Controlling the Subversive Spreadsheet
Risks, Audit and Development Methods

5th - 6th July 2001
Vrije Universiteit
Amsterdam, The Netherlands
European Spreadsheet Risks Interest Group

EuSpRIG 2001 Symposium

"Controlling The Subversive Spreadsheet: Risks, Audit and Development Methods"

Editors:
David Chadwick and Leon Strous

Symposium Hosted By

vrije Universiteit Amsterdam
Faculty of Economics and Business Administration
Postgraduate EDP-Audit Programme

July 2001
Amsterdam, The Netherlands
PREFACE

Well, who’d have thought it? EuSpRIG all grown up and holding its second conference? And a truly international one at that?

Since the infant EuSpRIG first took its first faltering steps at an informal meeting of two academics and a tax inspector an awful lot has happened. First, of course, was formalising three guys in a bar into a proper learned society. The support and commitment of all at the initial meeting last year was wonderful, and has been carried forward to the present.

The first conference - The worry and doubt - Will anyone come? Will anyone submit a paper? - Well, that was an enormous success of course, and we had to stretch the definition of “European” a little to encompass Hawaii and Singapore. Since then we’ve had one or two spreadsheet suppliers sit up and take notice, and the proceedings are getting quoted in the press to a surprising extent.

And now to 2001 - What a line-up! - We have papers from the UK to New Zealand, from academics, auditors and business consultants - and even one from an undergraduate and his supervisor, all addressing the theme “Controlling the Subversive Spreadsheet”. At our general meeting, after the conference we will be considering our name - The past two years show that “European” is just not appropriate.

All this success is of course based on hard work, and I must pay tribute to (in alphabetical order) David Chadwick, Pat Cleary and Leon Strous who have worked so hard to ensure the success of this year’s conference. Thanks also to our sponsors, PricewaterhouseCoopers, the Vrije Universiteit in Amsterdam and the Information Systems Audit and Control Association Northern UK chapter, whose financial and material support has been crucial in setting up the conference. Not to forget the supporters who informed their members about this event. Thanks too of course to all the contributors - the visible side of the proceedings - and to the small army of reviewers, who have helped assure the quality of the volume you hold.

This is my last conference as Chair of EuSpRIG. My thanks to all the committee members, supporters and all others who have helped it grow. My very best wishes to all for the future.

May 2001

Ray Butler, CISA
Chairman, EuSpRIG
FOREWORD

EuSpRIG was founded in March 1999 when researchers from ISACA (Northern UK Chapter), University Wales Institute Cardiff and the University of Greenwich came together to discuss the ever increasing problem of business risk associated with spreadsheet errors. As a result of this meeting an interest group was founded dedicated to increasing awareness of the problem amongst academia and industry and to promoting research regarding the extent and nature of the problem, methods of prevention and detection of errors and methods of limiting damage. This has brought together researchers and professionals in the areas of business, software engineering and audit to actively seek useful solutions.

In the short time of its existence the group has grown and now includes interested parties from HM Customs and Excise, KPMG Management Consulting, PricewaterhouseCoopers, Anderson Consulting, the British Computer Society (BCS) and the Dutch Computer Society (NGI) as well as many other companies and individuals. In July 2000, as part of this co-operative initiative the founding members organised a symposium to bring together all interested parties to discuss the issues involved. The symposium, held at the University of Greenwich in London, was a great success, with some forty attendees from all walks of industry and academia. This initial interest is further developed during this year's symposium in Amsterdam.

It is surprising to learn how many important business decisions are made based on information provided by spreadsheets. It is worrying to know how many errors these spreadsheets (can) contain. The objective of the symposium is to promote discussion and co-operation amongst those concerned with authorising, auditing and developing spreadsheet models and by so doing, improve the reliability and integrity of information portrayed on spreadsheets. The papers cover a broad spectrum of practical experience and research. The topic areas include the types of errors, development methodologies, audit practice and tools.

June 2001

David Chadwick
Programme chair

Leon Strous
Organizing chair
EuSpRIG Committee

Raymond Butler, chairman, HM Customs & Excise
Patrick Cleary, secretary, University of Wales Institute Cardiff
David Chadwick, treasurer, University of Greenwich
Grenville Croll, member, Andersen's
Graham MacDonald, member, HM Customs & Excise
Robert Li, member, KPMG
Barry Pettifor, co-opted member, PricewaterhouseCoopers
Leon Strous, co-opted member, NGI

MORE INFORMATION

More information about Eusprig, its activities and projects can be found on

http://www.gre.ac.uk/~cd02/EUSPRIG
EuSpRIG Second Symposium

Controlling The Subversive Spreadsheet: Risks, Audit and Development Methods

SYMPOSIUM PROGRAMME COMMITTEE

David Chadwick (Chair), University of Greenwich, London
Professor Margaret van Biene-Hershey, Vrije Universiteit, Amsterdam
Raymond Butler, H M Customs & Excise, Liverpool
Harmen Ettema, PricewaterhouseCoopers, Amsterdam
Luc Kordel, Dexia Bank, Brussels
Robert Li, KPMG, London
Professor Ray Panko, University of Hawaii, Honolulu
Barry Pettifor, PricewaterhouseCoopers, London

SYMPOSIUM ORGANIZING COMMITTEE

Leon Strous (Chair), De Nederlandsche Bank / NGI, Amsterdam / Utrecht
Patrick Cleary, University of Wales Institute Cardiff, Cardiff
Carla Heldens, Vrije Universiteit, Amsterdam

SYMPOSIUM CHAIRPERSONS

Prof. Margaret van Biene-Hershey (Vrije Universiteit)
Raymond Butler (HM Customs & Excise)
Harmen Ettema (PricewaterhouseCoopers)

SYMPOSIUM SESSION CHAIRPERSONS

Margaret van Biene-Hershey (Vrije Universiteit)
Raymond Butler (HM Customs & Excise)
David Chadwick (University of Greenwich)
Pat Cleary (University of Wales Institute Cardiff)
Patrick O’Beirne (Systems Modelling Ltd.)
Barry Pettifor (PricewaterhouseCoopers)
Leon Strous (De Nederlandsche Bank / NGI)
SPONSORS

PricewaterhouseCoopers (main sponsor)

Vrije Universiteit, Postgraduate EDP-Audit Programme (sponsor and host)

Information Systems Audit and Control Association (ISACA) Northern UK Chapter

SUPPORTERS

BCS CASG (British Computer Society Computer Audit Specialist Group)

ISACA BeLux and NL Chapters

NGI (Nederlands Genootschap voor Informatica / Dutch Computer Society)

NOREA (Nederlandse Orde voor Register-EDP-Auditors)

University of Greenwich

University of Wales Institute Cardiff
# TABLE OF CONTENTS

## PREFACE

### Session 1. Risks and the need for standards
Chairperson: Pat Cleary, University of Wales Institute Cardiff, UK

- "Euro conversion in spreadsheets"
  *Patrick O'Beirne, Systems Modelling Ltd., Ireland*  
  3

- "Applying the Cobit framework to spreadsheet development"
  *Ray Butler, H.M. Customs & Excise, UK*  
  7

### Session 2. Development methodologies and techniques
Chairperson: Ray Butler, H.M. Customs & Excise, UK

- "Safer spreadsheets with model master"
  *Jocelyn Paine, University of Bristol, UK*  
  17

- "An evaluation of the quality of a structured spreadsheet development methodology"
  *Kamalasen Rajalingham / David Chadwick / Brian Knight, University of Greenwich, UK*  
  39

- "New Guidelines for writing spreadsheets"
  *John Raffensperger, University of Canterbury, New Zealand*  
  61

### Session 3. Audit and alternative approaches
Chairperson: Barry Pettifor, PricewaterhouseCoopers, UK

- "Spreadsheet auditing"
  *David Nixon / Mike O'Hara, University of Salford, UK*  
  79

- "Peer audit and self-audit for reducing errors"
  *David Chadwick / Rodney Sue, University of Greenwich, UK*  
  95

- "Spreadsheet assurance by 'control around' is a viable alternative to the traditional approach"
  *Harmen Eitema / Paul Janssen / Jacques de Swart, PricewaterhouseCoopers, The Netherlands*  
  107
Session 4. Audit and alternative approaches (continued)
Chairperson: Patrick O’Beirne, Systems Modelling Ltd., Ireland

“A real alternative to spreadsheets”
Dane Knight, Brixx, UK
Session 1.

Risks and the need for standards

Chairperson: Pat Cleary
University of Wales Institute Cardiff, UK

Papers in this session:

“Euro conversion in spreadsheets”
*Patrick O'Beirne, Systems Modelling Ltd., Ireland*

“Applying the Cobit framework to spreadsheet development”
*Raymond Butler, H.M. Customs & Excise, UK*

Quote: "Business managers and auditors urgently need two (linked) products. These are:
- A brief synopsis of spreadsheet risks, to explain why they should take spreadsheets in their organisation seriously;
- A statement of good practice in the design, use and control of spreadsheets.”
(Raymond Butler)
EURO CONVERSION IN SPREADSHEETS

Patrick O’Beirne
Systems Modelling Limited
Tara Hill, Gorey, Co. Wexford, Ireland
pobeirne@sysmod.com

ABSTRACT

This paper describes risks and mitigation approaches for the challenge of converting millions of spreadsheet files in the Eurozone by 1.1.2002 at the latest. The usefulness and limitations of conversion tools are described along with two examples of faulty converters. Good practices are recommended with specific recommendations for conversion, reformatting, redesign, and some environmental issues.

1. INTRODUCTION

Thousands of millions of end-user spreadsheet and word processor files in the Eurozone will have to be converted to the euro currency by 1.1.2002 at the latest. Spreadsheet files ("worksheets") are a very important store of financial data. They may contain prices, quotations, contracts, payroll, interest and depreciation calculations, plans, budgets, financial report consolidations, and historical data. Worksheets often duplicate some information that is also recorded elsewhere and was copied in by typing, copy & paste, programming code, database add-ins, and macros. However, modifying the original data source will usually not update the same information in the worksheet. Therefore, there is a high risk of creating inconsistencies between worksheets and other information systems. All this is not immediately obvious on reading a spreadsheet as they are rarely documented.

2. CONVERSION TOOLS

Conversion tools exist for spreadsheets and word-processor documents. See the euro FAQ at http://www.sysmod.com/eurofaq.htm for a list of suppliers. A limited version of IBM's FIS converter is available free at http://www.ibm.com/euro/. But all automatic converters suffer from the fundamental problem of not knowing which numbers represent quantities and which represent amounts. Money cells are rarely formatted as such; they may not even be formatted as 2 decimals, depending on the country. The argument of whether conversion tools are more safely used by I.T. people or by users will, in most organisations, be decided by the simple fact that the I.T. personnel resources are only enough to verify the highest risk conversions. These are the worksheets with the most business-critical impact, or those developed by high-status individuals such as the Managing Director or the Financial Controller.

Current tools have the ability to select areas for conversion, retain original values and/or formats, add cell annotations, and even create copy worksheets, to assist the auditing of the results. Testing spreadsheets is a whole topic in itself - see
http://www.sysmod.com/spreads.htm for links on spreadsheet errors and auditing. Be careful to test converters thoroughly in-house before letting them loose on all the users.

3. CAUTIONARY TALES

I was sent a euro converter for evaluation. I installed it on a Windows 98 system, did a quick calculation check in MS Excel, and it looked OK. However, on exiting Excel, the "Do you want to save" dialog box had been corrupted - for example, a "Cancel" button performed the Save action! When I reported this to the authors, they at first denied that their product could cause the problem. Then I tracked it down to a dynamic link library (DLL) dated 1996 that their installation procedure had placed in my Windows 98 \\windows\system directory. When I told them about that, they actually tried to pass the blame on to MS saying: "the behaviour was caused by a faulty component (MSO97.DLL provided by Microsoft Corporation)" The truth is, the DLL was not faulty in itself, they had supplied an out-of-date version for that installation. They have something to learn about installation management and version checking.

I tried out another desktop euro converter and found that it correctly converted figures in text in MS Word, but rounded some results incorrectly in Excel. The author tracked it down to a third party math library routine that they had used to build the converter. So the lesson is to test the tools!

4. GOOD SPREADSHEET PRACTICES

Starting from now, users can put into place the good practices that will make life easier in future. The first is to put constant factors and money input values into cells of their own, for ease both of identification and of change. If these cells are referred to more than once, give them range names so that formulas can have descriptive entries such as =Sale_Price*euro_to_iep_rate rather than the more cryptic cell references.

To avoid converting again a worksheet that has already been converted, you should identify the currency unambiguously so you know what's converted and what's not.

If it is known that every important worksheet will be touched in the time before the changeover to the euro, the following approach shows how to prepare the spreadsheet for the ultimate changeover.

5. FORMAT AS YOU GO

Assuming that users have at least a year to go before general euro conversion is required internally, the idea is that people take the time to format each currency input cell as "money" as they use them. Over the period of a year, all sheets in use will get flagged. For good practice, one agreed cell - such as A1 in the first sheet you see when the workbook is opened - should contain an annotation "currency flagged by POB on 28 July 2000" or similar.
At the end of a business cycle, say one year, all the used spreadsheets have been touched. The exceptions include those used only as external references, which have not been updated in a year. They will have to be checked by someone who knows the design of the system of linked spreadsheets.

When ready to change, use a conversion tool that can operate on cells selected according to their format. Or if you know how, you could write the macro yourself that performs the following change on all the numeric cells of a workbook. If the format property is "Money", and the cell does not contain a formula, the macro divides the value by the fixed conversion factor, and changes the style to "Euro". The user will have to decide what is appropriate for rounding. There may be a lot of cells with two distinct types - amount of money for payment ("Money", rounded to 2 decimals) and rates of charging "Money_Rate" say), which might not be rounded.

6. START AFRESH

There may be even simpler ways. Many people start the year with a blank standard template that fills up as the year progresses. In that case, all they have to do is create a new template, which may be no different from the original except that it is explicitly designated as being in euro. If some data needs to be carried forward, it may be simpler to create an extra column or group of cells in the final saved version of one year's sheet that contains the converted values, and copying and pasting those values into the next year's sheet.

If such redesign is being considered, take the time to implement another important principle of good worksheet design: designate an input area that keeps all input data together; and another area, with a range name, for data that is to be transferred in block to another worksheet.

If it is required to show parallel sets of data for both NC (National Currency) and euro, that requires a restructuring of the worksheet anyway. So the conversion can be built in then.

7. THE EURO SYMBOL (€)

Excel 6, 95(7) and 97 users will be aware that the default table of three-letter-currency-abbreviations in Excel does not contain the ISO code EUR, only the obsolete XEU. It can be added using the Windows Control Panel Regional Settings/Currency tab/Currency symbol. The whole area of font and euro symbol support for Windows applications is covered in an article by John Gray entitled "Notes on the Euro Symbol" which can be downloaded from the Systems Modelling Ltd. web site http://www.sysmod.com/euronotes.pdf (409K).

Applying the CobiT® Control Framework to Spreadsheet Developments

Raymond J Butler, CISA
HM Customs & Excise Computer Audit Service National Office, Queens Dock, Liverpool L74 4AA UK

{ ++44 (0)151 703 8741 ray.butler@hmce.gsi.gov.uk

©Crown Copyright reserved, used by permission.

Extracts from CobiT® © 1996, 1998, 2000 by the Information Systems Audit and Control Foundation, reprinted by permission of the Information the Systems Audit and Control Foundation and IT Governance Institute. No other right or permission is granted with respect to this work.

ABSTRACT

One of the problems reported by researchers and auditors in the field of spreadsheet risks is that of getting and keeping management’s attention to the problem. Since 1996, the Information Systems Audit & Control Foundation and the IT Governance Institute have published CobiT® which brings mainstream IT control issues into the corporate governance arena. This paper illustrates how spreadsheet risk and control issues can be mapped onto the CobiT framework and thus brought to managers’ attention in a familiar format.

1. A BRIEF INTRODUCTION TO COBIT®

1.1. What is CobiT?

CobiT®, Control Objectives for Information & related Technology is a tool set which helps business managers to understand and manage the risks associated with implementing new technologies, and demonstrate to regulators, shareholders and other stakeholders how, and how well they have done this. It is based on international best practice in IT management and control.

The tool set facilitates IT governance, defined as “a structure of relationships and processes to direct and control the enterprise in order to achieve the enterprise’s goals by adding value while balancing risk versus return over IT and its processes” [ISACF 2000(1)] In an age where business is almost entirely dependent on technology, IT Governance is an essential element of wider corporate governance.

1.2. CobiT’s Contents

The framework defines

- 34 IT processes in 4 broad groups. These processes depend on and impact on IT resources.
- High-level control objectives for each of the 34 processes,
- 318 detailed control objectives, and associated audit guidelines.

CobiT also contains Management Guidelines, including Maturity Models, Critical Success Factors, Key Goal Indicators and Key Performance Indicators for each of the 34 processes.

1.3. CobiT’s Audience: Management, Users And Auditors

The framework is designed to help three distinct audiences:
Management – who need to balance risk and control investment in an IT environment which is often unpredictable.

Users – who need to obtain assurance on the security and controls of the IT services they depend on to deliver their products and services to internal and external customers.

Auditors – who can use it to substantiate their opinions and / or provide advice to management on internal controls.

Apart from responding to the needs of the immediate audience of senior management, auditors and security and control professionals, CobiT can be used within enterprises by business process owners in meeting their responsibility for control over the information aspects of their processes and by those responsible for IT in the enterprise.

2. HOW DOES COBIT COVER SPREADSHEET RISKS?

No specific mention is made of Spreadsheets, or of end-user computing. Instead, CobiT provides a generic framework for all the principal IT processes. These can be adapted, scaled and applied to IT solutions at all levels, from a whole Enterprise Resource Planning system to a (relatively) simple spreadsheet development. The example below shows how this can be done.

2.1. An Example from CobiT

The high-level control objective for the process defined as “Acquire and Maintain Application Software” states [ISACF 2000(2)] that

“Control over the IT process of acquiring and maintaining application software that satisfies the business requirement to provide automated functions which effectively support the business process is enabled by the definition of specific statements of functional and operational requirements, and a phased implementation with clear deliverables, and takes into consideration

- functional testing and acceptance
- application controls and security requirements
- documentation requirements
- application software life cycle
- enterprise information architecture
- system development life cycle methodology
- user-machine interface
- package customisation”

This is supported by 17 detailed control objectives covering

- Design Methods
- Major Changes to Existing Systems
- Design Approval
- File Requirements Definition and Documentation
- Programme Specifications
- Source Data Collection Design
- Input Requirements Definition and Documentation
- Definition of Interfaces
- User-Machine Interface
- Processing Requirements Definition and Documentation
- Output Requirements Definition and Documentation
- Controllability
- Availability as a Key Design Factor
- IT Integrity Provisions in Application Programme Software
All of these controls can be scaled and applied to spreadsheet development. In our “ideal environment”, some of these (such as design approval, and testing) will require specific formal controls and procedures, some (such as availability) may depend on wider Office Technology platform controls.

3. THE MATURITY MODEL

CobiT’s maturity model for control over IT processes provides a method of scoring which enables an organisation to grade its IT control procedures on a scale from 0 (non-existent) to 5 (optimised). This approach has been derived from the Maturity Model for software development capability defined by the Software Engineering Institute. Management use the maturity model to map the current status of:

- their organisation,
- the best practice or the general standard,
- international standards

and define where the organisation wants to be against these levels. Figure 1 [ISACF 2000(2)] illustrates the rankings and the way in which an organisation can use the model to map the maturity of their current and desired practices onto the model.

4. APPLYING CobiT TO SPREADSHEETS

The framework can be readily scaled to spreadsheet developments. The following control objectives (high level and detailed) and maturity model is offered as:

- A demonstration of the adaptability of CobiT to spreadsheets, and
- a “first draft” upon which a formal set of overall and more detailed control objectives can be built.

The controls will obviously need to be applied only to the degree justified by the actual or potential impact that a spreadsheet model has upon the organisation in which it is used. A simple impact assessment and the contents of documentation, etc. have been described in previous papers by this author [Butler, 2000] and will not be reproduced here.

4.1. Control Objective

Control over the process of developing and maintaining spreadsheet models and applications that satisfy the business requirement to provide accurate and error-free business models and analyses which
effectively support the business process is enabled by the definition of specific statements of functional and operational requirements, and a phased implementation with clear deliverables, and takes into consideration

Design Methods

Security and data retention requirements

Testing and Acceptance

Documentation Requirements

4.2. Detail Control Objectives:

Design Methods

The organisation should employ a spreadsheet development methodology which requires that appropriate procedures and techniques, involving close liaison with model users, are applied to create the design specifications for each new spreadsheet development and to verify the design specifications against the user requirements.

Major Changes to Existing Systems

Management should ensure, that in the event of major changes to existing spreadsheet models or applications, a similar development process is observed as in the case of the development of new models.

Design Approval

The organisation's spreadsheet development methodology should require that the design specifications for all spreadsheet development and modification projects be reviewed and approved by management, the affected user departments and the organisation's senior management, when appropriate.

Programme Specifications

The organisation's spreadsheet development methodology should require that detailed written specifications be prepared for each spreadsheet development or modification project. The methodology should further ensure that specifications agree with design specifications.

Testing

Testing to ensure that:

- The spreadsheet calculations
- Data input and controls over data
- Links between host systems and the spreadsheet, between parts of the spreadsheet and between spreadsheets in a multi-file suite of models
- Output reports

operate correctly and as specified according to the development test plan and established testing standards should be performed and documented before the development is approved by the user. Adequate measures should be conducted to prevent disclosure of sensitive information used during testing.

User Documentation and instructions
The organisation's spreadsheet development methodology should provide that adequate user reference and support manuals be prepared (preferably in electronic format) as part of every spreadsheet development or modification project.

**Security and retention**

- The organisation's spreadsheet development and use methodology should include directions for ensuring that:
- Access to spreadsheet models is restricted to authorised persons;
- Spreadsheet models are protected against inadvertent or unauthorised modification;
- Spreadsheet models and applications are retained in electronic form for the period of time appropriate to the purpose of the spreadsheet.

**4.3. Maturity Model**

Control over the process of developing and maintaining spreadsheet models and applications that satisfy the business requirement to provide accurate and error-free business models and analyses which effectively support the business process.

