessary. The elements of the operations section can only be modified by authorised spreadsheet programmers by having them 'unprotected' first and then protected again after the changes have been made.

4. Conclusion

This paper has presented work presently in hand towards a methodology for increasing the integrity of spreadsheet models. The proposed methodology consists of various techniques, guidelines and rules governing the process of spreadsheet design and development.

A significant feature of this approach is that it adopts concepts from software engineering and employs important principles and techniques such as unique definition of spreadsheet model elements, tree-based hierarchical decomposition, and separation of data and operations.

The principles and techniques put forth in this paper have the potential to eliminate and reduce the occurrence of many types of spreadsheet errors. The unique definition of elements of a spreadsheet model prevents any ambiguity concerning the meaning of a particular data value or formula. This prevents any misinterpretation of the model, which could subsequently lead to various types of errors.

The tree-based representation of a spreadsheet formula enhances understanding of the structure of and relationship between formula operations. Formulae lie at the heart of spreadsheet modelling and must be treated with care and caution. They are also a major source of errors. The tree-structure also helps identify any omitted or redundant elements in the formula.

The technique of separation of data and operations is one which is not alien to structured approaches, where the data is defined and held separately from the functions or methods that operate on them. This technique applied in spreadsheet design and development ensures that formulae are protected from accidental overwriting. Data integrity can also be maintained as each piece of data value has a single point from which it is referenced by its dependant formulae and copies.

The various features of this approach, as detailed in this paper, are currently being trialled with cohorts of students at the University of Greenwich. They form the basis of further work towards a working methodology. Such a methodology would represent a major step towards managing the quality of spreadsheets.

5. References

[1] K. Rajalingham, D. Chadwick, B. Knight, and D. Edwards, “An Approach to Improving the Quality of Spreadsheet Models", *Proceedings of the Seventh International Conference on Software Quality Management SQM'99*, Southampton, UK, British Computer Society, March 1999, pp. 117-131.

[2] R.R. Panko and R.P. Halverson, Jr., "Spreadsheets on Trial: A Framework for Research on Spreadsheet Risks", *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, Maui, Hawaii, January 1996.

[3] KPMG Management Consulting, “Executive Summary: Financial Model Review Survey”, KPMG, London, 1997.

[4] M. Ward, "Fatal Addition", *New Scientist*, 16th August 1997.

[5] D. Chadwick, J. Knight, and P. Clipsham, " Information Integrity In End-user Systems", *Proceedings of the First Annual IFIP TC-11 Working Group 11.5 Working Conference on Integrity and Internal Control in Information Systems*, Zurich, Switzerland, Chapman & Hall, December 1997.

[6] K. Rajalingham and D. Chadwick, "Integrity Control of Spreadsheets: Organisation & Tools", *Proceedings of the Second Annual IFIP TC-11 Working Group 11.5 Working Conference on Integrity and Internal Control in Information Systems*, Virginia, USA, Kluwer Academic Publishers, November 1998, pp. 147-168.

[7] F. Wood, *Business Accounting 1 (7th Edition)*, Pitman Publishing, 1996.

[8] N. Kantaris and & P.R.M. Oliver, *Excel 5 Explained*, Bernard Babani (publishing), 1994.

[9] A.E. DiAntonio, *Spreadsheet Applications*, Prentice-Hall, 1986.

[10] J. Kavanagh, "Shoddy Business Models Breed Financial Disaster", *Computer Weekly*, 19 June 1997.

[11] S.J. Davis, "Tools for Spreadsheet Auditing", *International Journal of Human-Computer Studies* (Vol 45), 1996, pp.429-442.

[12] D.G. Hendry and T.R.G. Green, "Creating, Comprehending, and Explaining Spreadsheets: A Cognitive Interpretation of What Discretionary Users Think of the Spreadsheet Model", *International Journal of Human-Computer Studies* (40:6), June 1994, pp.1033-1065.