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Non-existent</td>
<td>There is no process for designing and specifying spreadsheets. Typically, spreadsheets are developed in an unstructured manner by untrained end-users, with little or no documentation of actual requirements and no testing. There is an extremely high risk of error in important spreadsheets.</td>
</tr>
<tr>
<td>1 Initial/Ad Hoc</td>
<td>There is an awareness that a process for developing spreadsheets is required. Approaches, however, vary from development to development without any consistency and typically in isolation from each other. The organisation’s business depends upon a variety of individual solutions with varying degrees of documentation and control and now suffers legacy problems and inefficiencies with maintenance and support. There is a very high risk of errors in important spreadsheets.</td>
</tr>
<tr>
<td>2 Repeatable but Intuitive</td>
<td>There are similar processes for developing and maintaining spreadsheets, but they are based on the expertise within the users, not on a documented process. The success rate with spreadsheets depends greatly on individual users’ skills and experience levels. Maintenance is usually problematic and suffers when internal knowledge has been lost from the organisation. There is a high risk of errors in important spreadsheets.</td>
</tr>
<tr>
<td>3 Defined Process</td>
<td>There are documented development and maintenance processes. An attempt is made to apply the documented processes consistently across different spreadsheet developments, but they are not always found to be practical to implement. They are generally inflexible and hard to apply in all cases, so steps are frequently bypassed. As a consequence, spreadsheets are often developed and implemented in a piecemeal fashion. Maintenance follows a defined approach, but is often time-consuming and inefficient. There is medium risk of errors in important spreadsheets.</td>
</tr>
</tbody>
</table>
| 4 Managed and Measurable | There is a formal, clear and well-understood spreadsheet development and implementation methodology and policy that includes a formal design and specification process, a process for testing and requirements for documentation, ensuring that all spreadsheets are developed and maintained in a consistent manner. Formal approval mechanisms exist to
### Maturity Level Characteristics

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ensure that all steps are followed and exceptions are authorised. The methods</td>
</tr>
<tr>
<td></td>
<td>have evolved so that they are well suited to the organisation and are likely</td>
</tr>
<tr>
<td></td>
<td>to be positively used by all staff, and applicable to most important</td>
</tr>
<tr>
<td></td>
<td>spreadsheet developments. There is a low risk of errors in important</td>
</tr>
<tr>
<td></td>
<td>spreadsheets.</td>
</tr>
</tbody>
</table>

#### 5 Optimised

Spreadsheet development and maintenance practices are in line with the agreed processes. The development and maintenance process is well advanced, enables rapid deployment and allows for high responsiveness, as well as flexibility, in responding to changing business requirements. The spreadsheet development and implementation process has been subjected to continuous improvement and is supported by internal and external knowledge databases containing reference materials and best practices. The methodology creates computer based documentation in a pre-defined structure that makes production and maintenance very efficient. There is a very low risk of errors in important spreadsheets.

## 5. CONCLUSIONS

### 5.1. Can the CobiT Approach Help with Spreadsheets?

As illustrated above, the CobiT approach can easily be applied to spreadsheets. In itself, this adds no new insights into the problem of spreadsheet risk, or into good practice and control issues. It will, however be a very useful method of presenting those issues. It will allow spreadsheet risk, good practice and control to be presented to managers in a familiar format. It will therefore help the audit and spreadsheet development communities to “market” the issues to decision makers, and raise them as corporate and IS governance rather than parochial technical issues.

### 5.2. What Next?

Business managers and auditors urgently need two (linked) products. These are

- a brief synopsis of spreadsheet risks, to explain why they should take spreadsheets in their organisation seriously.
- A statement of good practice in the design, use and control of spreadsheets.

It is hoped that the proceedings of this and the previous EuSpRIG conference will provide much of the source material for this, and that this paper will influence its production on the CobiT format, increasingly familiar to and used by business managers, our intended audience.

## 6. RESOURCES AND REFERENCES

Much of CobiT is available as an open standard for download from the Information Systems Audit and Control Association web site at [www.isaca.org](http://www.isaca.org)

- CobiT 3rd Edition Executive Summary, IS Audit & Control Foundation, Chicago, Ill, July 2000 [ISACF 2000(1)]
- CobiT 3rd Edition Control Objectives, IS Audit & Control Foundation, Chicago, Ill, July 2000 [ISACF 2000(2)]
- CobiT 3rd Edition Management Guidelines, IS Audit & Control Foundation, Chicago, Ill, July 2000 [ISACF 2000(3)]
- Butler, R (2000) “Is this Spreadsheet a Tax Evader?”, proceedings of the 33rd Hawaii International Conference on System Sciences

---

Extracted from *Controlling the Subversive Spreadsheet – Risks, Audit and Development Methods*  

Copyright © 2001 European Spreadsheet Risks Interest Group [www.eusprig.org](http://www.eusprig.org)
Session 2.

Development methodologies and techniques

Chairperson: Raymond Butler
H.M. Customs & Excise, UK

Papers in this session:

"Safer spreadsheets with model master"
*Jocelyn Paine, University of Bristol, UK*

"An evaluation of the quality of a structured spreadsheet development methodology"
*Kamalasen Rajalingham / David Chadwick / Brian Knight, University of Greenwich, UK*

"New Guidelines for writing spreadsheets"
*John Raffensperger, University of Canterbury, New Zealand*

Quote: "Make your spreadsheet resemble a system that your reader already knows."
(John Raffensperger)
Ensuring Spreadsheet Integrity with Model Master

Jocelyn Paine,
Visiting Fellow, Institute of Learning and Research Technology, University of Bristol

http://www.ifs.org.uk/~popx/

ABSTRACT

We have developed the Model Master (MM) language for describing spreadsheets, and tools for converting MM programs to and from spreadsheets. The MM decompiler translates a spreadsheet into an MM program which gives a concise summary of its calculations, layout, and styling. This is valuable when trying to understand spreadsheets one has not seen before, and when checking for errors. The MM compiler goes the other way, translating an MM program into a spreadsheet. This makes possible a new style of development, in which spreadsheets are generated from textual specifications. This can reduce error rates compared to working directly with the raw spreadsheet, and gives important facilities for code reuse. MM programs also offer advantages over Excel files for the interchange of spreadsheets.

1 INTRODUCTION

Spreadsheets are alarmingly error-prone to write. To quote [Panko 2000], “given data from recent field audits, most large spreadsheets probably do contain significant errors”. The most recent audits he cites found errors in at least 86% of spreadsheets audited. In a 1997 feature entitled “Fatal Addition” [Ward 1997], New Scientist reported that 90% of the spreadsheets audited in a study carried out by Coopers and Lybrand were found to have errors. Given the billions of spreadsheets in use, this leaves the worlds of business and finance horribly vulnerable to programming mistakes. Studies show the chances of any given spreadsheet cell containing an error are somewhere between 0.3% and 3%, so that a spreadsheet of only 100 cells has about a 30% chance of having one error or more.

Creating the spreadsheet is not the only problem. Spreadsheets can be difficult to read, and hence to debug and maintain. Look at the example below - a small professionally-written educational model for calculating income elasticities, shown with formulae visible – and make your best attempt to read it:

<table>
<thead>
<tr>
<th>Income elasticity of demand</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Income elasticity</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>Income elasticity of demand</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>Income elasticity of demand</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Firstly, the use of cell locations – C9, C10 - in formulae makes them hard to understand, because the location’s name conveys no information about its purpose. This presumably increases the reader’s cognitive load, because of the time needed to match locations with their targets, and the need to hold information in short-term memory while doing so. Excel does provide facilities for naming cells, but many developers don’t use these, so trying to understand cell locations is a common readability problem.

Secondly, even in such a small spreadsheet, an annoying amount of horizontal scrolling is needed before one can see all the formulae. This is partly because the spreadsheet contains not only the formulae but also a lot of text and empty cells used for layout. User-interface design experts agree that horizontal scrolling should be avoided wherever possible. In a larger spreadsheet, the reader would have to search a considerable amount of screen area, both to find all the formulae, and to match the cell locations with their targets. It seems reasonable to assume that if the reader cannot see the entire spreadsheet in one go, he or she will have to remember parts of it, further adding to cognitive load. As one developer said: “I end up with vast numbers of notes on little bits of paper”. If the spreadsheet has more than one worksheet, flicking between them is an extra inconvenience.

A third difficulty, illustrated by the “Lazy Days” example below [Rajalingham et al 2000], is repeated formulae, due to a calculation being replicated over several time periods, departments, or other entities. This example contains identical calculations of total wage and average wage for each type of personnel:
Lazy Days Staff Budget Costs 1995-1996

<table>
<thead>
<tr>
<th>Staff Numbers</th>
<th>Basic Wages £</th>
<th>Overtime Wages £</th>
<th>Total Wages £</th>
<th>Average Wages £</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managers</td>
<td>1</td>
<td>17700</td>
<td>0</td>
<td>17700</td>
</tr>
<tr>
<td>Grade 1</td>
<td>3</td>
<td>45540</td>
<td>1400</td>
<td>46940</td>
</tr>
<tr>
<td>Grade 2</td>
<td>9</td>
<td>122340</td>
<td>2000</td>
<td>124340</td>
</tr>
<tr>
<td>Grade 3</td>
<td>12</td>
<td>102350</td>
<td>0</td>
<td>102350</td>
</tr>
<tr>
<td>Grand Totals</td>
<td>25</td>
<td>287930</td>
<td>3400</td>
<td>291330</td>
</tr>
</tbody>
</table>

There is much repeated structure here, but the fact that it is repeated is not explicit. The columnar layout and the nature of the application make this likely, but the reader needs to examine and compare all the formulae before being sure.

To overcome such problems, we are developing a tool, named Model Master or MM, that “decompiles” spreadsheets, generating concise specifications of their calculations. The example below shows a specification generated by the decompiler for the elasticity model:

```plaintext
attributes <
    new_quantity
    old_quantity
    new_real_income
    old_real_income
    demand_change
    real_income_change
    income_elasticity
    good_type
>
where

    demand_change = new_quantity / old_quantity - 1
    real_income_change = new_real_income / old_real_income - 1
    income_elasticity = demand_change / real_income_change
    good_type = if( income_elasticity > 0,
                     "So, this product is a normal good.",
                     "So, this product is an inferior good." )
```

This lists the variables or “attributes” represented by the cells, and the equations relating them. It gives an alternative view of the spreadsheet, which we believe to be valuable when checking for errors and when trying to understand spreadsheets written by other people.

The significant point about these listings is that they can be regarded as programs. Not only can we decompile existing spreadsheets into this form; we could go in the other direction and compile such programs into spreadsheets. We have in fact devised a complete programming language for describing spreadsheets, which we also call MM, and a compiler that translates MM programs into spreadsheets.

What are the benefits of generating spreadsheets from MM programs, rather than writing directly in Excel? One is that typing errors and other mistakes in an MM program are more likely to be detected than errors in the spreadsheet itself. This is because spreadsheets have little redundancy: if the programmer mistypes a cell location, or through a slip of the cursor accidentally types a formula into the wrong cell, the resulting code still means something to the spreadsheet. On the other hand, when programming MM, one works with names such as balance and start_time, rather than with cell locations. The programmer must give a list of all such names – to “declare” them – when writing the program. So if a typing error is made, MM can check the name against this list, realising that since it doesn’t occur there, there was an error.

As the example above demonstrates, MM programs consist of almost nothing other than lists of variables together with the equations or formulae relating them. So if the variable names are well chosen – total_wage rather than tot_w, hours_per_week and not hwpwk – the programs ought to be comprehensible to non-programmers, so long as they understand the business processes being modelled.
Although in some organisations, spreadsheets are regarded as private to their author and not to be checked by anyone else, others may find this useful in managing development.

Another benefit of MM is code reuse. MM programs can be divided up into separate modules. These can be put into program libraries and included in a program just by mentioning the name of the file where they are stored. Authors can therefore build up a central repository of modules, reducing duplicated work and the risk of cut-and-paste errors. We regard this as extremely important, and emphasise that it is not a trivial claim. This feature of MM stems from mathematical techniques used in designing the OBJ family of algebraic specification languages [Goguen and Tracz 2000], and provides the simplest and most comprehensive module system that we know of.

MM makes possible a style of spreadsheet development similar to that practised with conventional languages such as Fortran and Java. In this, the MM program is primary, and the spreadsheet is merely a means of running it and displaying the results: something to be generated whenever one needs to do a calculation, but otherwise to be ignored. This is extremely different from the way in which most Excel programmers work, and we suspect few would be willing to give up the convenience of interactive model construction and testing possible when you work directly with the spreadsheet. However, because of MM’s benefits, particularly with respect to code reuse, we believe it important to find out how it can best be used to complement and increase the integrity of this kind of development. We are therefore continuing work on the compiler and decompiler, and on an integrated user interface which allows them to be invoked directly from Excel.

We also believe that MM programs, whether written from scratch or decompiled out of an existing spreadsheet, are a better medium for interchanging spreadsheets than are Excel XLS files. We are investigating MM for this purpose.

In the rest of this paper, we describe the MM language, compiler, and decompiler from the user’s point of view, and discuss the benefits of MM as a medium for spreadsheet interchange. We then give some details of the implementation, and finally summarise our progress and the work yet to be done. We start with an account of the language: without this, it is difficult to understand the compiler and decompiler. The main point of the paper is to present MM in enough detail that we may receive worthwhile criticisms and suggestions for improvement.

2 THE MM LANGUAGE AND COMPILER

2.1 Attributes, Objects, and Equations

MM views the world in terms of attributes which have values, usually numbers or strings, and which are interrelated by equations. Attributes may be single-valued, or may range over regions of time or space, with a value for each point in the region. It is sometimes convenient to think of them as grouped together to make objects. The simple example below is an MM description of a company:

```plaintext
attributes <
  incomings [ 1995:2004 ]
  outgoings [ 1995:2004 ]
  profit [ 1995:2004 ]
>
where
  profit[ all t ] = incomings[ t ] - outgoings[ t ]
```

(We do realise that as a business application, this program is trivial. This is not a case of the “toy language” syndrome, whose sufferers design languages that won’t scale up to realistic applications; we are using MM to build real models. But we do need to simplify so that we don’t obscure key concepts.)

To generate a spreadsheet, we pass the program text to the MM compiler, which checks for errors, and if none are found, produces an Excel file. Were we writing the spreadsheet directly, we would almost certainly use successive rows to represent successive years, allocating attributes to adjacent columns. This is what the compiler does, heading each column with the identifier used for its attribute. Appendix 1 shows a generated spreadsheet in the queuing simulation example.

2.2 Layout

MM separates the appearance of a spreadsheet from its calculations. All MM programs are written in two parts: the variables and equations, and a separate section stipulating how these are to be arranged and styled. It is this section that deals with matters such as cell formats, background colours, the allocation of variables to cells, and
whether a variable is to run down a column or along a row. This means that the results of calculations can be displayed in a variety of ways without changing the program. An interesting example of this is discussed in the section below on Multidimensional Tables.

We based the layout language on HTML, the language in which Web pages are written, largely because this is now very well-developed for expressing layout. And of course, many people have written Web pages and will have some knowledge of it. For those who do not, we shall explain the basics. For the purposes of this paper, the important thing about it is that an HTML document consists of plain text interspersed with tags that specify the document’s appearance. For example, \texttt{<H1>The MM user manual</H1>} creates a main heading; \texttt{<P>...</P>} enclose a paragraph; and the tags \texttt{<TABLE>...</TABLE>, \texttt{<TR>...</TR>}} and \texttt{<TD>...</TD>} are used to display tables, where \texttt{<TABLE>...</TABLE>} enclose the entire table, \texttt{<TR>...</TR>} enclose a row, and \texttt{<TD>...</TD>} enclose cells within the row. Our layout language is similar, augmented with special tags that insert attributes into the tables. Two examples are shown in the section on the decompiler.

2.3 Code Reuse; Named Constants

Had we stored the program above in a file called \texttt{company}, we could reuse it in another program:

```
include "company"
company2 =
  company
where
  incomings[ 1995 ] = 1000 and
  incomings[ all t > 1995 ] = incomings[ t-1 ] * 1.2
```

Here, we have extended \texttt{company} by adding an initial value for the 1995 value of \texttt{incomings}, and a growth law for the other values. This example illustrates code reuse. It also shows how one equation can be applied to a number of points within an attribute’s domain, as with the formula for profit earlier.

Instead, we could add some new attributes:

```
include "company"
constant average_wage = 500
company3 =
  company plus attributes <
    workforce [1995:2004] >
where
  outgoings[ all t ] = workforce[ t ] * average_wage
```

In this program, we have given \texttt{company} the new attribute \texttt{workforce} representing the number of employees. We calculate \texttt{outgoings} from it (again, in a terribly naïve way). This also shows that we can define constants to make code more readable.

In models that simulate some process that spans many regions of time or space, with the same thing happening in each region, constants often enable us to change the number of regions by altering just one number. The queuing simulation in Appendix A has two examples. To achieve the same effect on the raw spreadsheet would require a lot of error-prone copying and editing of cell references.

2.4 Parameterisation: Defining Templates

In the above, the company’s life span is hard-wired into the code. To change it, we would need to do a global find-and-replace of the years 1995 and 2004. We can avoid this by using parameters to our object definitions. Below, our definition of \texttt{company} has become \texttt{company_template} and acquired two parameters, \texttt{T1} and \texttt{T2}. These specify the time over which the attributes range. We then reconstruct \texttt{company} from this template. The great thing about this is that we can construct general templates and then use them in a whole variety of applications just by providing suitable parameters.

```
company_template( T1:integer, T2:integer ) = attributes <
  incomings [ T1:T2 ]
  outgoings [ T1:T2 ]
  profit    [ T1:T2 ]
>
where
```
2.5 Function Definitions

The equations in MM programs can call built-in Excel functions, as shown by the queuing simulation, which calls \( \min \) and \( \text{rand} \) amongst others. We considered that it might be useful for users to be able to define their own functions in terms of these built-ins. We implemented this in a prototype, but have omitted it from the current version because of difficulties over its interaction with templates.

2.6 Ranges as Arguments; Slices

Many of Excel’s built-in functions accept cell-range arguments. For example, \( \min(C4:D10) \) finds the smallest number anywhere within the rectangle whose top-left corner is \( C4 \) and whose bottom right is \( D10 \). We clearly need an equivalent in MM, which we provide by using the \( \text{range} \) operator. So \( \min( \text{range}\ \text{incomings} ) \) would find the smallest number anywhere in the \( \text{incomings} \) column. Using the word \( \text{range} \) before the attribute name just adds a bit of redundancy to confirm that the user did indeed intend a range and not just one value.

Sometimes, we need to pass only some of an attribute’s values. This is done by “slicing” – using subscripts to select a rectangular subregion. For example, \( \min( \text{range}\ \text{incomings}[1995:2000] ) \). The queuing simulation does this. It sets up a series of servers (representing shop assistants or similar), with one column for each. Each row represents one customer; the entry for the customer in row \( R \) and the server in column \( S \) specifies when \( S \) could start processing \( R \). For each customer, the simulation needs to search for the server with the earliest start time. It does this by slicing out the row, and then passing all the server-columns in it to \( \min \). This explains what is happening with \( \text{potential\_start\_time} \) in the example.

2.7 Ranging over Kinds of Employee: a More Realistic Example

From the examples above, you might gain the impression that MM is useful only for applications where all the attributes range over time. This is not so, as demonstrated by the “Lazy Days” spreadsheet shown in the Introduction. Here is the MM program that would produce it (without the data, which we assume to be typed directly into the spreadsheet):

```plaintext
base employee_kind =
{ "Managers", "Grade 1", "Grade 2", "Grade 3",
  "Grand Totals" }
attributes <
  staff_numbers [ employee_kind ]
  basic_wages    [ employee_kind ]
  overtime_wages [ employee_kind ]
  total_wages    [ employee_kind ]
  average_wage   [ employee_kind ]
>
where
  total_wages[ all e ] =
    basic_wages[ e ] + overtime_wages[ e ] \quad \text{and}
  average_wage[ all e ] =
    total_wages[ e ] / staff_numbers[ e ]
```

What we are doing here is to define a domain called \( \text{employee\_kind} \), and to say that our attributes all have one value for each element of this domain. (We call this domain a “base”, a word taken from the mathematics underlying MM.) This is analogous to what we did when making attributes range over the years 1995 to 2004, except that we have replaced the year numbers by symbolic values like “Grade 1”. To save space, we have not shown the layout definitions that would produce the above spreadsheet, but essentially, they would consist of \(<\text{table}>\), \(<\text{tr}>\) and \(<\text{td}>\) tags used so as to position the attributes and the headers \( \text{Lazy Days Staff Budget Costs 1995-1996, Average, and so on as they appear in the spreadsheet grid.} \)

2.8 Multidimensional Tables

There is no reason for attributes to range over only two dimensions. Had we holographic 3-d computer displays rather than flat monitors, we could imagine a cubical spreadsheet which tabulates the yearly costs due to a company’s departments: years running down columns, cost categories (overheads, lighting, transport) running...
from left to right across the front of the display, and department running from front to back, with a sheet for I.T. at the front, one for Personnel behind it, and a third for Sales behind that. Such attributes are easy to declare:

```
base cost_type = { “Overheads”, “Lighting”, “Transport” }
base department = { “I.T.”, “Personnel”, “Sales” }
attribute <
  costs : lifespan * cost_type * department
>
```

Even without a cubical display, we can display such a table in a variety of ways. We might have a number of tabbed panels, with a worksheet on each panel and one panel per year, department running down and cost type running across. We could swap cost type and department. Or we could put all the information on one worksheet, stacking successive years one under the other. And so on.

In such situations, to separate presentation from layout is particularly valuable. Excel enables this to be done to a limited extent by using pivot tables. However, these are an ad-hoc trick. We consider that using a separate layout section is the correct way to go, and we are working on notations to make it easy to express how multidimensional attributes are to be displayed. (Technically, we base these on the fact that the mapping from multidimensional attribute base to two-dimensional worksheet surface is a linear transformation. This makes storage allocation uniform, because the storage allocator can represent all such mappings as matrices.)

There are some spreadsheets, for example Storeys [ProFund page], that are better than Excel for displaying multidimensional tables. MM would be particularly valuable as a front-end to these.

### 2.9 Units and Dimensional Analysis

There is an Oxford story, recounted in [Morris 1978], that when John Keble was looking after the books of Oriel College, his accounts showed an inexplicable deficiency of between £1800 and £1900. It was eventually discovered that he had added the date to the college liabilities. We have spreadsheets now ... and they do nothing to prevent such errors. However, a paper on the programming language Algol68 [Cleaveland 1975] indicates a cure. Cleaveland suggests that it should be possible to declare variables to have units such as meters, seconds or pounds sterling. Knowing the properties of the arithmetic operators, a compiler could then check that expressions were dimensionally correct. If we declared $A$ to be seconds and $B$ to be meters, adding $A$ to $B$ would be invalid (adding length to time makes no sense), but multiplying them would be permitted. Hopefully, this would enable a large number of errors to be detected. The mechanism would easily extend to financial programs, those used by Oriel College included.

We have experimented with this in MM, by augmenting the programming language in two ways. Firstly, the user was allowed to declare names for units, such as cm, lb, £K and $. Secondly, he or she could attach these to attributes, telling the compiler for example that costs has units £K. Appendix 2 shows an example of the compiler checking unit errors.

Although the feature is useful, we have left it out of the current version of MM. One reason is that there are technical problems about the way it interacts with templates. Another is that it complicates function definition, because a function that operates on quantities with units must specify what units each argument can have, and the result’s units in terms of these. For example, were we to define $\text{square}(N)$ as $N \times N$, we would need to specify that if $N$ has units cm, then the result has units cm$^2$, and so on. Yet another problem is that consistent use of units is perhaps too verbose. For example, if we declare a constant length: $\text{cm} = 1 \text{ cm}$ we have told MM that it has units cm. Strictly speaking, if we do not give the quantity assigned to it – 1 – a unit too, we have committed an error, since a dimensionless number is not the same as one with a dimension. There is a tension here between conciseness and correctness which we have not yet resolved.

### 3 THE MM DECOMPILER

#### 3.1 Program Transformations

Going from spreadsheets to MM is harder than going in the other direction, because there are many different MM programs that could generate any given spreadsheet. For example, consider a trivial spreadsheet with only two occupied cells:

```
| Value 1 |
```

This can always be represented as an MM program with two single-valued attributes. But it could also have originated from a program with one attribute ranging over two points, or from a program with no attributes and two headings, or from a program with one attribute and one heading. We compare two of these possibilities below:

```
<table>
<thead>
<tr>
<th>attributes &lt; a &gt;</th>
<th>attributes &lt; s a &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>where a=1</td>
<td>where s=“Value” and</td>
</tr>
<tr>
<td>layout</td>
<td>a=1</td>
</tr>
<tr>
<td>&lt;table&gt;</td>
<td>layout</td>
</tr>
<tr>
<td>&lt;tr&gt;</td>
<td>&lt;table&gt;</td>
</tr>
<tr>
<td>&lt;td&gt;Value&lt;/td&gt;</td>
<td>&lt;tr&gt;</td>
</tr>
<tr>
<td>&lt;td&gt;&lt;attr name=&quot;a&quot;/&gt;&lt;/td&gt;</td>
<td>&lt;td&gt;&lt;attr name=&quot;s&quot;/&gt;&lt;/td&gt;</td>
</tr>
<tr>
<td>&lt;/tr&gt;</td>
<td>&lt;/td&gt;&lt;attr name=&quot;a&quot;/&gt;&lt;/td&gt;</td>
</tr>
<tr>
<td>&lt;/table&gt;</td>
<td>&lt;/tr&gt;</td>
</tr>
</tbody>
</table>
```

Although the decompiler can make some attempt to decide what the user intended – a column of numbers with text above is probably one multi-valued attribute plus a static heading stating its name – it will in general need to ask the user for more information.

The crucial insight here was that any spreadsheet can be trivially rewritten as an MM program, provided that MM allows any attribute to be arranged anywhere in a spreadsheet, at any position relative to another attribute. (If it didn’t, there would be some arrangements not expressible in MM). Once we have this program, we can transform it in various ways, using hints provided by the user or gleaned by examining the spreadsheet.

As an example, consider the Staff Numbers column from the “Lazy Days” spreadsheet shown in the Introduction. The trivial equivalent of this in MM (naming attributes after their cells) is this:

```
attributes < b2 b3 b5 b6 b7 b8 b9 >
where
  b2=“Staff” and b3=“Numbers” and
  b5=1 and b6=3 and b7=9 and b8=12 and b9=25
```

With a hint from the user, or by noticing that the two name cells contain text and are unrelated to any other cells, we can assume them to be headings, and remove them from the list of attributes, moving them into the layout section:

```
attributes < b5 b6 b7 b8 b9 >
where
  b5=1 and b6=3 and b7=9 and b8=12 and b9=25
```

The remaining attributes are all numbers. With another hint from the user, or by assuming that a column with text above it corresponds to one attribute, we could replace b5 to b9 by one multi-valued attribute:

```
attributes < b5b9[1:5] >
where
  b5b9[5]=25
```

We call this “rebasing”.

With yet another user hint, or by using the heuristic that text above a columnar attribute is most likely its name, we could replace b5b9 by a meaningful name:

```
attributes < StaffNumbers[1:5] >
where
  StaffNumbers[1]=1 and StaffNumbers[2]=3 and
  StaffNumbers[3]=9 and StaffNumbers[4]=12 and
  StaffNumbers[5]=25
```
We can do the same with the other columns. Of course, when an attribute is renamed or rebased, we have to transform any equations that refer to it. This may entail replacing names in the equation, but may also require changing or adding subscripts. For example, any reference to \( b^9 \) would have to be replaced by \( b_{5b9}[5] \) and then by \( \text{StaffNumbers}[5] \).

### 3.2 Pattern-wizards and rolled equations

In the section on compiler implementation, we talk about “unrolling” equations. This is what the compiler does when it takes an equation like

\[
\text{total_wages[ all e ]} = \text{basic_wages[ e ]} + \text{overtime_wages[ e ]}
\]

that applies to more than one point in an attribute and generates one instance of the equation for each of these points, i.e. one for each cell. To produce concise readable programs, the decompiler has to do the reverse: “rolling”.

It would be ideal if the decompiler could do this automatically. However, recognising when to is not easy, and for the moment, the user will need to specify rolling and the range over which to roll as another program transformation. Chris Browne, an expert on spreadsheets and their history, has suggested [Chris Browne’s Linux spreadsheet page] that we could start collecting common patterns of repetition and incorporating them into “wizards” that search for instances and offer to apply the most appropriate transformation. He has also suggested that the decompiler be used in the same way as people use algebra manipulation programs: we can provide a large repertoire of transformations, but the user may need to apply a lot of trial and error before finding a “best” MM program for a given spreadsheet.

Isakowitz, Schocken and Lucas [Isakowitz et al 1995] describe a decompiler which has a fair amount of success at rolling equations: we have the impression that the spreadsheets they tried had a very repetitive form generated by drag-and-drop copies. We have been told that these occur very frequently, due to the need to replicate calculations across time periods, so when rolling equations, it probably is worth looking for such patterns.

### 3.3 Error-checking

Once the decompiler has generated its equations, we can inspect them for errors, such as cells that should contain formulae but which have actually been hard-coded (section 5 of [Rajalingham et al 2000] under “Hard coding”). This will, we believe, be the decompiler’s most important use as an analysis tool.

Because of the tasks the decompiler has to perform, however, it can also display other information. It has to calculate dependencies between cells, and to discover which cells are used but not initialised, or initialised but not used. Once we have this information, displaying it – either inside Excel by changing cell colours, or in a separate analysis window - is not difficult.

The equations and dependency diagrams are also useful to people who will run but never modify a spreadsheet. In this connection, we are using some routines from the decompiler to extract programs from Excel and implement them for the Web, in a project for the Institute of Learning and Research Technology (ILRT) at the University of Bristol. This involves putting a series of economic models onto the Web as an educational resource – part of the Biz/Ed Virtual Learning Arcade - for students of business and economics [Biz/Ed page].

### 4 MM AS A STANDARD FOR SPREADSHEET INTERCHANGE

We are investigating MM as a medium for the interchange of spreadsheets. At present, XLS files are the most common means of transferring but we believe that using MM programs offers significant benefits. To transfer a spreadsheet developed in MM, one would just send the MM program. To transfer one that was not, one would decompile. As we have already mentioned in the Decompiler section, if one is prepared to help the decompiler by suggesting appropriate program transformations, the resulting program should be easier to understand than its parent spreadsheet.

These are the advantages we believe MM files have for interchange:

- Unlike XLS files, MM programs are text files, which are easy to process with general-purpose tools such as editors. This facilitates debugging when writing programs that process them.
- XLS is a complicated format, and descriptions of it are fairly hard to find, whereas the MM language has a simple syntax which we would be happy to publicise. When writing the MM compiler, we took care to write the parser as a separate module, with a well-defined API and clear documentation describing the data structures produced. We could also make this generally available.

---


Copyright © 2001 European Spreadsheet Risks Interest Group [www.eusprig.org](http://www.eusprig.org)
• One may sometimes want to process the spreadsheet formulae in other ways than evaluating them: for example, to produce an annotated cross-reference listing for reference documentation. This is easier to do with MM programs than with XLS files, partly because of the complexity of the latter, and partly because MM programs make more of the structure apparent, for example the replication of calculations over different time periods.

• MM completely separates the program from its appearance. The latter can be changed simply by modifying the layout section.

5 IMPLEMENTATION

5.1 The Compiler

The compiler uses standard methods from compiler technology, but differs from other compilers in that it allocates storage not for a one-dimensional machine memory, but for the two-dimensional, and visible, medium of a spreadsheet. It has 7 passes:

1. Input: reads and parses the input and builds the syntax tree. We use the freeware JavaCC [JavaCC page] parser-generator, which translates a grammar into a Java parser.

2. Semantic checking: analyses the syntax tree for errors such as undeclared identifiers and objects defined in terms of themselves. One theme that occurs here and in the decompiler and Web-based spreadsheet evaluator is the need to discover dependencies between attributes. There are standard directed-acyclic-graph algorithms for this.

3. Identifier substitution: replaces all identifiers by their definitions.

4. Colimit: performs an algebraic operation analogous to multiplying together all the object definitions in the tree, resulting in one big object definition for the entire program.

5. Storage allocation: allocates locations for all attributes, taking them from the layout section if there is one. This generates a map associating each attribute with a cell address.

6. Code generation: “unrolls” equations, for example replacing profit[ all t ] = incomings[ t ] – outgoings[ t ] by one equation for each value of t. Then it replaces attributes by their cell addresses. This gives us a “cellmap”: a map associating each used cell with the formula it contains.

7. Output: writes out the formulae together with instructions for placing them into their cells.

The compiler is written in Java, for portability. Why worry about this, when Excel only runs under Windows? Firstly, for our Web work, we want to generate code for our own Web-based spreadsheet engine, which may not be running on a Windows server. That will not concern many users, but another point is that Excel is not the only spreadsheet, and we do not want to restrict the range of machines on which MM can be used. Unix, for example, has the public-domain Oleo, Gnumeric and Dismal spreadsheets amongst others (a list can be found at [Spreadsheet FAQ page]), and MM would be as useful with these as with Excel. We have also mentioned its potential for use with ProFunda.

Because of portability, we are committed to Java, but the language has undoubted disadvantages for writing compilers. It is verbose; it is object-oriented to an extent which just does not suit compiler-writing; and there are certain idioms very useful here which Java has trouble with, such as higher order functions. For these reasons, we have written large parts of MM in Kawa [Kawa page], a Java implementation of the functional programming language Scheme. Kawa programs can be linked with Java, and so do not compromise portability.

Mathematical Background

MM was developed from a branch of mathematics known as category theory. Here is not the place to discuss details. Briefly stated however, category theory, like logic, is a tool for studying mathematical and computational concepts, one concerned much more with their form than with their content. It comes equipped with very general equivalents of common notions such as sum and product which can be applied to a vast and varied range of different situations. The computer scientist Joseph Goguen [Goguen 1975, Goguen 1992] has put it to use, together with another branch of mathematics known as sheaf semantics, in answering the questions “what is an object?”, and “what does it mean to say a system is composed out of objects?”. It turns out that his formulation can be implemented as a new style of programming, which we call System Limit Programming –
“limit” being the mathematical operation used to assemble components into a system. This gave us the attribute-and-base notation for MM, and – together with the ideas described in [Goguen and Tracz 2000] – the module structure.

5.2 The Decompiler

The decompiler has four passes:

1. Input: reads formulae from a spreadsheet file, again parsing it with JavaCC. Converts these into the same internal representation as that generated by pass 6 of the compiler, and then builds a cell-map.

2. Dependency analysis: builds a graph of cell dependencies, enabling input-only and output-only cells to be identified. An optional phase here checks for text cells with no dependants or dependees and assumes them to be headers and other static text, removing them from the cellmap.

3. Program transformation: reads and obeys transformation commands, rewriting attribute lists and equations as directed.

4. Output: pretty-prints the result as an MM program.

It is interesting to compare the decompiler with the factoring algorithm described by Isakowitz, Schocken and Lucas [Isakowitz et al 1995]. We developed our decompiler independently, but there are similarities. Like us, the authors recommend separating layout, which they call “editorial” information, from program, and their algorithm does so. It works in two stages, first allowing the user to split the spreadsheet into regions corresponding to separate attribute groups (we would call these objects: Isakowitz et al refer to them as relations). We do not do this yet, regarding the spreadsheet as “flat”, with all attributes in the same object, but it probably is useful. We are considering identifying blocks automatically, by using a depth-first region-numbering algorithm to find all disconnected regions of the spreadsheet.

Their second stage removes the static text, and then converts each relation into an MM-like language which the authors call FRL. Unlike us, the authors see this just as a convenient internal representation, and not as something to be understood by the user. FRL does not have explicit bases: in our terms, all attributes are based from 1 upwards.

6 PROGRESS AND FURTHER WORK

We have built several prototypes of the MM compiler, testing language features such as dimensional analysis, and are now completing a version which uses the same internal representations as, and will be integrated with, the decompiler. For the latter, some work still has to be done on allowing the user to describe program transformations conveniently, and on heuristics for identifying attribute names and automatically acquiring other useful information from the spreadsheet.

At present, both the compiler and decompiler are command-line-driven programs which communicate with Excel via SYLK files [SYLK page]. This is convenient for development, because SYLK is a textual representation, much easier to read and to edit than XLS. However, Microsoft have not bothered to maintain it – one source states that it has not been revised since Excel 2.0 – and it can not represent all the style information used in later versions of Excel. Nor can it handle multiple worksheets. To overcome this, we intend to connect MM directly to Excel so that it can be run as an Excel add-on, taking information from, and passing it to, Excel cells and cell ranges. This will also make it possible to provide the user with an integrated development environment for MM, which we regard as very important.

The largest model on which we have tested the compiler is the queuing simulation below; the decompiler has so far been run only on very small spreadsheets, of similar size to the examples in the introduction. We shall be trying it out on more spreadsheets of this kind when putting them onto the Web, as we describe under the decompiler section on Error Checking. This will be the only usability evaluation under the current funding, though we hope to set up a joint project with Clemson University to do more formal evaluation.

For the longer term, we note as we did in the introduction that MM enables two kinds of spreadsheet development. One is the conventional kind where one works directly with the spreadsheet. Here, MM is useful just as an analysis and error-detection tool, through its decompiler. The other is similar to that practised with conventional languages such as Fortran and Java. In this, the MM program is primary, and the spreadsheet is merely a means of running it and displaying the results: something to be generated whenever one needs to do a calculation, but otherwise to be ignored. Using MM in this way, one can take full advantage of the facilities for
code reuse, and can easily restructure spreadsheets as described under the section on Multidimensional Tables. We need to find out how best to combine the two kinds of development so as to provide the convenience of interactive inputting and testing with the readability, integrity and maintainability of MM programs.

7 ACKNOWLEDGEMENTS

I have done most of MM’s design and implementation in my own time. However, funding for putting the models onto the Web, and for the latest work on the compiler and decompiler was provided via the ILRT by the Joint Information Systems Committee 5/99 call for teaching and learning resources. I wish to thank Jack Ponton of the Department of Chemical Engineering at the University of Edinburgh for telling me about SYLK files, Margaret van Biene-Hershey for a valuable discussion on spreadsheet integrity as well as a delightful visit to Harmelen, Rob Kemmeren, and the anonymous referees for their comments on the draft version of this paper.

APPENDIX 1: QUEUING SIMULATION IN MM

The program below is an MM version of a queuing simulation. We designed it by reverse-engineering a spreadsheet written by Thomas Grossman, a professor of management science at Calgary University. He has written a variety of simulations in order to help students understand queuing behaviour [Grossman 1999]. Ours is derived from his basic four-server queue, downloadable as 4vanilla.xls from the Web page cited with this paper. Because the original spreadsheet, while interesting, is relatively complex and difficult to understand, we thought it was a good test for MM.

The program demonstrates several features of MM.

- Comments for program documentation.
- Attributes ranging over more than one dimension. The attribute potential_start_time has one dimension representing time: this runs from row to row, as with the other attributes. Its other dimension represents servers, and runs along the columns: there is one column for each server.
- Constant declarations make changing the spreadsheet structure easy. potential_start_time is declared as ranging over event*[1:N], where event is itself declared as [1:10]. In other words, potential_start_time ranges over the two-dimensional region [1:10,1:4]. The N referred to here is declared as a constant equal to 4. Simply changing this to another value would automatically change the number of columns allocated to the servers. Consider how little effort this is compared to modifying the spreadsheet itself!
- The use of Excel’s built-in functions if, match, min and rand.
- Cell ranges in function arguments. In the function call min( range potential_start_time[e] ), e is time: it selects one of the rows. potential_start_time has one column for each server, so the expression range potential_start_time[e] causes a cell range containing all these columns within the row to be passed to min.
- Commands for setting cell formats, and for placing headings in columns. We have now replaced these by the HTML-based layout language. Here is the program:

/* Queue.mm */

This is a process-driven simulation of queuing. It has two main components: a set of customers, and a set of servers. The number of customers and servers is fixed before starting. Customers enter a shop, go to a server, queue, get served, and leave. Each customer interacts once only and then leaves.

We could slice up time by allocating equally-spaced time points to successive rows. Instead, following Grossman, each row represents one customer's complete sequence of transactions: arriving, starting to queue, being served, and leaving. We handle this in MM by declaring the relevant attributes to run over a base (domain of observations) 1:10 similar, where the e'th point represents the e'th customer. These attributes are, for example, customer_number, arrival_time, service_start_time and service_time, all quantified over 'event'.

Again following Grossman, we allocate one column to represent each server. The only attribute of a server that we use directly is the potential start time that the customers would be served by it. So the columns hold these times. In MM, we do this by declaring the attribute potential_start_time's base to have a second dimension, running from 1 to N, the number of servers.
By default, the MM compiler puts the name of each attribute above its column. You can change this with the 'name' qualifier. The 'format' qualifier changes the format of the cells for the attribute values.

*/

def base event = [1:10];     // Each point represents one customer.
constant N = 4;              // The number of servers.
< customer_number : event
    name "Customer" br "#"  // The e'th element is the number of the e'th customer.
    interarrival_time : event
        name "Interarrival" br "duration"  // The e'th element is the time between the arrivals of
        // the e-l'th and e'th customers. For customer 1, it's  // the time between start and the customer's arrival.
        interarrival_time_mins : event
            name "Interarrival" br "duration" br "(mins)"
            format 0.00  // The same thing but in minutes, for display.
    arrival_time : event
        name "Arrival"  // The e'th element is the time at which the e'th
        format hh:mm  // customer arrives.
< potential_start_time : event * [1:N]
    name "Potential" br "start"
    format hh:mm  // The e,N'th element is the time at which the e'th
    // customer could start being served by server N, given  // that it may already be busy serving someone else.
< next_server : event
    name "Server" br "#"  // The e'th element is the server that will be used for
    // the e'th customer.
< service_start_time : event
    name "Service" br "start"
    format hh:mm  // The e'th element is the time at which the e'th
    // customer starts being served.
< service_end_time : event
    name "Service" br "end"
    format hh:mm  // The e'th element is the time at which the e'th
    // customer finishes being served.
< service_time : event
    name "Service" br "duration"
    format hh:mm  // The e'th element is the time taken to serve the e'th
    // customer.
< service_time_mins : event
    name "Service" br "duration" br "(mins)"
    format 0.00  // The same thing but in minutes, for display.
    start name "Start"  // When the simulation begins (time at which the servers
    format hh:mm  // open).
>

where

customer_number[all e] = e  // Every customer has a unique number. We display this,
// but don't use it in the calculation.

interarrival_time[all e] = 10 * rand() / ( 24 * 60 ) and
// The interarrival times are taken from a uniform
// distribution over the interval 0..10 minutes. Excel
// represents times as 1 day = 1, and its rand() function
// returns numbers from the interval 0.0..1.0, so we
// scale down by dividing by 24*60 to bring the range to
// 1 minute, then multiplying by 10.
interarrival_time_mins[all e] =
    interarrival_time[e] * 24*60 and
// Scale to 1 minute = 1, for nice display.
service_time[all e] = 20 * rand() / ( 24 * 60 ) and
// Service times are generated in the same way.
service_time_mins[all e] = service_time[e] * 24*60 and
// Scale to 1 minute = 1, for nice display.
arrival_time[all e>1] =
    arrival_time[e-1] + interarrival_time[e] and
arrival_time[1] = start + interarrival_time[1] and
// The arrival time of each customer is the arrival time
// of the previous one (or the start time, for customer
// 1) plus the generated interarrival time.
potential_start_time[all e>1,all N] =
    if( next_server[e-1] = N,
        service_end_time[e-1], arrival_time[e] ) and
potential_start_time[1,all N] = start and
// potential_start_time[e,N] is the earliest time at
// which server N can serve customer e, given that it may
// already be busy. For all servers,
// the potential start time for the first customer is the
// start of the simulation. The potential start time for
// other customers is the time at which the
// server finishes with the previous customer, if it has
// one; otherwise the customer's arrival time.
service_start_time[all e] =
    min( range potential_start_time[e] ) and
// The time when we can actually start is the minimum of
// the potential start times. We calculate this by using
// Excel's min function. The MM 'range' construct
// delivers the range of potential_start_time
// across all servers.
service_end_time[all e] =
    service_start_time[e] + service_time[e] and
// The time when the customer finishes being served is
// their start time plus the generated service time.
next_server[all e] =
    match( service_start_time[e],
        range potential_start_time[e], 0 ) and
// To find the first free server for customer e, we use
// Excel's match function. This scans the potential start
// times until it finds service_start_time, and returns
// the index.
start = 9 / 24
// The start time is 9:00 am.

This is the spreadsheet MM generates:

<table>
<thead>
<tr>
<th>Customer</th>
<th>Interarrival</th>
<th>Interarrival</th>
<th>Arrival</th>
<th>Potential</th>
<th>Server</th>
<th>Service</th>
<th>Service</th>
<th>Service</th>
<th>Service</th>
<th>Start</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>duration</td>
<td>duration</td>
<td>start</td>
<td>start</td>
<td>#</td>
<td>start</td>
<td>end</td>
<td>duration</td>
<td>duration</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.000985</td>
<td>1.22</td>
<td>09:01</td>
<td>09:00</td>
<td>09:00</td>
<td>09:00</td>
<td>09:00</td>
<td>09:00</td>
<td>09:00</td>
<td>09:03</td>
</tr>
<tr>
<td>2</td>
<td>0.005369</td>
<td>8.50</td>
<td>09:09</td>
<td>09:02</td>
<td>09:09</td>
<td>09:09</td>
<td>09:09</td>
<td>09:09</td>
<td>09:09</td>
<td>09:09</td>
</tr>
<tr>
<td>3</td>
<td>0.005981</td>
<td>7.46</td>
<td>09:17</td>
<td>09:10</td>
<td>09:17</td>
<td>09:17</td>
<td>09:17</td>
<td>09:17</td>
<td>09:17</td>
<td>09:17</td>
</tr>
<tr>
<td>4</td>
<td>0.015736</td>
<td>2.77</td>
<td>09:19</td>
<td>09:21</td>
<td>09:19</td>
<td>09:19</td>
<td>09:19</td>
<td>09:19</td>
<td>09:19</td>
<td>09:19</td>
</tr>
<tr>
<td>5</td>
<td>0.004946</td>
<td>7.12</td>
<td>09:26</td>
<td>09:26</td>
<td>09:26</td>
<td>09:26</td>
<td>09:26</td>
<td>09:26</td>
<td>09:26</td>
<td>09:26</td>
</tr>
<tr>
<td>6</td>
<td>0.001818</td>
<td>2.33</td>
<td>09:28</td>
<td>09:29</td>
<td>09:28</td>
<td>09:28</td>
<td>09:28</td>
<td>09:28</td>
<td>09:28</td>
<td>09:28</td>
</tr>
<tr>
<td>7</td>
<td>0.002665</td>
<td>1.11</td>
<td>09:33</td>
<td>09:33</td>
<td>09:33</td>
<td>09:33</td>
<td>09:33</td>
<td>09:33</td>
<td>09:33</td>
<td>09:33</td>
</tr>
<tr>
<td>8</td>
<td>0.001533</td>
<td>2.35</td>
<td>09:35</td>
<td>09:33</td>
<td>09:35</td>
<td>09:35</td>
<td>09:35</td>
<td>09:35</td>
<td>09:35</td>
<td>09:35</td>
</tr>
<tr>
<td>9</td>
<td>0.001845</td>
<td>2.66</td>
<td>09:38</td>
<td>09:38</td>
<td>09:38</td>
<td>09:38</td>
<td>09:38</td>
<td>09:38</td>
<td>09:38</td>
<td>09:38</td>
</tr>
<tr>
<td>10</td>
<td>0.005955</td>
<td>8.63</td>
<td>09:46</td>
<td>09:46</td>
<td>09:46</td>
<td>09:46</td>
<td>09:46</td>
<td>09:46</td>
<td>09:46</td>
<td>09:46</td>
</tr>
</tbody>
</table>

// This is the calculation.
APPENDIX 2: ERROR DETECTION

A2.1 USE OF IDENTIFIERS

The extract below demonstrates error-checking. We edited the queuing simulation so that it contained a duplicated identifier and two undeclared ones. The compiler has detected these and reported them to the listing file.

```c
1: /* Queue.mm */
2: ...
97: 98:   service_time : event
99:     name "Service" br "duration"
100: // The e'th element is the time taken to serve the e'th
101: // customer.
102: 103:   service_time : event
    ^
Error: Duplicate attribute service_time in object ...
... 132: 133:   arrival_time[all e>1] =
134:     arrival_time[e-1] + interarrival_time[e] and
    ^
Error: Undeclared identifier interarrival_time
... 139: 140:   potential_start_time[all e>1,all N] =
141:     if( next_server[e-1] = M,
    ^
Error: Undeclared identifier M
```

A2.2 UNITS

The extract below shows the compiler detecting unit mismatch errors in constant definitions. Two of these are errors across the + operator: the compiler assumes that adding quantities with different units is not allowed. The other is an error in the ^ (power) operator, where it is assumed that one can only raise a quantity to a dimensionless power. So raising a length to the power 2 to get an area is permitted, but raising it to the power of 2 times another length does not make sense.

```c
1: /* UnitsIncompatibleErrors.mm */
2: 3: unit cm
4: unit sec
5: unit £
6: 7: constant a = 1 cm + 2 sec
    ^
Error: The left-hand argument has units cm, but the
right-hand argument has units sec.
... 15: constant e = (1 cm) ^ (2 cm)
    ^
Error: Operator ^ expects something with no units here,
not units cm.
... 22: 23: constant i = 2(cm/sec) + 3cm*1£
    ^
Error: The left-hand argument has units cm * sec^-1, but
the right-hand argument has units cm * £.
```
REFERENCES


JavaCC page. This was [http://www.metamata.com/], but Metamata have just (24/4/2001) been taken over by Webgain. We are told the new page will shortly appear on [http://www.webgain.com/].


Spreadsheet FAQ page. This is the Frequently Asked Questions list for the comp.apps.spreadsheets newsgroup. It contains sections on free and commercial Unix spreadsheets, amongst other useful information, at [http://www.faqs.org/faqs/spreadsheets/faq/] 3:30pm 16/4/2001.

SYLK page. This has links to information about the SYLK file format and to code for parsing SYLK files, at [http://www.worldgate.ca/~rschulz/misc/] 3:20pm 16/4/2001.


Copyright © 2001 European Spreadsheet Risks Interest Group [www.eusprig.org]
An Evaluation of a Structured Spreadsheet Development Methodology

Kamalasen Rajalingham, David Chadwick, Brian Knight
Information Integrity Research Centre
School of Computing & Mathematical Sciences
University of Greenwich
30 Park Row, Greenwich
London SE10 9LS, United Kingdom
K.Rajalingham@gre.ac.uk

ABSTRACT
This paper presents the results of an empirical evaluation of the quality of a structured methodology for the development of spreadsheet models, proposed in numerous previous papers by Rajalingham K, Knight B and Chadwick D et al. This paper also describes an improved version of their methodology, supported by appropriate examples. The principal objective of a structured and disciplined methodology for the construction of spreadsheet models is to reduce the occurrence of user-generated errors in the models. The evaluation of the effectiveness of the methodology has been carried out based on a number of real-life experiments. The results of these experiments demonstrate the methodology's potential for improved integrity control and enhanced comprehensibility of spreadsheet models.

1 INTRODUCTION
This methodology represents a significant development or advance in the research into integrity control of spreadsheet models and the development of a methodology for spreadsheet model development. An outline of the methodology is presented by Knight B et al [Knight et al, 2000] and Chadwick D et al [Chadwick et al, 1999].

In this paper, an enhanced version of the methodology is presented. The methodology is essentially based on structured analysis of data, the outcome of which is represented as Jackson-like structures. It is shown that this analysis allows a straightforward modularisation, and that individual modules may be represented with indentation in the block-structured form of structured programs. The benefits of structured format are discussed, in terms of comprehensibility, ease of maintenance, and reduction in errors [Knight et al, 2000].

In order to assess and establish the quality of the methodology, four different experiments have been carried out. The results of these experiments have been analysed and they are presented in this paper.

2 A STRUCTURED METHODOLOGY FOR SPREADSHEET MODELLING

2.1 Introduction
Based on software engineering principles, mainly borrowed from Jackson [Jackson-1975], it has been found that spreadsheet models can be represented in a form identical to the data structure diagram developed by Jackson. Jackson [Jackson-1975] has shown how these Structure Diagrams can be mapped onto program code. In this paper, the proposed methodology demonstrates how these techniques can in fact be transferred to the production of spreadsheets, and this can give a more comprehensible format for spreadsheets, based on indented format. This is done by using a structured algorithm.

2.2 The Structured Algorithm Underpinning the Methodology
The algorithm consists of seven principal stages:

- Specification and Design of Outputs
- Conceptual Design of the Workings Section
- Logical Design of the Workings Section
• Construction of the Workings Section Structure
• Construction of the Input Section Structure
• Implementation of Functions and Relationships
• Completion of the Output Section

In this section, the methodology is applied in the construction of a spreadsheet model comprising a single module (as defined by the methodology). This is a simple model which does not require resolution of graph structures, which potentially result in the creation of separate modules, and recursive relationships.

It is based on a Trading and Profit and Loss Account [Ward-1996]. The original model is shown in Figure 1.

| T Howe Ltd |
|-----------------|-----------------|
| **Trading and Profit and Loss Account for the year ended 31 December 19X4** | |
| **Sales** | 135,486 |
| **Less Cost of goods sold** | |
| Opening stock | 40,360 |
| Add Purchases | 72,360 |
| Add Carriage inwards | 1,570 |
| **Total** | 114,290 |
| **Less Closing stock** | 52,360 61,930 |
| **Gross profit** | 73,556 |
| **Less Expenses** | |
| Salaries | 18,310 |
| Rates and occupancy | 6,515 |
| Carriage outwards | 1,390 |
| Office expenses | 3,212 |
| Sundry expenses | 1,896 |
| Depreciation: Buildings | 5,000 |
| Equipment | 9,000 |
| Directors' remuneration | 9,500 52,823 |
| **Net profit** | 20,733 |
| Add Unappropriated profits from last year | 15,236 |
| **Total** | 35,969 |
| **Less Appropriations** | |
| Proposed dividend | 10,000 |
| General reserve | 1,000 |
| Foreign exchange | 800 11,800 |
| Unappropriated profits carried to next year | 22,219 |

**Figure 1**: The Conventional Layout

**Stage 1: Specification and Design of Outputs**

This activity is carried out from the point of view of the model interpreter(s). The model interpreters are the end-users who interpret or use the output of the spreadsheet model for a particular purpose or to make business decisions. The methodology insists on the presentation of outputs on separate worksheets. The output(s) specified would consist of headings, labels and references to the workings and input sections. These sections would be constructed on separate worksheets later.
Based on the example of a Trading and Profit and Loss Model, the model developer would first examine the desired output(s). A typical output structure is shown in Figure 2.

![Figure 2: Output Structure](image)

If this layout was presented to a group of model developers, who are each asked to independently produce the spreadsheet model, they would come up with different layouts and structures based on experience and personal likes (and dislikes). Adopting the proposed methodology, a group of model developers assigned to independently build the spreadsheet model, should produce a set of structurally identical models.

**Stage 2: Conceptual Design of the Workings Section**

The purpose of constructing the workings or calculations section is to systematically and methodically perform the interim and final calculations based on and required by the model output(s). In developing the conceptual model, the first step is to identify the highest-level functions or model elements. These take the form of formulae with no dependents. They are therefore not referenced by any other elements within the spreadsheet model. Such functions can be referred to as root elements of the model.

The workings section of the spreadsheet model is represented in the form of Jackson structures [Jackson-1975]. The root elements would be placed at the top of the hierarchy, hanging from a box containing the title of the spreadsheet model. The immediate precedents of the root element would then assume their positions just below, adjacent to each other. In the same manner, each element would be positioned just below the model element of which it is a direct precedent. In many spreadsheet models, a root element would represent multiple instances, where each instance corresponds to a different time period, group, category, etc. This is shown as an iteration (appropriately labelled) with the root element appearing below it.

When a top-down approach is adopted without allowing duplication of elements, the initial model could take the form of a graph structure as opposed to the desired tree structure. The purpose of this is to distinctly show instances of multiple dependants of a particular element of the model. This potentially results in a structure as shown in Figure 3.
Figure 3: The Conceptual Design in Graph-form
Based on the desired outputs for the Trading and Profit and Loss Account example shown in Figure 2, the only root element that can be identified is **Unappropriated profits carried to next year**, as it is not referenced by any other model element. Figure 4 presents the conceptual design of the workings section.

Figure 4: Conceptual Design of the Workings Section

This model distinctly shows the precedents of the various functions. The leaves expenses and appropriations are represented as iterations in Figure 4. This is because each of them refers to a group of related inputs, defined as a range. The elements of a range are always operated on or manipulated as a set rather than individually. This structure should be transformed or resolved into a pure tree structure (if it is not already so) in the next stage.
Stage 3: Logical Design of the Workings Section

We have looked at a conceptual design which took the form of a tree. Not all spreadsheet models are of this simple form, but have structure charts in the form of a more general graph. Figure 3 shows an example of such a chart. The chart is different to that in Figure 4 in that there is a loop in the relationships connecting A, B, C and D, so that we do not any longer have a tree form. In this chart, data block D contributes to block B and to block C.

We can of course turn the graph into a tree. In order to accomplish this, two important rules have to be observed.

**Rule 1:**

*The initial graph structure is resolved into a tree-structure by duplicating elements with more than one dependant. The precedents of these elements are not included in the model at this stage. This is illustrated in Figure 5.*

Based on Figure 5, D and G are duplicated in order to resolve the graph into a tree structure. The precedents of D are not included in the model.

**Rule 2:**

*If a duplicate element has one or more precedents, it forms a separate module represented as a tree. The module consists of the duplicate element along with its precedents. This process is similar to First Normal Form (1NF) of normalisation in relational database design. This is shown in Figure 6.*
Referring to our Trading and Profit and Loss Account example, the model does not contain any graph sub-structures. Therefore, this stage is not applicable and can be omitted/skipped. In other words, the conceptual design of the workings section also represents its logical design.

**Stage 4: Construction of the Workings Section Structure**

To maintain the structure modelled in the logical design in the spreadsheet view, the indentation principle is used, both on the row labels and on the corresponding values themselves. In fact, we can also insist that the values are indented by assigning a spreadsheet column to each level of indentation. These can be referred to as *virtual columns*. The logical design of the model (represented as Jackson tree-like structures) is systematically mapped onto the physical spreadsheet based on rigorous rules prescribed by the methodology.

The following are the types of elements that can be found in the Jackson structures:

- Iteration (not associated with a single data value)
- Selection (representing mutually exclusive options)
- Function (takes the form of a formula)
- Leaf (reference to an input or input range)
- Constant (reference to an input and denoted using a C)
- Labels (each label is associated with a function, constant, leaf or iteration)
- Module (a branch or sub-structure referenced more than once within the model)
- Reference to a different iteration of the same module (indicated using *indices*)

All *iteration labels* are placed in the same column but are suitably indented to reflect their levels. All *function* and *input labels* are also placed in the same column, adjacent to the column containing iteration labels. They too are indented according to the levels at which they occur in the Jackson structures. The functions corresponding to the labels are built in a set of (virtual) columns adjacent to the column containing the function and input labels. The functions are located in different *virtual columns*, according to their position in the data structures. The term ‘virtual columns’ is used as the multiple physical columns are viewed as a single *logical* column. As such, each row can only contain exactly one function.
The positioning of functions in the various virtual columns is also consistent with the indentation of their corresponding labels. When these functions appear in different 'virtual columns', the comprehensibility of the model is improved significantly. The precedents of each function become easily identifiable.

**Figure 7:** Organisation of Functions in Virtual Columns

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>G</td>
</tr>
<tr>
<td>H₁</td>
<td>H₂</td>
<td>Hₙ</td>
</tr>
</tbody>
</table>

Based on Figure 7, A is a function of B and C, while C is a function of D and G. The precedents of function D are E and F, whereas the precedents of function G are H₁, H₂ and H₃. Referring to the Trading and Profit and Loss Account example, the logical design of the workings section of the spreadsheet model is now mapped onto the physical spreadsheet. This is shown in Figure 8.

**Figure 8:** Workings Section Structure

An asterisk (*) is placed next to a function label (in column A) to denote that the function operates on an input range (defined in the input section). The definition of a range in this context is described in the next stage.

**Stage 5: Construction of the Input Section**

There are reasons why cells for data input should be grouped together in an input section, separate from the structured modules described above. One reason is to do with the utmost importance of obtaining accurate data entry. A second reason is that input cells are often referred to by more than one calculated cell. Apart from these reasons, it is also a precaution against any accidental overwriting of formulae. This...
strategy is similar to the method introduced by DiAntonio [DiAntonio, 1986]. DiAntonio’s method advocates the isolation of facts by splitting the spreadsheet into two parts, one for the facts and one for the solution.

The design of this part of the user interface should be as free from constraints as possible; so as not to hinder the main objective: ease of use and absence of data errors. We are therefore, quite at liberty to put all data input cells into unstructured modules, since there are never any dependencies between them. Any dependency relationship in spreadsheet involves a calculated cell, and either other calculated cells or data input cells. However, they do not exist between data input cells and other data input cells.

Based on the leaves identified in the Jackson structures, the input section can be created. A problem that can be anticipated at this stage is the difficulty in adding or deleting data from the input section while having the changes reflected in the workings section. In view of this problem, the methodology requires that a group of related inputs be defined as a range and only the range is referred to in the workings section. A reference to a group of related inputs or an input set (range) is shown in the Jackson structure by a leaf represented as an iterated component.

The input section for the Trading and Profit and Loss Model can now be created in order to provide the workings section with the values required. This is done on a separate worksheet. The worksheet should be labelled input. Based on the logical design for the spreadsheet model, shown in Figure 4, the end-leaves can be implemented in an input section. This is shown in Figure 9.

The input data corresponding to the input groups expenses (C11 to C18) and appropriations (C20 to C22) are defined as ranges, and assigned the range names expenses and appropriations respectively.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>05 Sales</td>
<td>135,486.00</td>
<td></td>
</tr>
<tr>
<td>06 Opening stock</td>
<td>40,360.00</td>
<td></td>
</tr>
<tr>
<td>07 Closing stock</td>
<td>52,360.00</td>
<td></td>
</tr>
<tr>
<td>08 Purchases</td>
<td>72,360.00</td>
<td></td>
</tr>
<tr>
<td>09 Carriage inwards</td>
<td>1,570.00</td>
<td></td>
</tr>
<tr>
<td>10 Expenses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 Salaries</td>
<td></td>
<td>18,310.00</td>
</tr>
<tr>
<td>12 Rates and occupancy</td>
<td>4,515.00</td>
<td></td>
</tr>
<tr>
<td>13 Carriage outwards</td>
<td>1,240.00</td>
<td></td>
</tr>
<tr>
<td>14 Office expenses</td>
<td>3,212.00</td>
<td></td>
</tr>
<tr>
<td>15 Sundry expenses</td>
<td>1,886.00</td>
<td></td>
</tr>
<tr>
<td>16 Depreciation: Buildings</td>
<td>5,000.00</td>
<td></td>
</tr>
<tr>
<td>17 Depreciation: Equipment</td>
<td>9,000.00</td>
<td></td>
</tr>
<tr>
<td>18 Directors' remuneration</td>
<td>9,500.00</td>
<td></td>
</tr>
<tr>
<td>19 appropriations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 Proposed dividend</td>
<td>10,000.00</td>
<td></td>
</tr>
<tr>
<td>21 General reserves</td>
<td>1,000.00</td>
<td></td>
</tr>
<tr>
<td>22 Foreign exchange</td>
<td>800.00</td>
<td></td>
</tr>
<tr>
<td>23 Unappropriated profits from last year</td>
<td>15,236.00</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: Input Section

Stage 6: Implementation of Functions and Relationships

The structured spreadsheet modules represent the calculation or workings section. The structured spreadsheet modules also facilitate auditing and comprehension of the composition/meaning of calculations (expressed as formulae). The various formulae can now be physically implemented or
programmed. This stage involves constructing the various formulae and functions required in the workings section. The workings section structure has already been produced and will be used as a basis for the creation of the appropriate functions.

References to inputs are first entered into the relevant cells in the workings section. This includes functions on input ranges, such as **total expenses** and **total appropriations**.

A bottom-up approach is taken in the implementation of formulae and functions in the workings section. Figures 10 (a) and 10 (b) show the final state of the workings section of the Trading and Profit and Loss model. In Figure 10 (a), the structure of the underlying functions are shown as entered by the model developer. Figure 10 (b), on the other hand, shows the surface values of the functions based on the current state of inputs.

![Figure 10 (a): Workings Section](image)

![Figure 10 (b): Workings Section](image)

Based on Figures 10 (a) and 10 (b) it can be noticed that both the semantics and the data are clarified in this layout. For example, we can see straight away on the semantic level that Unappropriated profits carried to next year is derived from three figures: Net Profit, Unappropriated profits from last year and Total appropriations.

On the data level we see that **24,219** is made up from **20,733**, **15,286** and **11,800**. Likewise, we see immediately (from the asterisk *) that Total expenses references an input range from the input section. Notice also that columns in the spreadsheet show figures on the same semantic level, enabling valid comparisons between figures to be made. For example, column C shows net profit, unappropriated profits from last year and total appropriations. These figures give a valid impression of the state of the Trading and Profit and Loss Account at this level of detail. If we were to include a figure from a different
level, e.g. *purchases* (from column F), it would confuse the picture, since it has already been included in net profit.

Referring to Figures 10 (a) and 10 (b), it is beyond any doubt that the use of *indentation* and *virtual columns* make it far more straight-forward to make sense of and comprehend the composition of functions. However, the fact that references to data and other formulae within a particular formula take the form of cell addresses rather than meaningful labels is not entirely desirable.

In order to enhance the comprehensibility of formulae, cell addresses should be replaced with meaningful labels so that formulae are expressed in natural language form. Based on the *Trading and Profit and Loss Account* example, meaningful names would first be assigned to every piece of input data. The exception to this rule applies to a data value which is part of a related set of data that is always treated and operated on as a set, in which case it will be defined as a range along with the other related inputs. If every piece of input data in the input section is given a unique name, the workings section would now appear as shown in Figure 10 (c).

**Figure 10 (c): Workings Section**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>Unappropriated profits carried to next year</td>
<td>=C1+C16+D19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>Net profit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>Gross Profit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>Sales</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>Cost of goods sold</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Opening stock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Add Purchases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Add Cargoes received</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Closing stock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Total expenses **</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Add Unappropriated Profits from last year</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Less Total appropriations *</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This technique should be applied to all elements of the workings section. Every function/formula should be assigned a name so that meaningful names instead of cell addresses can be used for references within formulae in the *workings* and *output* sections. This is shown in Figures 10 (d) and 11 (c).

**Figure 10 (d): Workings Section**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>Unappropriated profits carried to next year</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>Net profit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>Gross Profit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>Sales</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>Cost of goods sold</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Opening stock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Add Purchases</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Add Cargoes received</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Closing stock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Total expenses **</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Add Unappropriated Profits from last year</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Less Total appropriations *</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Stage 7: Completion of the Output Section**

This stage brings the spreadsheet model development process to a conclusion. References to corresponding functions in the workings section can at this stage be entered into the relevant cells of the output section. The final state of the output section is shown in Figures 11 (a), 11 (b) and 11 (c).
Figure 11 (a): Output Section

Figure 11 (b): Output Section

Figure 14 (c): Output Section
3. EVALUATION OF THE STRUCTURED METHODOLOGY

3.1 Introduction

In order to evaluate the effectiveness of the proposed methodology, a series of well-organised experiments were undertaken. An analysis of the results of these experiments would revealed the methodology’s potential for integrity control of spreadsheet models. A major problem encountered was to persuade certain groups of spreadsheet users, especially those in industry (as opposed to academia) to take part in the trials. Therefore, the selection of user groups involved consideration of various factors such as circumstances, experience and various constraints. Two different strategies are formulated to evaluate the quality of the proposed methodology for spreadsheet model development.

User Groups or Participants

Ideally, the methodology should be tested on spreadsheet users, of varying levels of spreadsheet literacy, in both business and academia. Past experiments on spreadsheet errors have involved different types of users, from experienced spreadsheet developers from industry to novice spreadsheet students. It has been impossible for the authors to obtain consent to conduct trials with users in business organisations due to various reasons e.g. the time commitment, the problem of confidentiality of client data, the difficulty of obtaining a cohort of users all working with the same model at the same time, as well as the difficulty of obtaining a sufficiently large cohort to produce statistically significant results.

Referring to past experiments undertaken, it is found that most of the participants of such tests were students at an institution associated with the author(s). In most cases where the subjects were industry or commercial users, the experiment was either conducted by the particular organisation or the information derived from the normal operations of the organisation, published by the company itself.

Three different groups of students at a University were selected as participants for the experiments. They were as follows:

- Undergraduates
- Post-graduate students
- Students on a short course designed primarily for professionals in industry.

Types of Errors

Ideally, the tests should demonstrate the capacity of the proposed structured methodology to address all types of spreadsheet errors. The taxonomy or classification of spreadsheet errors [Rajalingham et al, 2000] is used as a basis for organising tests for as many different types of errors as possible.

Spreadsheet Models

The spreadsheet models selected and used for experimental purposes should be common business and financial models. The models should address the different features of the proposed methodology. Moreover, the models should have the capacity to be used to test for as many different types of errors as possible.

The spreadsheet models selected for the experiments are as follows:

- A Trading and Profit and Loss Account for a particular year [Wood-96]
- A Trading and Profit and Loss Account for several years [Wood-96]
- A Post-tax Income Distribution Model [Slater-90]
- Another common business model.
3.2 The Evaluation Strategies

Error Prevention

The first strategy for testing the quality of the proposed methodology is based on error prevention. It involves comparing the occurrence of errors in spreadsheet models developed based on the proposed methodology to the occurrence of errors in models built using conventional unstructured methods. The aim of this strategy is to establish whether or not there is a material difference in error rates between spreadsheet models produced using the two different approaches. The hypothesis is that users commit significantly fewer errors by adopting the proposed structured methodology. The first experiment is based on this strategy while the subsequent three experiments are based on a different strategy (error detection).

Error Detection

The second strategy for evaluating the effectiveness of the proposed methodology is based on error detection. It involves comparing the probability of detecting errors in spreadsheet models developed based on the proposed methodology to the probability of detecting errors in models constructed based on conventional unstructured methods. Errors are deliberately seeded into the spreadsheet models. The aim of this strategy is to establish whether or not there is a significant difference in the probabilities of error detection between spreadsheet models produced using the two different approaches. The hypothesis is that users are able to identify significantly more errors seeded into a model developed using the proposed structured methodology. This is a reflection of its comprehensibility. This is particularly important for audit, review and update purposes. Apart from the first experiment, the subsequent four experiments are based on this strategy.

3.3 The Experiments and their Results

Experiment 1

This experiment was carried out in two different stages, each involving two groups of students at a University. The purpose of the experiment was to compare two different approaches to the development of a single-module spreadsheet model. The first approach was based on conventional unstructured methods for spreadsheet model development while the second approach was based on the proposed structured methodology. This experiment was based on the first testing strategy, described earlier. The spreadsheet model used was based on a Trading and Profit and Loss Account for a particular year [Wood-96].

Stage 1

The first stage of the experiment involved the development of a spreadsheet model without any guidance or support. Subjects were given the desired output of the model as shown in Figure 3. In order to create the spreadsheet model based on the required output, they were provided with all the formulae needed. They had to employ suitable methods based on personal experience or discretion, and carry out the exercise independently. A total of 42 post-graduate students and 26 short course students (most of whom were professionals in industry) took part in this experiment.

The first test was carried out on a group of 22 post-graduate students. The students were pursuing a taught masters programme. Most of them had graduated in other disciplines and had limited prior knowledge of information systems. Each participant had to build the same spreadsheet model on two different occasions. The purpose of having the participants re-build the same model was so that it can be used as a control in the experiment.

The second test was performed on a group of 12 short course students. Most of the students were employed on a full-time basis in industry. Each participant had to build the spreadsheet model without having had a lesson on the proposed methodology.

---

The text is an excerpt from "Controlling the Subversive Spreadsheet – Risks, Audit and Development Methods" proceedings of EuSpRIG 2001 Conference. For more information, visit www.eusprig.org.
**Stage 2**

The second stage of the experiment involved the development of the same spreadsheet model based on a Trading and Profit and Loss Account. However, before they carried out the exercise, the students were given a lesson on employing the proposed methodology for structuring and building a single-module spreadsheet model.

The first test was carried out on a group of 20 post-graduate students. The students were also pursuing a taught masters programme. Each participant had to first build the spreadsheet model using a method they were familiar with. This was not based on any structured methodologies. The purpose of this exercise was to make sure that the errors committed by this group of students were in fact consistent with those produced by the previous group. The group was subsequently given a lesson on using the proposed methodology to construct a single-module spreadsheet model. They were then asked to re-construct the spreadsheet model based on the proposed methodology.

The second test was conducted on a group of 14 short course students. This was a different group of students but were pursuing the same short course. Moreover, they had a similar background, in that they were also mainly professionals in industry. The participants were asked to create the spreadsheet model, having had a lesson on building spreadsheet models using the proposed methodology.

**Results**

<table>
<thead>
<tr>
<th>Stage</th>
<th>Test 1a</th>
<th>Test 1b</th>
<th>Test 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>Post-graduate Students</td>
<td>Short Course Students</td>
<td></td>
</tr>
<tr>
<td>Sample Size</td>
<td>22</td>
<td>22</td>
<td>12</td>
</tr>
<tr>
<td>Mean Number of Errors</td>
<td>3.9</td>
<td>3.6</td>
<td>3.3</td>
</tr>
</tbody>
</table>

The errors include both quantitative and qualitative errors [Rajalingham et al, 2000]

**Experiment 2**

This experiment was based on the second evaluation strategy (error detection) and carried out in two stages. A total of 104 undergraduates took part in this experiment. The students were in two different groups. Both groups had to detect a total of 12 errors that had been seeded into a spreadsheet model. They were given the same amount of time to complete the exercise. The model was based on a Trading and Profit and Loss Account for several years. However, there was a fundamental difference between the layout or structure of the model used by the first group and the model used by the second group.

**Stage 1**

The first group consisted of 55 students and were presented a spreadsheet model in a conventional layout. Their task was to identify the twelve errors that had been seeded into the model. They were not aware of how many errors there were in the model.

**Stage 2**

The second group, on the other hand, was made up of 49 students. This group was working on the same model but it was structured based on the proposed methodology. The same errors had been seeded into this model as well, and group members had to independently detect them. They were unaware of the number of errors. They were given a brief and general lesson on how to interpret a spreadsheet model based on the proposed methodology without any references to the particular model used.
Results

Experiment 3

This experiment was based on the second evaluation strategy (error detection) and carried out in two stages. A total of 41 post-graduate students and 23 short course students participated in this experiment. Two identical tests were performed in each stage. Each test involved a different subset of students. Therefore 4 groups of subjects had to detect a total of 10 errors that had been seeded into a spreadsheet model. The model used in the first stage had a different structure/layout to the model used in the second stage. All participants were given the same amount of time to complete the exercise. The model used in this experiment was based on a Post-tax Income Distribution Model [Slater-90]. The original model was modified slightly to decrease its size.

Stage 1

In the first stage of the experiment, the spreadsheet model was presented based on the original (conventional) layout. They had to identify a total of 10 errors that had been seeded into the model. The first test involved a group of 19 post-graduate students while the second test was conducted on a group of 11 short course students.

Stage 2

In the second stage of the experiment, the spreadsheet model was re-designed and re-structured according to the proposed methodology. The same 10 errors were then deliberately seeded into the model. The participants of the experiment at this stage were given a brief and general lesson on how to interpret a spreadsheet model based on the proposed methodology without any references to the particular model used.

The first test was performed on a group of 22 post-graduate students while the second test involved a group of 12 short course students.

Results

Experiment 4

This experiment was very similar to the previous experiment. The only difference was that a different spreadsheet model was used. However, this was also a common business model. The model was simplified and its size reduced to make it less time-consuming to work on. The experiment was carried
out in two stages and involved a total of 44 post-graduate students and 23 short course students. Two identical tests were performed in each stage. Each test involved a different subset of students. The task of the 4 groups of participants was to detect a total of 10 errors that had been seeded into the spreadsheet model. All participants were given the same amount of time to complete the exercise.

**Stage 1**

In the first stage of the experiment, the spreadsheet model was presented based on the original (conventional) layout. The first test involved a group of 24 post-graduate students while the second test was conducted on a group of 12 short course students.

**Stage 2**

In the second stage of the experiment, the spreadsheet model was re-designed and re-structured according to the proposed methodology. They same 10 errors were then seeded into the model. As done in the previous experiment, the students taking part in the experiment at this stage were given a brief and general lesson on how to interpret a spreadsheet model based on the proposed methodology without any references to the particular model used. The first test was performed on a group of 20 post-graduate students while the second test involved a group of 11 short course students.

**Results**

<table>
<thead>
<tr>
<th>Stage</th>
<th>Test 1</th>
<th>Test 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stage 1</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subject</td>
<td>Post-graduate Students</td>
<td>Short Course Students</td>
</tr>
<tr>
<td>Sample Size</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>Mean Errors Detected (%)</td>
<td>44.3</td>
<td>36.5</td>
</tr>
<tr>
<td><strong>Stage 2</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subject</td>
<td>Post-graduate Students</td>
<td>Short Course Students</td>
</tr>
<tr>
<td>Sample Size</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>Mean Errors Detected (%)</td>
<td>67.8</td>
<td>78.1</td>
</tr>
</tbody>
</table>

**4. CONCLUSION**

The suitability of a methodology based on Jackson-like structures for spreadsheet modelling has been investigated. It appears that there are several possible advantages to the adoption of a structured method based on a Jackson data-oriented approach. These advantages may be summarised as follows [Knight et al, 2000]:

- a clear modularisation principle
- a top-level overview of module structure
- a structured indented format to the layout of module
- the possibility of automatic structuring of existing spreadsheets

The proposed methodology imposes a strict discipline in the process of spreadsheet development using software engineering principles. This reduces the occurrence of errors as spreadsheet models are designed and constructed in a structured and organised manner. The methodology distinctly describes a technique for modelling the spreadsheet problem and subsequently mapping the design onto the physical spreadsheet according to prescribed rules and a structured algorithm. The spreadsheet model is organised in a form which facilitates understanding and interpretation of the model in an unambiguous way. It is also appropriately decomposed into modules. This reduces the occurrence of most types of errors and increases the probability of detecting errors which are already present in the spreadsheet models.

In order to assess and establish the quality of the methodology, five different experiments have been carried out. The results of these experiments have been analysed and they are presented in this paper. The results of the series of four experiments conducted provide substantial evidence of the methodology's
potential for controlling the integrity and improving the comprehensibility of spreadsheet models. A more detailed version of the complete set of experiments and analysis of their results will be published in due course.

5. REFERENCES


Copyright © 2001 European Spreadsheet Risks Interest Group www.eusprig.org
NEW GUIDELINES FOR SPREADSHEETS

John F. Raffensperger
Department of Management, Private Bag 4800
University of Canterbury, Christchurch, New Zealand
j.raffensperger@mang.canterbury.ac.nz
22 May 2001

ABSTRACT

Current prescriptions for spreadsheet style specify modular separation of data, calculation and output, based on the notion that writing a spreadsheet is like writing a computer program. Instead of a computer programming style, this article examines rules of style for text, graphics, and mathematics. Much “common wisdom” in spreadsheets contradicts rules for these well-developed arts. A case is made here for a new style for spreadsheets that emphasises readability. The new style is described in detail with an example, and contrasted with the programming style.

1. INTRODUCTION

Proper writing of spreadsheets is a special concern to management educators and operations researchers. Educators should teach good habits and must read students' work. Operations research consultants use spreadsheets to produce math models for clients; these models can be large and complicated. Math modelling is now widely taught with spreadsheets, so teachers of math modelling must also teach spreadsheet use (see [Winston, 1996], for example). So we operations researchers have a special interest in good spreadsheet style. While this article has an operations research point of view, it has general application.

Spreadsheets are often hard to read and spreadsheets often contain mistakes. [Panko and Sprague, 1998] found that around 2% of spreadsheet cells contain errors. The likelihood of a correct model less than 50% for even a spreadsheet with only 35 cells. Why are spreadsheets so difficult? Part of the problem is due to existing prescriptions for spreadsheet style.

Most literature about spreadsheet style specifies a spreadsheet style based on computer programming and show a preoccupation with standards. In effect, people who have written about spreadsheet style have in their heads the metaphor that "Writing a spreadsheet is like writing a computer program."

The computer program style is a "standard" format with separate blocks for data, calculation, and output, as in [Bromley, 1985], [Kee, 1988], [Kee and Mason, 1988], [Stone and Black, 1989]. Some articles, such as [Bissell, 1986], [Edge and Wilson, 1990], and [Crain and Fleenor, 1989], mainly recommended that a standard format include various mixes of the following: heading, date, file name display, author, approval signoff line, table of contents, error summary box, instruction area, range names, global protection, absolute references where possible, validation formulas for input and output, no constants in formulas.
Most spreadsheet writers do not have a background in data processing [Cragg and King, 1993], so a "spreadsheet-as-program" metaphor is one that a reader is unlikely to expect, understand, or want. [Cragg and King, 1993] and [Davis, 1996] found that managers like spreadsheets for the freedom from internal information technology groups and may be hostile to perceived interference. So spreadsheet users do not use and do not want this style, and they do not want imposed standards.

Avoiding standards, [Edwards, Finlay, and Wilson, 2000] wrote a good high-level overview of spreadsheet use, guidelines for scoping spreadsheets, and "best practices" for verification. Similar to other writers about spreadsheet style, their article views the spreadsheet as a decision support system. By contrast, this article focuses more on cell-by-cell construction, and for the first time, draws on literature about style for text, mathematics, and graphics.

Abstractly, it is true that a spreadsheet has three parts: data, calculation, and output. However, this functional structure was originally designed for the computer's needs of batch processed punched cards, rather than the reader's needs. The input-calculation-output structure implies the calculation is a black box that the reader should ignore rather than read. Computer code is not written primarily for self-display, but a spreadsheet is meant to be seen and read. Instead of "data and calculation" or even "decision variable and constraint," the reader will be thinking in terms of the business problem, such as "person, shift assignment, preference." This suggests that a spreadsheet should be organised to follow the business logic. To do this, we need a new metaphor to replace "The spreadsheet is a computer program."

[Conway and Ragsdale, 1997] cite and comment on [Kee, 1988], “Reliable spreadsheet software begins with a standard format for developing spreadsheet applications'... However, contrary to Kee's sentiments (given above), for many optimisation problems we find that the forced use of a standard format results in spreadsheets models [sic] that are more difficult to construct, less reliable, and more difficult to understand." Later: “In most cases, we believe the spreadsheet design which communicates its purpose most clearly will also be the most reliable, auditable and modifiable design."

Conway and Ragsdale wrote the only paper with ideas other than "have modules for input, computation, and output, and avoid constants in formulas". Their important new proposals are (1) that related formulas should be in physical proximity, and (2) how we should write depends on how we read. All previous work found prescribes against such structure or ignores these aspects of layout. This article significantly extends the emphasis on readability by examining how styles for text, graphics, and mathematics can be applied to spreadsheets. The goal is to go beyond personal taste in spreadsheet style, to find well-tested concepts in more developed media, and use those rules for spreadsheets.

A spreadsheet is a mixture of text, graphics and mathematics, a form of expression and computation. Rules of style for those forms apply to spreadsheets. For example, as with writing, a spreadsheet is easier to read if its text has proper spelling and grammar. As with graphics, a spreadsheet is easier to read colour is used carefully. And as with mathematics, a spreadsheet should be easier to understand if the formulas are reasonably simplified.

With the gracious permission of Lindo Systems, Inc. (http://www.lindo.com), this paper uses as an example a spreadsheet called Assign.xls. Assign.xls is distributed with What's Best!
Lindo Systems, 1996, a commercial spreadsheet solver. The file can be downloaded with the student version of the company's Solver Suite from http://www.lindo.com. Assign.xls was written by operations researchers who do spreadsheets for a living and have a stake in making them understandable. Assign.xls is a simple employee scheduling problem. It illustrates key concepts of spreadsheet writing for educators and operations researchers.

In the following, a numeric cell is a formula or a constant referenced by formulas in other cells. A label, even if it is a numeric constant, is not a numeric cell. Also, we will avoid the word "user". The analogy to writing provides clearer terminology - writer and reader. You will see Excel formulas of the form WB(a, “operator”, b), where "operator" is "=" or "<=" or ">=". This is how What's Best! defines a constraint, where a is the left-hand side and b is the right-hand side.

The new guidelines for writing spreadsheets: Write a spreadsheet as text, mathematics and graphics are written. This paper is organised in order of these recommendations.

- Make your spreadsheets read from left to right and top to bottom.
- Be concise.
- Format for description, not decoration.
- Expose rather than hide information.

2. MAKE A SPREADSHEET READ LEFT TO RIGHT AND TOP TO BOTTOM.

2. 1. [Gopen and Swan, 1990] used linguistics and cognitive psychology to study scientific writing. They wrote, "Since we read from left to right, we prefer the context on the left, where it can more effectively familiarize the reader. We prefer the new, important information on the right, since its job is to intrigue the reader. Information is interpreted more easily and more uniformly if it is placed where most readers expect to find it." The contextual information on the left is considered old, and information on the right is considered new. Gopen and Swan are echoed by [Cohen, 1997] and [Microsoft, 1995, p. 384].

How would this apply to a spreadsheet? In a spreadsheet, old information is the input data, since the reader is expected to know it. The new information is the formula, derived information that the reader seeks. We expect to see the data first, and when we have digested that, we expect the output formula, to the right or below nearby. Intermediate formulas logically are data for later formulas, so the rule applies recursively to all numeric cells. Therefore, to write clearly, each formula should depend only on cells above and to the left.

Exceptions depend on reader expectations. An accounting balance sheet typically has years in columns. A year's profit in one column at the bottom may flow to the next year in the next column at the top. The reader of accounting may reasonably expect one year's bottom line to flow into next year.

[Archer, 1989] and [Davis, 1996] observed that cell relationships can be represented as a directed graph, which Archer called a cell relationship diagram. Excel's auditing toolbar has buttons to insert temporary arrows that point to the dependents or precedents of a cell. We will call these graphical segments - Archer's cell relationship diagram - the arcs of precedence.
A spreadsheet reads from left to right and top to bottom if, for every numeric cell, all the cell's arcs of precedence start above and to the left of the cell.

Clicking several times on the Trace Precedents button displays the complete precedence tree of the cell. This tree can be quite illuminating. If a precedence tree is tangled like a bowl of spaghetti, the spreadsheet legitimately can be called a spaghetti spreadsheet! If a spreadsheet's precedence tree includes blank cells, the spreadsheet is perverse, since it depends on information that is missing.

The Assign.xls Model sheet is shown in Figure 1. The first numeric cell is the Preference Total cell, but no other information about preferences appears on the Model sheet. After hunting, we find the Preferences sheet listed at the bottom. Assign.xls has the objective function at the top and front, the way one might write a Lindo model ([Lindo Systems, 1996]). Preference Total is the objective function, because E4 has range name WBMAX (in the box below the auditing toolbar). The precedence arc is no help; all precedents of this cell are on another sheet. We will come back to Assign.xls.

2.2. Have short arcs of precedence. Old information should be followed immediately by the related new information ([Gopen and Swan, 1990]). [Higham, 1993, p. 15], writing about mathematics, states, "Try to minimize the distance between a definition and its place of first use." [Conway and Ragsdale, 1997] wrote, "Things which are logically related ... should be arranged in close physical proximity and in the same columnar or row orientation."

If a cell is close to its dependents, the reader will more easily see the relationship between them. The spreadsheet will be naturally organised by blocks of meaning, blocks that reflect the business rather than the mechanical requirements of the spreadsheet. By contrast, arranging by input-calculation-output separates related cells by a large visual distance. A cell in an input block and a cell in a calculation block will be far from each other, so their logical relationship is harder to see.

![Figure 1. Assign.xls, version 1. The Preference Total precedence arc points meaninglessly off screen. The Preference Total cell is too far from its precedents.](image-url)
3. BE CONCISE.

Regarding text, [Strunk and White, 1979, p. 23] wrote about text, "Vigorous writing is concise." [Higham, 1993, p. 25] wrote about mathematics, "Do not use mathematical symbols unless they serve a purpose." [Tuft, 1983, pp. 51, 100] wrote about graphics, "Graphical excellence is that which gives to the viewer the greatest number of ideas in the shortest time with the least ink in the smallest space..." [Daellenbach, 1994] wrote about math modelling, "A good model is a model that is as parsimonious as possible in terms of the variables/aspects included."

None of the spreadsheet literature was concerned with this; that a spreadsheet should be concise is entirely novel, and controversial. But the literature shows clearly that about 2% of spreadsheet cells are wrong ([Panko and Sprague, 1998]). Reducing the number of cells therefore may increase the probability of a correct spreadsheet.

3. 1. Be concise with sheets. If it will fit on one sheet, consider putting it on one sheet. This is controversial - other writers encourage the use of multiple sheets, with little justification (e.g. [Mather, 1999]). Of course, one sheet may not be optimal for every spreadsheet, but multiple sheets are overused. How can multiple sheets cause problems?

Multiple sheets breed unnecessary "spurious" cells and labels, as information is copied from other sheets. A spurious cell has exactly one precedent; it simply points to another cell, such as the formula "B25=A 10” (not in Assign.xls). Mathematically, this is the same as $a = b$. A writer wants to remind the reader of data from another section; later formulas refer to the spurious reference rather than the original data. Such copying can duplicate the entire “input sheet” on every sheet! Now the reader feels the temptation to check back and forth, to verify the formula visually. Various bad habits motivate the use of spurious references, but mainly the problem is from long arcs of precedence or multiple sheets. In text, the item should be edited out as redundant. In graphics, the duplicate visual would be erased. In mathematics, the extra variable should be substituted out algebraically.

Multiple sheets dislocate related blocks; the reader must remember a context from one screen to the next. When moving from one screen to the next, the reader worries, I hope it's laid out the same as the previous one I just spent all this time learning."

Multiple sheets are hard to navigate. Many people do not know the key sequence for changing sheets (Ctrl-Pg Up, Ctrl-Pg Dn) and therefore must use the mouse, but most can use keys to move around a single sheet.

Multiple sheets make information harder to find. The search function in Excel does not scan different sheets by default, but only the selected sheet. To select several sheets at once, the reader must click each sheet tab while pressing Ctrl. Following this, the search will scan the selected sheets. Most people will not know such arcana.

Multiple sheets render auditing tools useless, as we saw above. An arc of precedence to off sheet cells displays an uninformative "off the sheet" icon, so the reader cannot visually see how cells are related. Putting the model on one sheet lets the reader see it in one glance, with the View Zoom command if necessary.
Multiple sheets make it hard to find blanks. Writers frequently do not bother to delete blank sheets in the file (Excel defaults to 16). This can be disconcerting to the reader. Are formulas on Sheet 6? There is no way to know, except to open Sheet 6 and look, a waste of time.

These problems with multiple sheets tend to be overlooked in the name of "modularity." What about large spreadsheets? Isn't it more convenient for large files if they are broken up in different sheets? Of course, many times a modular approach may be called for. However, large spreadsheets can often be reduced in size considerably by moving everything to one sheet, then eliminating the spurious cells.

3.2. Use a minimum of blank space, and only to divide blocks visually. Empty cells are not equivalent to empty space in graphics. Blank space in a picture does not hide anything and is not accidentally covered by nearby information. Its effect is purely visual. By contrast, a spreadsheet cell is a discrete object. It can be formatted as hidden. It can be blank, but perversely another cell can depend upon it. A blank cell may appear to contain data that is really in a different cell. Minimum blank space keeps the spreadsheet on fewer screens, and helps avoid an impression of hidden or misplaced information.

Though empty cells are not equivalent to blank space in a graphic, empty cells can be used in the same way, if the writer is careful. Separate blocks with blanks as one separates sentence's with a period or paragraphs with an indent. But arrange the data types so there is a reasonable minimum of punctuation. Small blocks can be adjacent without blank space between.

3.3. Keep information in one cell logically in one cell visually. Make blank cells look blank. Cells that appear to hold information should hold information. Try to avoid labels that overlap neighbouring cells. Leave on the default grid so cells are displayed.

In Figure 1, the Preference Total label could be in B4, C4, or D4. In fact, both C4 and D4 are blank. The label is in cell B4 with leading spaces. The leading spaces are unnecessary and should be deleted. The label should be put in cell D4, adjacent to the cell it labels.

For Figure 2, cut and paste was used to put Assign.xls on one sheet. Preference Total was moved to the bottom, so its precedents would be above it. Then Excel's default grid lines were turned on, and blank rows and columns were deleted. The spreadsheet now fits in three screens, even at low 640 by 480 resolution.
Figure 2 Assign.xls, version 2, on one sheet. Cells in rows 4, 6, and 8 refer to cells below, against the direction we read. B12, B16, and B20 also refer to cells below, and are unrelated to cells nearby. The spreadsheet should be arranged so each cell depends only on cells above.

The *What's Best!* constraints cells in row 4 depend on cells below, so the DAYS REQD, EVES REQD, and NITES REQD blocks should be moved down. Similarly, cell B3 is referenced by a cell below and to the right, with no direct bearing on nearby cells. It is like a sentence out of context, or a definition given too early. It should be moved downwards to be near the cells that use it. We will do this for Figure 3.

3.4. Be concise with blocks. Align data types consistently. If "names" are listed column-wise in one place, then list them column-wise everywhere. This gets the reader oriented to viewing the spreadsheet in a consistent way. Break this rule to confuse the reader and waste space with extra labels. Align the primary data type in rows. Excel has 65,536 rows versus 256 columns, so if the primary data type is downwards, the writer is less likely to run out of room. More importantly, we read left to right, so the reader expects to see the table structure across the top, in the column labels. The repetitive information should be below, so the reader views it by paging down. Where possible, have a single table.

[Conway and Ragsdale, 1997] wrote, "A design that results in formulas that can be copied is probably better than one that does not." Four signs for finding such a design are time, label repetition, concreteness, and formula transposition. All numeric cells varying with the same time periods should be in a single table, with the time periods in the far left column (except in accounting balance sheets). Label repetition (e.g. the days of the week labels in Figure 1) suggests that the writer could consolidate blocks. A concrete data type, such as "ex-president", is probably a better structure than an abstract data type, such as "data" or "constraints". Transposing with formulas is a loud sign that data types in the spreadsheet are misaligned. Transposing with formulas makes construction and modification difficult.
When a spreadsheet requires more than one table, stack unrelated blocks vertically (like a web page) or horizontally (like ticker tape), but not both (a bulletin board). Information one screen down and one screen right is hard to find, unless it is part of a table that starts on the far left. A bulletin board layout requires more keystrokes to navigate. Row and column operations will affect other blocks, so changes require moving cells, which is time consuming and error prone. So avoid the bulletin board structure.

Assign.xls was originally structured as a linear program, decision variables, constraints, objective coefficients, etc. (A linear program is a set of linear inequalities used to solve a business problem.) The objective function is at the top, as it might be in a linear program. A client is unlikely to want to use it that way. The client instead will probably think about ex presidents. What is Bush's assignment this week? How can a new ex-president be added? Structured by decision variables, preferences, and constraints, the client must copy three sections separately. We will rearrange Assign.xls so the primary data type is an ex-president, and move the “#Work” cell closer to where it is used. If we want to add another ex-president, there is only one block to copy. The amount of visual space used by Assign.xls (as in Figure 3) is now two and a half screens, better than the original four sheets.

4. BE CONCISE WITH CELLS.

All cells in a spreadsheet, sometimes even blanks, require the reader's interpretation. Unnecessary cells take up the reader's precious time, time that the writer wants the reader to spend on understanding the business logic. Unnecessary cells can contain mistakes, add visual and logical clutter, add to the bulk of the file, and confuse the reader.
4.1. Eliminate spurious references (described above). Fortunately, Assign.xls has none.

4.2. Erase dangling cells. A dangling cell is a numeric cell without dependents, a calculation not used anywhere else. A cell not on the precedence tree for the bottom line is dangling. Again, following [Archer's, 1989] cell relationship diagram, dangling cells will be leaves on the graph. Usually, a spreadsheet contains a "bottom line", such as an objective function or balance sheet total profit. Except for this, dangling cells are usually unnecessary.

Just as a spurious reference is analogous to redundant text, a dangling cell is analogous to irrelevant text, an unused variable, or unneeded decoration. Irrelevant verbiage in text is edited out. In graphics, repetitive visuals are erased. In mathematics, unneeded variables are substituted out. So, unless there is a compelling reason to keep them, erase dangling cells.

There are four main types of dangling cells: the unused input, the validation formula, the useless intermediate calculation, and the interpreted output. An unused input is a relic from an earlier version or an error in the model. (Whether a relic or an error, the spreadsheet needs more work.) We shall see more about relics later.

Authors cited earlier recommended formulas to check the validity of input and output. These extra formulas are dangling by definition. Temporary validation formulas are useful for debugging, but reduce readability, and eventually should be erased. Writers insert useless intermediate calculations, thinking the reader "will want to see a subtotal, just for their information". But the writer should instead drive home the main point rather than display trivial side matters. Let the readers put these in as they wish.

Interpreted output is like a comedian repeating the punch line, hoping for an extra laugh. For example, consider the formulas (not in Assign.xls):

D49: `SUM(D3:D48)`
D50: `if(D49>0, "Surplus of " +TEXT(D49,0), "0")`
D51: `if(D49<0, "Funding gap of " +TEXT(-D49,0), "0")`

Cells D50 and D51 merely interpret D49. Cell D49 should be labelled like "Surplus (gap)", and the two spurious formulas D50 and D51 should be erased. Avoid labels containing formulas. While they seem clever to the writer, the reader has a harder time distinguishing the numeric model from the documentation. Use formulas only for the model and use concise constant text for documentation.

Does Assign.xls have dangling cells? In the NIXON block of Figure 3, cell range C40:H42, and cells I40, J35 and J36 do not appear on the precedence tree of the objective function Total Preferences. But they are required for the constraints, and are referenced by the What'sBest! solver. This is a compelling reason to leave them in. But the "Total" cells J44, J46, and J48 are danglers of the useless-intermediate-calculation variety. They have no dependents and they are not constraints for the model, so they will be erased.

4.3. Simplify formulas. For example, the formula (not from Assign.xls) `C6*(A4) + A6*C6 + ((C6*A5))` could be simplified to `C6*(A4+A5+A6)`, or `C6*SUM(A4:A6)`, which changes automatically if a row were inserted. Table 1. summarises issues of formula readability.
Simplifying a formula requires knowledge of the order of precedence for arithmetic operators. The formula \( (C7/A8)*A7 \) could be simplified to \( C7/A8*A7 \) because multiplication and division are commutative. However, the reader may wonder if the writer meant \( CV(A8*A7) \), which is not the same. So put division last: \( C7*A8 \).

It is a good habit to use parentheses when we are not sure of the precedence of the arithmetic operators. However, in a long formula, extraneous characters can accumulate to make the formula hard to understand. When teaching students how to write a spreadsheet, educators ought to take a few minutes to remind students about the rules of precedence for arithmetic operators. Use the fewest characters necessary to write the formula correctly.

4.4. Nest and erase formulas where appropriate. Sometimes making a few cells slightly more complicated allows many cells to be erased. Nesting is especially appropriate when a formula in cell \( x \) has only one dependent, cell \( y \). The formula in \( x \) can be copied and substituted for the address of cell \( x \) in cell \( y \). This process should stop when the formula in the dependent cell \( y \) begins to lose readability.

Other authors (e.g. [Freeman, 1996] and [Mather, 1999]) say the opposite: complicated formulas should be separated into multiple cells. However, nesting of formulas follows from mathematical writing, which prescribes substituting out unnecessary variables. Virtually every other form of expression favours brevity. Why do we consider spreadsheet verbosity a virtue? The writer should weigh the number of cells against the readability of individual formulas, but the error thus far has been on too many cells.

Many cells can be eliminated with the sumproduct () formula. In Assign.xls, Figure, cells J37, J38, and J39 of the Nixon block multiply preferences by assignments, and then are summed in Preference Total. J37, J38, and J39 could be replaced with one sumproduct() and then nested directly into Preference Total. We can do the same substitution with the other blocks. The Preference Total becomes:

\[
C51: =\text{SUMPRODUCT}(C4:I6,C7:I9) + \text{SUMPRODUCT}(C14:I16,C17:I19) + \text{SUMPRODUCT}(C24:I26,C27:I29) + \text{SUMPRODUCT}(C34:I36,C37:I39)
\]

<table>
<thead>
<tr>
<th>More readable</th>
<th>Less readable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Similar terms repeat and reference similar ranges.</td>
<td>Irregular terms reference various ranges, references appear more than once.</td>
</tr>
<tr>
<td>References read left to right in row and column order.</td>
<td>References are ordered randomly in the formula.</td>
</tr>
<tr>
<td>Well-known formulas (e.g. the quadratic formula) are in a single cell.</td>
<td>Well-known formulas are made cryptic by separation into multiple cells.</td>
</tr>
<tr>
<td>Division generally appears at the end.</td>
<td>Division appears randomly.</td>
</tr>
<tr>
<td>Referenced cells contain constants.</td>
<td>Referenced cells contain formulas or blanks.</td>
</tr>
<tr>
<td>The formula has the fewest characters necessary.</td>
<td>The formula has unnecessary parentheses and spaces; the formula can be simplified.</td>
</tr>
</tbody>
</table>
This allows us to erase twelve cells, 4% of the numeric cells. All precedents of this cell are now constants, and may be viewed in a one click. Hence, it will be easier to debug and edit. While long, this formula is readable because it is regular.

4.5. Eliminate relics from earlier versions. A relic is anything no longer needed, but appears because the writer has not cleaned it up. Relics confuse the reader.

A simple way to look for spreadsheet relics is to press the End key, then the Home key. This moves the cursor to the lowest and furthest right cell for which computer memory has been allocated. If this last cell is not the bottom right cell of the intended spreadsheet, then there may be relics. Most of the time, extra rows and columns contain old formats.

A fast way to see if apparently blank rows or columns affect the spreadsheet is to delete them. If #REF! errors appear, undo the change and find the problem; formulas perversely refer to blank cells. References to blank are as dangerous as vagueness in legal writing.

Assign.xls contains relics of earlier versions. In the Model sheet, the data seem to be in columns A through J. But width was adjusted of columns as far right as AU. In fact, in the original version, pressing End and Home moves the cursor to cell IT22, far away from the apparent last cell in L 18. Fortunately, these relics are just old formats, as we see in Figure 4.

Figure 4. Assign.xls contains relics, unneeded formats from earlier versions.

5. FORMAT FOR DESCRIPTION, NOT DECORATION.

A format is descriptive if it displays information and if the reader already knows the format. A format is decorative if it displays no information. Writers format cells for various reasons: to get attention, to try to look professional, from perceived necessity, to encode data types, and to make a cell self-descriptive. These overlap somewhat, and are variously decorative or descriptive. The worst are cryptic; the best are crucial. In the following, we applied to spreadsheets Tufte's suggestions about graphics ([Tufte, 1983, p.183]).

Use formats to get attention or to try to look professional very sparingly. When a writer decorates with formatting, the writer should follow rules of style for those elements. Most spreadsheet writers do not have a degree in graphic arts, and should not attempt to decorate a spreadsheet. For the writer, default formats take no time to apply. For the reader, the default format is best in most cases.

Use only one font size. Large fonts take up screen space, take time to produce, and require changes in row heights which affect the entire spreadsheet width. Avoid using many colours;
readers bring their own complex interpretations and perceptions of colours, and may be colour blind. Avoid multiple styles in one cell, e.g. bold and italic, and use these rarely. Underscored spreadsheet text is ugly. To look professional, strive for a clean look rather than distracting big titles and heavy colours. To get attention on a few specific cells, use a logical layout and concise labels ([Tufte, 1983, p. 183]).

Use formats of perceived necessity very sparingly. The most common format of necessity is the column width. Except for the A column, which should contain labels, try to give all columns about the same width. Many different column widths can be disconcerting; the reader may try to find meaning in a cell width when there is none. Uniform columns are more aesthetically pleasing. Avoid varying row heights. A thick row will tend to force the reader to read primarily downwards. For the same reason, do not rotate fonts ([Tufte, 1983, p. 183]). Do not shrink fonts either - the reader's display may be small.

Format constant data differently from formulas. The most crucial structural distinction is the one between constants and formulas ([Conway and Ragsdale, 1997]). Constants can be interpreted faster than formulas, so the reader will see obvious constants with relief. The old approach to making this distinction was to put input data in a separate block, but this produces long arcs of precedence.

Without a compelling reason, do not try to encode data types beyond the distinction between constants and formulas. Additional encoding will not naturally convey information and can easily become cryptic ([Tufte, 1983, p. 183]). Instead, use a good block layout and clear labelling to make other distinctions between data types.

Following the principle of minimal formatting, either constants or formulas may be formatted - whichever results in the least formatting - with a light grey background colour and a thin grey border. A grey background is easy to notice, photocopies better than a coloured font, and is not distracting. In any case, document the difference on the spreadsheet with a formatted label, e.g. "Formulas are grey." In Figure 5, the formulas have been formatted.

![Figure 5 Assign.xls, final version. Abbreviations were eliminated. Formulas are grey.](image-url)
[Edwards and Finlay, 1997] also distinguish between constant data and formulas by protecting formulas to prevent accidentally overwriting a formula. Their argument is legitimate, but formula protection can be irritating, especially if a formula is wrong and password protected. Cell protection is a hidden format, so other formatting is required to indicate which cells may be changed. Readers should be informed that the spreadsheet is protected and told how to turn it off.

Which cells are constant in Assign.xls? The constant cells are the Assignments and Preferences blocks for each ex-president (Figure). With proper formatting, the reader would probably see that Assign.xls is rather simple. Most cells are preference inputs or decisions.

Use formats of self-description liberally. Format numbers according to their meaning, using widely understood conventions to make information self-descriptive. Right justify numeric cells and the labels above them. All cells of a given data type should display the same number of decimal places - the reader wants a number to self-descriptively indicate its magnitude by its width. If the number is a percent, format it as a percent, so it displays "%". If the number is money, format it to display a monetary symbol, such as "$". Show cents only if they are significant, otherwise decrease the number of displayed decimal places to zero. Show one decimal place ("$24.4") if the number is not in ones (e.g., if it is thousands), and label the cell, row, or column appropriately (e.g., with "(000)"). Show separators ("1,000,000").

6. EXPOSE RATHER THAN HIDE INFORMATION.

Put labels in the spreadsheet, and make sure most of them are on the left. On mathematical writing, [Higham, 1993, p. 25] recommends, "Avoid starting a sentence with a mathematical expression, particularly if the previous sentence ended with one, otherwise the reader may have difficulty parsing the sentence." The reader expects the label on the left, not a formula.

Spell out labels and use the spell checker. Abbreviations inhibit comprehension ([Tufte, 1983, p. 183]). Even supposedly "commonly understood" symbols (such as Q for the order quantity, A for order cost) should be written out in a spreadsheet (Order quantity Q, Order cost A).

Use proper case. Text is not clearer or easier to read if written in capitals ([Tufte, 1983, p. 183]). Just the opposite - it is harder to find the beginning and end of a sentence, and it is not what readers expect. Capitals are wider than lower case, so less text can fit in the width of a cell. The last version of Assign.xls appears in Figure 5, with proper case.

Of course, there is the old adage, "Do not put constants in a formula," since data in a formula hides information rather than exposes it, as in Table 2. But there is a more powerful rule: try to make every formula reference only constants.
Table 2 On the left, 0.07 should be in its own cell; references to B5, B6, and B7 should be in order. Cells should have self-descriptive formats.

A spreadsheet with hidden cells is perverse, because hidden cells are inaccessible dependents. Hiding cells or preventing changes in a spreadsheet is irritating and tends to reduce the reader's confidence in the spreadsheet. If a reader takes the time to audit a spreadsheet with hidden cells, the model cannot be proved correct, because the reader cannot see the formulas. The writer assumes the reader cannot improve the work, and preventing derivations from it diminishes its utility. The logic is literally hidden. Hiding cells is like writing in invisible ink.

Password protect can be over-used. Writers password-protect a spreadsheet to hide the entire spreadsheet from unauthorised readers, to hide portions of the spreadsheet from unauthorised readers, to prevent a reader from changing part or all of the spreadsheet (especially formulas), or to track changes to a spreadsheet. Excel allows the writer to password protect the spreadsheet in such a way that the writer can track changes made by others in a shared spreadsheet. Tracking changes is good. Hiding information tends to be bad.

Some writers feel the need to “idiot proof” a spreadsheet. This usually involves lots of formatting, heavy lines around the inputs, many colours, an input sheet, a summary sheet, password protection, etc. Instead, have the fewest cells necessary to produce the result, flow the logic from top to bottom and left to right, and put related cells close together. And put it all on one sheet. "Keep it simple" could be restated as "Keep it small."

Take the time to get it right. Text, graphics, and mathematics require editing. A large spreadsheet may need to be rewritten several times, as we find that we have misoriented data types, understood the problem better, or found a more succinct way to express the model. Make your spreadsheet resemble a system that your reader already knows. If the spreadsheet is automating or extending an existing system, the reader may be familiar with a related form ([Stewart and Flanagan, 1987]).

7. CONCLUSION

While we have used analogies to writing text, mathematics, and graphics, the analogy to writing text is perhaps the best. Creating a spreadsheet is more like writing text than it is like coding in C. We should tell students not to embed numeric constants in formulas, but we might explain it by saying "because you should not hide a key definition in a footnote or appendix." Hopefully, this paper has helped to debunk the idea that a "spreadsheet as computer program" structure will improve spreadsheet readability and reduce spreadsheet error. In its place, we proposed a new style for writing spreadsheets based on writing text, graphics, and math models. The style emphasises conciseness and readability.
8. ACKNOWLEDGEMENT

Thanks to Linus Schrage and the gentle folk at Lindo Systems for their help.

9. REFERENCES


Higham, Nicholas J. (1993), Handbook of Writing for the Mathematical Sciences, SIAM.


Session 3.

Audit and alternative approaches

Chairperson: Barry Pettifor
PricewaterhouseCoopers, UK

Papers in this session:

“Spreadsheet auditing”
David Nixon / Mike O’Hara, University of Salford, UK

“Peer audit and self-audit for reducing errors”
David Chadwick / Rodney Sue, University of Greenwich, UK

“Spreadsheet assurance by ‘control around’ is a viable alternative to the traditional approach”
Harmen Ettema / Paul Janssen / Jacques de Swart,
PricewaterhouseCoopers, Netherlands

Quote: “The control around approach gives an overall assurance, while traditional assurance is focused on accuracy of formulas.”
(Harmen Ettema / Paul Janssen / Jacques de Swart)
Spreadsheet Auditing Software

David Nixon, Mike O'Hara
Information Systems Institute, University of Salford, United Kingdom

ABSTRACT

It is now widely accepted that errors in spreadsheets are both common and potentially dangerous. Further research has taken place to investigate how frequently these errors occur, what impact they have, how the risk of spreadsheet errors can be reduced by following spreadsheet design guidelines and methodologies, and how effective auditing of a spreadsheet is in the detection of these errors. However, little research exists to establish the usefulness of software tools in the auditing of spreadsheets.

This paper documents and tests office software tools designed to assist in the audit of spreadsheets. The test was designed to identify the success of software tools in detecting different types of errors, to identify how the software tools assist the auditor and to determine the usefulness of the tools.

1. INTRODUCTION

Despite much research into the causes of spreadsheet errors [Panko, 2000], and the ways of avoiding spreadsheet errors, these errors are still the rule rather than the exception. Initial error rates in spreadsheet development and debugging are consistent with those in traditional programming Error rates in 'live' spreadsheets are so high because the steps, followed in traditional programming to reduce these errors, are overlooked in spreadsheets. Errors can be classified, based on types of error, source of error and/or time of error [Rajalingham et al, 2000]. Whilst cell-by-cell team code inspection proves the most successful method of spreadsheet auditing, this still only produces an 80% success rate. Software Engineering concepts can be applied successfully to spreadsheet model development and auditing [Grossman, 2002]. Some spreadsheet development methodologies / guidelines are based upon Software Engineering principles and concepts [Knight et al, 2000]. Good spreadsheet design is essential in reducing spreadsheet errors [Raffensperger, 2001] and increasing readability and maintainability. An efficient use of resources will usually require some form of risk analysis [Butler, 2000] to decide to what depth a spreadsheet model should be audited. The conceptual difference between what a user sees, and what the computer sees, in a spreadsheet is the main reason that auditing spreadsheets is so difficult. Auditing is helped by using a visual approach to cell descriptions [Chen & Chan, 2000]. The visual auditing approach lends itself to the use of software auditing tools.

From the review of the academic research performed relating to spreadsheet errors, it was concluded that auditing of spreadsheets is incredibly difficult, particularly without the use of a visual auditing approach. In order to address this issue, software has been produced to assist in the audit of spreadsheets. These software tools tend to provide a visual approach to assist the user in auditing the spreadsheet. Most also tend to be aimed at the spreadsheet developer, often providing additional functions to assist in the development of spreadsheets as well as providing auditing functions. These tools range from those that merely assist in cell inspection and audit, to those that attempt to identify unique formulae and potential problem cells. To date, little, if any, research has been published that assesses the usefulness and capabilities of these software tools. This paper documents an investigation into software auditing tools and attempts to answer the following questions:

- Are software auditing tools for spreadsheets useful?
- How do software auditing tools assist the user in auditing spreadsheets?
- What errors are software auditing tools good/poor at identifying?
To answer these questions, four software auditing tools, along with the auditing functions built in to Excel, were tested against a specially designed spreadsheet that contained seeded errors. The software tools tested were:

- Excel's Built-In Auditing Functions - These are included as standard functions in Microsoft Excel, and for the purpose of the test were deemed to be primarily the functions available on the Excel auditing toolbar. [http://support.microsoft.com/kb/289245](http://support.microsoft.com/kb/289245)

- The Spreadsheet Detective - A commercially available product that provides extensive auditing functions [http://www.spreadsheetdetective.com/](http://www.spreadsheetdetective.com/)

- The Excel Auditor - A commercially available product that provides functions for both auditing and development of spreadsheets [http://www.bygsoftware.com/auditor/auditor.htm](http://www.bygsoftware.com/auditor/auditor.htm)

- The Operis Analysis Kit - A commercially available product that provides auditing and development functions for spreadsheets [http://www.operis.com/oak.htm](http://www.operis.com/oak.htm)

- Spreadsheet Auditing for Customs and Excise (SpACE) - A tool in use at Customs and Excise to allow auditors to audit a business' VAT calculation spreadsheets. [http://www.auditwaresystems.co.uk/product.asp?productID=11](http://www.auditwaresystems.co.uk/product.asp?productID=11)

2. THE SOFTWARE AUDITING TOOLS TEST

2.1 The Sample Errors Spreadsheet

In order to investigate different spreadsheet auditing software, a Sample Errors Spreadsheet was developed. This spreadsheet is loosely based upon a spreadsheet in use at the CWS, that is used to produce flash turnover reports in the case of an OLAS system failure (OLAS is the accounting software used by CWS Retail). The spreadsheet is split into three worksheets. The first of these is used to enter flash turnover data, the second sheet is used to hold the data downloaded the previous week from OLAS, and includes the rest of the data needed to produce the report held on the third worksheet. Once the data has been manually entered into the first worksheet, and the file loaded into the second worksheet, the report can be produced and distributed. Figure 1 shows the errors added to the spreadsheet:

<table>
<thead>
<tr>
<th>Error Numbers</th>
<th>Error Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 4</td>
<td>Qualitative</td>
<td>Formulas overwritten with values</td>
</tr>
<tr>
<td>5</td>
<td>Qualitative</td>
<td>Fixed value used when a named cell reference should be used</td>
</tr>
<tr>
<td>6</td>
<td>Qualitative</td>
<td>Presentation (illegal)</td>
</tr>
<tr>
<td>7</td>
<td>Qualitative</td>
<td>Incorrect Protection</td>
</tr>
<tr>
<td>8 to 10</td>
<td>Quantitative: Mechanical</td>
<td>Incorrect summation</td>
</tr>
<tr>
<td>11 &amp; 12</td>
<td>Quantitative: Logical</td>
<td>A formula copied down a column looks at the wrong column</td>
</tr>
<tr>
<td>13</td>
<td>Quantitative: Logical</td>
<td>Incorrect Summation of a column including subtotals</td>
</tr>
<tr>
<td>14</td>
<td>Quantitative: Logical</td>
<td>Percentages added rather than calculated on a total cell</td>
</tr>
<tr>
<td>15</td>
<td>Quantitative: Logical</td>
<td>Percentages averaged rather than calculated on a total cell</td>
</tr>
<tr>
<td>16</td>
<td>Quantitative: Omission</td>
<td>A row missing from all the worksheets</td>
</tr>
<tr>
<td>17</td>
<td>Quantitative: Omission</td>
<td>A row missing from one of the worksheets, although it exists on the others</td>
</tr>
</tbody>
</table>

Figure One – Seeded Errors
2.2 Method Used to Test Software Tools

In order to maintain a level of consistency across the tests of spreadsheet tools, where possible, certain guidelines were followed. The tests each followed a three-stage procedure. The first stage involved the author becoming familiar with the software by examining it in an open session in order to gain a thorough understanding of the capabilities and limitations of the software, the way the software works and for whom the software was intended. This first stage allowed the author to investigate functions included in the software tool that were beyond the scope of the test errors in the Sample Errors Spreadsheet, to study any documentation provided with the software tool, and to become familiar with the software. The second stage involved the testing of the software tool against the Sample Errors Spreadsheet and the completion of the Test Results section of the Spreadsheet Auditing Software Test Form, including the allocation of a result to each error tested based upon the following criteria:

Figure Two – Pass/Fail Criteria

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSED</td>
<td>The software spotted the error with minimal intervention from the user, often highlighting the problem before the user examined the erroneous cell(s) on an individual basis. The software made the error much easier to spot by using visual display methods.</td>
</tr>
<tr>
<td>ALMOST-PASSED</td>
<td>The software hinted strongly at the possibility of the error and after a little investigation, the user, aided by the display tools in the software, was able to find the error.</td>
</tr>
<tr>
<td>ALMOST-FAILED</td>
<td>The software suggested the presence of an error but failed to identify the rogue cell(s). An investigation by the user found the offending error.</td>
</tr>
<tr>
<td>FAILED</td>
<td>The software failed to provide any suggestion that the error existed.</td>
</tr>
</tbody>
</table>

The second stage tests were, with the exception of the Excel built in auditing functions test, completed by allowing the software to guide the author to identification of each error. The Excel built in auditing functions were, due to the limitations in the way they work and the fact that they are available for use alongside the other tools, tested on a cell-by-cell inspection basis. The final stage of the test involved the documentation of any additional features, problems and an overall impression of the software.

The exception to the aforementioned three-staged approach to testing the software was SpACE. Unfortunately it was not possible for the author to acquire a copy of SpACE for testing purposes without attending a training course in its use. As there was no training course scheduled before the submission date of this paper, it was impossible to test SpACE in the same three staged approach used in the tests of the other software tools. As a result, it was agreed that the second stage, the test against the Sample Errors Spreadsheet, would be done by an experienced user of SpACE, under the observation of the author, with the author awarding results based upon the observation of the test and completing the Spreadsheet Auditing Software Test Form after the test had taken place. Screenshots and reports were taken from each of the tests to allow future analysis of the results.

Due to the nature of the errors and the result criteria, along with the fact that the tests took place over a period of time, the results of the test could be classed as largely subjective. To attempt to reduce any subjective aspect of these results a normalisation stage took place once all of the tests had been completed. This involved examining each error in turn across the five software tools tested and where necessary, adjusting the results slightly to establish a level of
consistency and to reduce the subjective elements of the results. Although a degree of subjectivity undoubtedly remains in the test results, because they are based upon questions of opinion such as "with this information in this format, how likely would an auditor be to identify this error?", as the expressed primary aims of the tests were to assess usefulness, methods and detection rates across error types, it was felt that a certain level of subjectivity in the test results could be tolerated, particularly as the results were to be drawn from five different software tools.

2.3 Excel's Built In Auditing functions

The first software to be tested against the Sample Errors Spreadsheet was Excel's own built in auditing functions. These built in functions were unique, in that, as they are included in Excel as standard, they were also used to supplement other software tools tested. As a general rule, the test utilised the built in functionality when testing other software tools only when the software had indicated potential errors within the cell or range of cells.

In the test, which was performed largely on a cell-by-cell inspection basis, the built in functions successfully highlighted the four 'formulas overwritten by values' errors, using both the view formula mode on the offending worksheets, and by showing that the cells in question did not contain any precedents. The functions failed to identify the use of a constant in a formula as a potential error, and had no options to identify patterns in labels on the report sheet so failed to identify the 'illogical presentation' error. As the built in functions required the user to remove worksheet protection before use, and makes no attempt to identify unprotected cells, the functions failed to find the 'incorrect protection' error. The built in functions had more success on the three mechanical errors, only struggling on the formula that involved a totalling of separate rows rather than a whole column. On the first three logical errors (domain errors) the functions clearly showed the constituents of the formulae, but this gave no suggestion that the constituents were incorrect. The built in functions fared slightly better on the final two logical errors (totalling and averaging of percentages) although it would still be possible for the user to overlook these errors, particularly if the user did not possess sufficient domain knowledge. The built in functions failed to indicate any possible errors in the final two errors in the test, both omission errors.

The major drawback of the built in auditing tools in Excel is the fact that in the majority of cases, the tools are used on an individual cell basis that means that the auditor is, in effect, doing a cell-by-cell inspection. Whilst this has proven successful in tests, and the auditing functions provide a more visual representation of the cell under investigation, a major reason for using a software auditing tool is to reduce the time spent on auditing spreadsheets by avoiding the time-consuming cell-by-cell method. The built in tools prove useful in the investigation of single cells and as a result can be used alongside other software auditing tools. The built in functions do not, however, provide the user with any guidance to identify potentially unsafe cells.

2.4 The Spreadsheet Detective

The Spreadsheet Detective is an Excel add-in and is produced on a shareware or licenceware basis by Southern Cross Software. The full registered version costs approximately £100. Unlike the test on the built in auditing functions in Excel, the test of The Spreadsheet Detective was not done on a cell-by-cell inspection basis.
There are two fundamental ways in which the software attempts to assist the user in the auditing of the spreadsheet. The first is the identification of formula schema, which should mean that the user has to check far fewer formula than under a cell-by-cell inspection method, as copied formula are identified as belonging to the same schema and therefore need not be checked. The second is the listing of potential problems such as references to non-numeric cells or unprotected schema in the footer report.

On the test itself, The Spreadsheet Detective performed particularly well, only really struggling on the errors that were particularly difficult to find. The annotate sheet function which provided an overlay to a worksheet with different types and colours of shading and text descriptions of formulae, meant that the first four errors concerning formula that were overwritten as values were obvious.

The footer report included a list of potential error cells including the 'constant in formula' error so this was also easy to find. The software failed to highlight the 'illogical presentation' error, although this could come to light when the adjacent cells are examined as they were deemed to be within a new formula schema as the order had changed. The 'incorrect protection' error was particularly easy to identify as unprotected cells were listed in the footer report despite the fact that worksheet protection had been disabled.

The Spreadsheet Detective achieved two 'almost-passed' and a 'passed' result for the three incorrect summation errors. It achieved the 'passed' result because of the automatic naming function which shows the formula in an English language type format, based upon the row and column headings, which meant that the error resulting from totalling the wrong rows stood out. The two logical domain errors, involving formulae looking at the wrong columns, were also much easier to find due to the automatic naming function. The error concerning the total formula that included both sub totals and detail lines was highlighted in the footer report under references to non numerics as it also included blank cells, so this was also easy to identify. A combination of the new schema shaded overlay, and the automatic naming function, meant that the totalling and averaging of percentages errors were relatively easy to identify. The lack of any un-referenced cells identification, or label pattern recognition, meant that the software did not highlight the final two omission errors.

The Spreadsheet Detective proved successful in the test, and has many functions that go well beyond those errors in the test spreadsheet. The two pronged approach of using an overlay to identify formula schema and types of cell, and having a summary report of potential errors, meant that the user was guided towards many of the errors by the software, the summary report proving particularly useful in identifying the more subtle errors such as unprotected ranges. The automatic naming of the constituents of formulae worked well in the errors spreadsheet, although this may prove less useful in a spreadsheet that is not as logically laid out with easy to identify row and column labels. However, this naming function remains one of the best features of the software and could even encourage developers to design spreadsheet models in a more logical way, to comply with these naming conventions.

2.5 The Excel Auditor

This software is produced by BYG Software. The Excel Auditor is an add-in for Excel and, despite its name, provides many functions outside the usual scope of auditing software.
The Excel Auditor provides two primary and two secondary auditing tools. The primary tools being the Audit Map, which provides a traditional audit map of a worksheet, and the Worksheet Cell Analyser, which documents the contents of a worksheet's cells. Both of these tools produce a report on a separate workbook. The secondary tools are the Cell Roots and Circular Reference Analyst functions.

On the test itself, The Excel Auditor performed poorly. Unlike the test of Excel's built in auditing functions, The Excel Auditor was not tested on a cell-by-cell inspection basis. This was because a primary aim of a supplementary auditing package should be to save the auditor the time that would be taken if a cell-by-cell inspection were necessary, and also to keep the test consistent with the other supplementary tools tested in this report. Unfortunately this meant that The Excel Auditor did not perform as well as the Excel functions due to the method of testing.

As a cell-by-cell inspection was not used with The Excel Auditor, the first four errors, concerning formulae overwritten with values, whilst clearly shown on the audit map, could easily be overlooked as they would require the use of the audit map alongside the original worksheet. In this case it could be argued that it is easier to move the cursor over each cell in the worksheet and watch the Excel input bar. As a result, The Excel Auditor only achieved an 'almost-failed' result for each of these errors. The Excel Auditor also achieved an 'almost-failed' result for the 'constant in formula' error as, whilst it clearly showed the constituents of the formula in the worksheet cell analyser, it failed to warn of any potential problems with this approach. The software failed to identify any problem with the 'illogical presentation' error and whilst the documentation function showed which worksheets had protection enabled, it could not do the same for individual cells.

On the three mechanical errors that related to incorrect summation, The Excel Auditor, whilst clearly documenting the formulae concerned, failed to suggest any problems with the formulae and therefore only achieved an 'almost failed' result for each of these errors. The same issues surfaced on the two logical/domain errors concerning 'incorrect formula' due to the wrong column being used' and the summation formula that included sub total rows as well as detail rows. Both of the incorrect percentage calculations suffered from the fact that they were buried away in the 64 page worksheet analysis report produced for the report sheet, although the average calculation was slightly easier to identify as an error. The final two omission errors both failed to be identified by The Excel Auditor, as there is no way of looking for label patterns or efficiently looking for cells with no dependents.

Perhaps the biggest problem with The Excel Auditor as an auditing tool is that is still requires cell-by-cell inspection to allow the auditor to confidently audit the spreadsheet. The software is also let down by the lack of visual aids to auditing, particularly by the fact that it functions on separate reports, meaning that the user is required to jump from report to spreadsheet whilst auditing the spreadsheet. The Excel Auditor is more likely to be of use as a documentation tool, rather than an auditing tool, as the documentation tools, although not covered in the test, are quite useful and easy to use.

2.6 The Operis Analysis Kit

This software is available from Operis Business Engineering Limited and is in the form of an Excel add-in. To perform spreadsheet auditing, Operis Analysis Kit adopts a two-pronged
approach. The first allows the user to search the worksheet for particular types of cells that are more likely to be problematic, such as those with no dependents or those with formula containing hardwired constants. The second approach concentrates on graphically mapping the cells on a worksheet overlay to identify formula schema, referred to in Operis Analysis Kit as distinct formula, and types of cell, and to document the distinct formula and relevant statistics.

In the test itself, the four 'formulas overwritten with values' errors were easily identified using the option to overlay colour-coded cells to the worksheets. The search for hardwired constants option correctly identified the 'fixed value in a formula' error. The software did not, however, highlight any potential problem with the 'illogical presentation' error, although this could be spotted when the adjacent formula cells were examined as they were deemed to have distinct formula, so an 'almost-failed' result was recorded for this test. Operis Analysis Kit failed the 'incorrect protection' error, as it makes no attempt to identify which cells are left un-protected. The three mechanical errors all achieved an 'almost-passed' result as each was identified as a distinct formula and the contents of these formula were clearly documented.

The two logical/domain errors achieved 'almost-failed' results, as although the formula was marked as distinct in each case, and the formulae were clearly documented, there was no suggestion that the formulae were incorrect. The 'incorrect summation of a column including sub totals' error was easily identified using the search option to find references to blank cells. The two incorrect percentages tests produced contrasting results. On the first of these tests, which involved the totalling of percentages, Operis Analysis Kit achieved a 'failed' result as the formula was not identified as distinct as it matched the formula in the cell to the left, whereas on the second test, that of the averaging of percentages,

Operis Analysis Kit achieved a 'passed' result as this was correctly identified as a distinct formula and upon examination of the cell documentation was obvious. The two omission errors also produced contrasting results; with the first error achieving a 'failed' result, as there were no options to identify label patterns, whereas the second error achieved a 'passed' result, as using the search for unreferenced cells on the file worksheet correctly highlighted the problem.

Operis Analysis Kit as a whole is very easy to use thanks to its operation via a simple additional menu in Excel. It is deceptively powerful, particularly in the search options, and managed to at least hint at all of the more 'findable' errors with the exception of its inability to identify unprotected cells.

2.7 Spreadsheet Auditing for Customs and Excise (SpACE)

SpACE was developed in-house by HM Customs and Excise for use by VAT inspectors in auditing client's spreadsheets, and is now available to the public. SpACE works by using a combination of search facilities, overlaid mapping options and the identification of unique formula, to attempt to highlight potential errors in a spreadsheet.

The test of SpACE was done slightly differently to the other software tool tests as the test was performed by Ray Butler, an employee of HM Customs and Excise and a regular user of the SpACE software, under the observation of the author. Unlike the author, Ray did not know the location of the seeded errors in the sample errors spreadsheet before the test. SpACE
proved exceptionally good on the test itself, recording at least an 'almost-passed' result on each error test.

The software easily passed the first four 'formula replaced with values' tests using either the colour coded overlay, or the identification of numeric cells with no dependants option.

SpAce achieved an 'almost-passed' result in the 'fixed value in formula' error as the software allows the user to search for fixed values, but the user has to enter the value to be found.

SpACE achieved an 'almost-passed' result on the 'illogical presentation' error as it allowed the user to search for particular text strings and apply colour coding which, along with the indication that the adjacent formula cells do not follow the same pattern as the other should highlight the problem.

SpACE successfully found the 'incorrect protection' error as, despite automatically unprotecting the worksheets, the software is still able to identify the individual cells that have been marked as unprotected. In line with most of the other tools tested, SpACE correctly identified the three mechanical errors (incorrect summation errors) as unique formula yet the lack of a visual report means that this error could be overlooked by the user. The two logical/domain errors both achieved an 'almost-passed' result, thanks largely to the SpACE option of attempting to link user specified root and bottom-line cells.

When the software failed to establish a link between these two cells, further investigation revealed the errors. The 'incorrect summation of column containing subtotals' error was easily found by SpACE as it was highlighted as a unique formula, which upon further investigation shows the error to be obvious. Both of the erroneous percentage calculations were highlighted by SpACE in a way that was obvious enough to earn a 'passed' result. SpACE was the only software tool that had any success with the first omission error, which involved a row missing from every sheet in the workbook. It achieved an 'almost-passed' result by a combination of a schema identification method that allows 'blocks' of formulae to be grouped together, and the option to highlight text strings with different coloured backgrounds. The final omission error, achieved a 'passed' result on SpACE as the option to identify cells with no dependencies highlighted the offending row on the file worksheet.

SpACE is a 'tried and tested' software tool and has obviously been in use and subject to improvement for some time. In addition to the functions covered in the test, SpACE includes more in-depth auditing tools such as the ability to check lists of data for duplicates and attempt to identify the numbers that make up a particular total. The only major function missing from SpACE, that was present in any of the other software tools tested, appears to be some form of English language type formula description tool similar to that found in The Spreadsheet Detective.

3. TEST RESULTS

3.1 Results by Software Tools

As can be seen in the Figures Three and Four, SpACE, The Spreadsheet Detective and Operis Analysis kit show test results of 43 (84%), 41 (80%) and 33 (65%). It is possibly significant however that the SpACE results were assisted by the fact that the test was carried out by an experienced SpACE user (although the results were allocated by the author), rather than by the author, who was new to all of the other software before the tests began. It should also be noted.
that even on a cell-by-cell inspection basis, the built in Excel functions only score 24 (47%), which suggests that all is not well and that the need to go beyond the built in auditing functions exists.

Operis Analysis Kit, which is probably the easiest tool to use and scored a respectable 33 (65%). The Excel Auditor only scored 14 (27%), although this was largely down to the method of testing as it needs to be used as a tool to assist in a cell-by-cell inspection, whereas it was tested on the same basis as the other three add-in packages.

During the test, it became apparent that certain functions were more useful than others, both in the Sample Error Spreadsheet test itself, and in the familiarisation stage of the test where other functions were investigated.

Possibly the most essential ability for a software tool to possess is the ability to recognise formula schema. It is this ability that lifts the software from being a tool to assist in a cell-by-cell inspection, to a tool that means a time-consuming cell-by-cell inspection can be avoided.

A second function that could be classed as essential is the ability to provide the user with a visual overlay on the worksheet, identifying different schema/different types of cells etc. Falling into the 'very useful' category are functions such as supplementary potential error cells reports, the option to search for particular cells that are prone to errors, such as constants in formulae or un-referenced cells, the ability to identify unprotected cells, and the use of English language formula descriptions based upon column and row labels.

It is in these functions that SpACE and The Spreadsheet Detective excel, and that enabled them to score so highly in the tests. Other features such as documentation and development tools, whilst not contributing to the test results due to the nature of the Sample Errors Spreadsheet, could certainly be useful, depending upon who the user of the software is and the role they perform i.e. developer/auditor/end-user.
### Figure Four – Detailed Test Results

#### 3.2 Results by Type of Error

When the results of each of the software tests are combined, it is possible to gain an overview of the levels of success that were achieved in relation to each type of error. From the results of the test, it is obvious that the software tools tested performed to a reasonable standard on all but the omission errors, for which the average result was 0.8, far below the average results reported for the other three error types. This is not altogether surprising, as these errors are the type that an auditor is more likely to identify with the human eye than with a software tool.

Those tools that highlighted these omission errors relied upon non-referenced cells elsewhere in the spreadsheet model, and identifying label/formula patterns. The other three error types all achieved good results in the tests, although the mechanical errors did not quite match the qualitative and logical errors. Perhaps the most surprising result was the success of the qualitative errors, although these were improved slightly by having a particularly easy to spot (yet quite common) error in four different places. Considering that a definition of a qualitative error is that it does not (yet) have an impact on the bottom line figures, it was anticipated that these would be particularly difficult to find. The results on all but the omission errors do suggest however that software tools can have a positive effect on software auditing detection and efficiency.
3.3 Results by Individual Errors

Figure Five - Average Results for Individual Errors

![Bar Chart]

Formulas overwritten with values (errors 1 to 4) On these errors, the results averaged 2.6, a particularly successful result that illustrated how easy these errors can be to detect. Only The Excel Auditor of the add-in software tools tested, failed to achieve a 'passed' result. The success in finding these errors was largely down to the colour-coded overlays that could be applied to the worksheet making these errors stand out.

Fixed value used when a named cell reference should be used (error 5) This error achieved a respectable result of 1.8, which when the cell-by-cell inspection method used in the built in Excel functions test was removed, increased to 2.3. Both The Spreadsheet Detective and Operis Analysis Kit explicitly highlighted this error via an error report and a search option respectively. SpACE was close behind with a search option but this option needed the value to be specified by the user.

Presentation Illogical (error 6) The presentation illogical error proved very difficult to detect, with SpACE getting the closest by allowing certain text strings to be formatted with a different background colour meaning that text patterns were easier for the user to identify. Other than this, Operis Analysis Kit and The Spreadsheet Detective both partially identified this error by virtue of the fact that the adjacent cells had new formula schema.

Incorrect Presentation (error 7) This error surprisingly proved quite difficult to identify in the tests. However both The Spreadsheet Detective and SpACE correctly identified the unprotected range of cells despite having previously disabled the worksheet protection. The Excel Auditor, Operis Analysis Kit and the built in Excel functions all failed to identify the erroneous cells, although The Excel Auditor did indicate which worksheets were protected.

Incorrect Summation (errors 8 to 10) These errors received a reasonable average result of 1.8. The more successful methods of finding these errors were the graphical overlays, which showed new formula schema and/or highlighted the cells being totalled. The exceptional result on these errors was by The Spreadsheet Detective highlighting the incorrect row error (error 9) by using the labels to indicate, in English, that the values being totalled were not from the correct rows.
A formula copied down a column looks at the wrong column (errors 11 and 12) These errors were designed to be difficult to detect and the result of 1.6 for each was higher than expected. Although each of the software tools were capable of highlighting the formula and displaying the formula in a readable style, the nature of this error is such that even when the formula is displayed clearly, the error is not always obvious without some domain knowledge. The only software tool to achieve a 'passed' result in this test was The Spreadsheet Detective. This was largely down to the English descriptions. SpACE achieved on 'almost-passed' thanks to its failure to establish a link between the source data and the bottom line cell, which prompted further investigation that subsequently detected the error.

Incorrect Summation of a column including subtotals (error 13) Whilst a visual overlay of this formula showed which cells were included in the formula, the nature of the error meant that this error could still be overlooked. 'Passed' results were achieved by The Spreadsheet Detective, Operis Analysis Kit and SpACE by highlighting the cell as a prospective error due to the fact that it contained blank cells.

Percentages added rather than calculated on a total cell (error 14) With a graphical representation of the formula in question this error is relatively easy to detect. The Spreadsheet Detective and SpACE both achieved a 'passed' result on this error by virtue of identifying a unique formula which when viewed with the built in Excel functions became obvious. Operis Analysis Kit, however, failed this test as the total cell to the left was the same logical formula and so the cell was not highlighted as a distinct formula.

Percentages averaged rather than calculated on a total cell (error 15) Although superficially similar to the previous error, this error proved to be much easier to detect. The fact that it was an average of percentages made the error easier to detect even on a formula report. This error was also highlighted as a distinct formula in Operis Analysis Kit meaning that this achieved a 'passed' result on this test, whereas a 'failed' result was achieved by Operis Analysis Kit on the previous error.

A row missing from all of the worksheets (error 16) The omission errors were deliberately included as a difficult test for the software tools. The first was designed to provide an error that was expected to be much more likely to be detected by the human eye than a software tool. The fact that the row in question was removed from all of the worksheets means that the software would have to have some way of identifying label patterns to identify the absence of the row. Only SpACE came close to identifying the absence of the row, thanks to a combination of text background colour coding and the ability to recognise unique formula across 'blocks' of formulae.

A row missing from one of the worksheets although it exists on the others (error 17) This omission error was slightly easier to find than the previous error as the row of data deleted from the Report worksheet still existed on the File worksheet. Both SpACE and Operis Analysis Kit achieved a 'passed' result on this test due to the option to highlight cells with no dependents.

4. TEST CONCLUSIONS

The test has successfully provided the following answers to the questions identified at the outset of the test.
Are software-auditing tools for spreadsheets useful? The tests proved conclusively that spreadsheet auditing software tools can be useful. The more successful software achieved detection rates of over 80%. However it must be remembered that software tools do not detect and correct errors, they merely point the user in the right direction.

How do software-auditing tools assist the user in auditing spreadsheets? The software tools used various different methods of identifying potential errors, however the most successful used a combination of the following three features:

- Schema Identification
- Visual Methods
- Search for/Report on potential error cells

The software tools tested tended to be designed as add-ins to Excel, and as such could be used alongside the built in Excel Auditing functions.

What errors are software-auditing tools good/poor at identifying? Software tools proved to be particularly good at detecting qualitative, quantitative logical and quantitative mechanical errors in the test. Software tools proved somewhat less successful at detecting quantitative omission errors. Given the nature of these errors, this was not a surprising result.

5. OVERALL CONCLUSIONS

Spreadsheet errors are still a major problem. Research has repeatedly found that rates of spreadsheet errors are in line with those in more traditional programming techniques. In the author's professional experience as a developer of a number of spreadsheet models, spreadsheets are not allocated as much time and resources as a traditional programming language would be for testing and auditing purposes. Cell-by-cell inspection is largely unheard of, and even dummy data testing can be limited. These experiences are confirmed as commonplace by the academic research published, thus confirming that the author's experiences are not unique.

Research has taken place, which has found that spreadsheet auditing is easier using a visual approach. A visual approach lends itself to software tools. However to date there has been no published research into the usefulness of and methods used in, spreadsheet auditing software tools. This paper provides an initial investigation into software tools. The evidence shows that software tools are definitely useful in the detection of software errors, although they perform poorly on omission errors. The software tools potentially have three methods of assisting in the audit of a spreadsheet. The first is by applying a visual approach to auditing to the user, and the second is to identify cells that potentially contain errors. The third method available to software tools, and potentially the most powerful, is the identification of formula schema, which allows the user to adopt a 'focused cell inspection' approach to spreadsheet auditing, rather than a traditional cell-by-cell inspection approach.

6. AREAS FOR FURTHER STUDY

Extensive research into spreadsheet errors has taken place. This research has covered identifying and categorising errors, quantifying errors, methods of designing spreadsheets to avoid errors and error detection. There is however, very little research into the use of software tools...
tools to detect errors in spreadsheets. This paper attempts to redress this by testing a series of software tools against a spreadsheet with seeded errors. Whilst the tests proved that spreadsheet auditing software is useful, certain limitations in the software suggest that further study in this field would be beneficial. The nature of the test meant that the results were largely subjective. The test was primarily carried out by the author, who also designed the sample errors spreadsheet. This meant that the author had to speculate how much user interaction would be needed to identify an error and award a result accordingly. A more objective test would be to have the test carried out by a group of people who did not know the location of the errors before the tests. Another limitation of the test was that the sample errors spreadsheet was based primarily on a financial reporting model. Spreadsheets are used for many different applications other than just financial applications. It would be useful, therefore, to perform the tests on a number of spreadsheets from different fields, such as a results analysis spreadsheet or a 'what if' based spreadsheet.

REFERENCES


TEACHING SPREADSHEET DEVELOPMENT USING PEER AUDIT AND SELF-AUDIT METHODS FOR REDUCING ERRORS

Chadwick D., Sue R.,
Information Integrity Research Centre,
School of Computing & Mathematical Sciences,
University of Greenwich, London SE10 9LS, United Kingdom
Phone:+44 0208 8331 8509   Fax: 0208 8331 8665  Email:cd02@gre.ac.uk

Abstract
Recent research has highlighted the high incidence of errors in spreadsheet models used in industry. In an attempt to reduce the incidence of such errors, a teaching approach has been devised which aids students to reduce their likelihood of making common errors during development. The approach comprises of spreadsheet checking methods based on the commonly accepted educational paradigms of peer assessment and self-assessment. However, these paradigms are here based upon practical techniques commonly used by the internal audit function such as peer audit and control and risk self-assessment. The result of this symbiosis between educational assessment and professional audit is a method that educates students in a set of structured, transferable skills for spreadsheet error-checking which are useful for increasing error-awareness in the classroom and for reducing business risk in the workplace.

Keywords: spreadsheets, errors, audit, risk-management, teaching

1.0 THE PROBLEM OF SPREADSHEET ERRORS
The world-wide problem of spreadsheet errors and the need to act to prevent, detect and minimise the effect of errors has been widely reported both in industry studies and in more formal academic research studies.

1.1 The Industry Experience
The Business Modelling Group at KPMG Consulting have been only too aware of the problems of spreadsheet errors and frequently cite the findings of a KPMG survey of financial models based on spreadsheets. The survey findings, reported in an article in Internal Auditing by Chadwick, D. [1], found that:

- 95% of models were found to contain major errors (errors that could affect decisions based on the results of the model),
- 59% of models were judged to have 'poor' model design,
- 92% of those that dealt with tax issues had significant tax errors,
- 75% had significant accounting errors.

These findings have been reinforced by a statement in the same article by David Finch, Head of Internal Audit at Superdrug plc, who believed that people shouldn't be too surprised at the high rate of spreadsheet errors as:

"The use of spreadsheets in business is a little like Christmas for children. They are too excited to get on with the game to read or think about the 'rules' which are generally boring…. There is often little control over end user developments in spreadsheets with little if any standardisation in development processes by users in different departments, little risk analysis and a general assumption that models, on which important business decisions are made, are accurate"

The UK Customs and Excise department collects VAT on behalf of the British government. Raymond Butler, a computer auditor in the department, has written extensively on the frequency of errors found in spreadsheets used by businesses to calculate the amount of VAT that they should pay.

Butler and his team have done much work on risk assessment methods to enable them to determine which VAT spreadsheets have a likelihood of error and need to be audited; Butler R. [13]. One of the questions the Customs & Excise audit team ask themselves is whether the organisation for whom the development is being made have an adequate policy governing development, testing and use of spreadsheet models and applications. They also
investigate whether there is evidence that this policy is observed and enforced.

Butler states that there are many examples of errors found in both field audits and experiments which show that good development practice is rarely codified into business procedures and even when it is, the rules and restrictions it requires are not followed to any significant degree.

1.2 The Academic Research Experience

For the past 15 years, Professor Raymond Panko at the University of Hawaii has been studying and writing prolifically upon the phenomenon of spreadsheet error. In addition to compiling bodies of data from research establishments and businesses around the world he has conducted his own experiments with students. It is not possible to report upon all of Panko's findings in this paper but a recent paper of his shows a comparison of findings from several educational studies (see fig. 1.1).

Fig.1.1 Spreadsheet Development Experiments: Panko R. [12]

<table>
<thead>
<tr>
<th>Study</th>
<th>Year</th>
<th>Sample</th>
<th>Subjects</th>
<th>Spreadsheets</th>
<th>% with Errors</th>
<th>Cell Error Rate (CER)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown &amp; Gould</td>
<td>1987</td>
<td>ED</td>
<td>9</td>
<td>27</td>
<td>63%</td>
<td>NR</td>
</tr>
<tr>
<td>Olson &amp; Nilsen (1,2)</td>
<td>1987-88</td>
<td>ED</td>
<td>14</td>
<td>14</td>
<td>NA</td>
<td>21%</td>
</tr>
<tr>
<td>Lerch (1,2)</td>
<td>1988</td>
<td>ED</td>
<td>21</td>
<td>21</td>
<td>NA</td>
<td>9.3%</td>
</tr>
<tr>
<td>Hassinen (2) on paper</td>
<td>1988</td>
<td>Ugrad</td>
<td>92</td>
<td>355</td>
<td>55%</td>
<td>4.3%</td>
</tr>
<tr>
<td>Hassinen (2) online</td>
<td>1988</td>
<td>Ugrad</td>
<td>10</td>
<td>48</td>
<td>48%</td>
<td>NR</td>
</tr>
<tr>
<td>Janvrin &amp; Morrison (3)</td>
<td>1996</td>
<td>Ugrad</td>
<td>78</td>
<td>61</td>
<td>NR</td>
<td>7% to 10%</td>
</tr>
<tr>
<td>Study 1, alone</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Janvrin &amp; Morrison (3)</td>
<td>1996</td>
<td>Ugrad</td>
<td>88</td>
<td>44</td>
<td>NR</td>
<td>8%</td>
</tr>
<tr>
<td>Study 1, dyads</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Janvrin &amp; Morrison (3)</td>
<td>1996</td>
<td>Ugrad</td>
<td>88</td>
<td>88</td>
<td>NR</td>
<td>8% to 17%</td>
</tr>
<tr>
<td>Study 2, alone</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kreie (post test)</td>
<td>1997</td>
<td></td>
<td>73</td>
<td>73</td>
<td>42%</td>
<td>2.5%</td>
</tr>
<tr>
<td>Teo &amp; Tan (4)</td>
<td>1997</td>
<td>Ugrad</td>
<td>168</td>
<td>168</td>
<td>42%</td>
<td>2.1%</td>
</tr>
<tr>
<td>Panko &amp; Halverson, alone</td>
<td>1997</td>
<td>Ugrad</td>
<td>42</td>
<td>42</td>
<td>79%</td>
<td>5.6%</td>
</tr>
<tr>
<td>Panko &amp; Halverson, dyads</td>
<td>1997</td>
<td>Ugrad</td>
<td>46</td>
<td>23</td>
<td>78%</td>
<td>3.8%</td>
</tr>
<tr>
<td>Panko &amp; Halverson, tetrads</td>
<td>1997</td>
<td>Ugrad</td>
<td>44</td>
<td>11</td>
<td>64%</td>
<td>1.9%</td>
</tr>
<tr>
<td>Panko &amp; Sprague (4)</td>
<td>1999</td>
<td>Ugrad</td>
<td>102</td>
<td>102</td>
<td>35%</td>
<td>2.2%</td>
</tr>
<tr>
<td>Panko &amp; Sprague (4,5)</td>
<td>1999</td>
<td>MBA (NE)</td>
<td>26</td>
<td>26</td>
<td>35%</td>
<td>2.1%</td>
</tr>
<tr>
<td>Panko &amp; Sprague (4,6)</td>
<td>1999</td>
<td>MBA ED</td>
<td>17</td>
<td>17</td>
<td>24%</td>
<td>1.1%</td>
</tr>
<tr>
<td>Panko &amp; Halverson, monads</td>
<td>2000</td>
<td>Ugrad</td>
<td>35</td>
<td>35</td>
<td>86%</td>
<td>4.6%</td>
</tr>
<tr>
<td>Panko &amp; Halverson, triads</td>
<td>2000</td>
<td>Ugrad</td>
<td>45</td>
<td>15</td>
<td>27%</td>
<td>1.0%</td>
</tr>
<tr>
<td>Total Sample</td>
<td></td>
<td></td>
<td>998</td>
<td>1170</td>
<td>51%</td>
<td>(7)</td>
</tr>
</tbody>
</table>

NR = not reported               ED = experienced developer
NE = not very experienced with development at work           Ugrad = undergraduate students

(1) Measured errors before subject had a chance to correct them; (2) Only measured error rate in formula cells; (3) Only measured error rate in cells linking spreadsheets; (4) Wall Task designed to be relatively simple and free of domain knowledge requirements; (5) MBA students with little or no development experience; (6) MBA students with considerable development experience; (7) Weighted average
Panko’s figures show that, over the given experiments, 51% of the spreadsheets contained errors. Of course no mention is made of the type of errors and their severity in terms of spreadsheet integrity; this aspect is more fully reported in industry based studies (see KPMG survey reported above). However, more importantly from the teaching view, Panko goes on to state:

“These studies… used a variety of subjects from rank novices to highly experienced spreadsheet developers. All subject groups made errors, and when Panko and Sprague [1999]* directly compared error rates for undergraduates, MBA students with little or no spreadsheet development experience, and MBA students with at least 250 hours of spreadsheet development experience, they found no significant differences in error rates across the groups”


2.0 THE CAUSES OF ERRORS DURING TRAINING

There is a common problem with students learning new computing skills. Research conducted by Van Vliet et al [2] into comparing self-appraisal and objective tests of learners abilities has indicated that novices of both sexes consistently overrate their own computer literacy skills. This has led them to think they have correctly solved a problem when, in fact, they have not.

Panko R.[12], cites one experiment in which student spreadsheet developers were given a spreadsheet to build from a written specification. The 'developers' were then asked to estimate the likelihood that they had made an error during development. The median estimate was 10%, and the mean was 18%. In fact, 86% had made an error in their spreadsheet. When debriefed in class and asked to raise their hands if they thought they were among the successful 14%, well over half of all subjects raised their hands.

Research conducted by Chadwick et al. [3] has also shown that student spreadsheet builders frequently make errors that are quite confident are not there and are often amazed when the errors are pointed out to them. The same research has catalogued the different types of common errors and has shown that, amongst student spreadsheet builders, there is a tendency to create formulae that do not accurately model the calculations and processing phenomena of the real-world, and to copy formulae incorrectly from one part of a spreadsheet model to another. An assertion made in Chadwick et al. [3] is that even when training is given it too frequently concentrates on ‘how to do things correctly’ and often ignores ‘how to avoid doing things incorrectly’. One possible way of ‘avoiding doing things incorrectly’ is to make novice spreadsheet builders more aware of common errors and encourage them to apply checking controls during development commensurate with the amount of risk associated with producing a possibly incorrect model. This is supported in the professional audit literature reviewing the teaching of computer auditing:

"In terms of teaching …several activities can help students …understand that control should be used sparingly but appropriately; the right amount of control depends on the associated risk” Herremans [4]

During the exercise reported in this paper, students were specifically asked to assess the risks inherent in any model they built. Error-awareness skills were also extensively developed by providing opportunities for students to check for errors:

1. through self-assessment by error-auditing their own spreadsheet models and also those deliberately seeded with errors for training purposes,
2. through peer-assessment by auditing the models of other students.

3.0 SELF AUDIT

Self-assessment performed by students has been gaining in popularity in the last few years as a means of enabling students to be reflective on the quality of their own work. Such an approach has been found to be effective in improving intellectual internalisation and intuition [9]. During the trials reported herein self-assessment techniques were based upon the self-audit methods in use in industry. Formal lecture sessions were given in which students were encouraged to consider the advantages and disadvantages of spreadsheet audit in general. The main lecture themes were designed to improve awareness of:

- the usefulness of auditing spreadsheet models,
• the need to find errors, their causes and effects,
• the business risks associated with incorrect spreadsheets,
• the common errors by being exposed to them through seeded models,
• the advantages of using self-audit check lists during development.

3.1 Improving Awareness of the Usefulness of Self-Auditing

Students were made aware that Control Self Assessment (CSA) techniques were accepted audit methods in industry. They were asked to consider the advantages and disadvantages of performing self-checking as in fig 3.1.

Fig 3.1 Control Self Assessment (CSA)

<table>
<thead>
<tr>
<th>CSA: Advantages</th>
<th>CSA Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can build CSA into own work schedule</td>
<td>May be tendency not to do at all</td>
</tr>
<tr>
<td>Learn by detecting own errors</td>
<td>May be that not all errors get detected</td>
</tr>
<tr>
<td>Greater awareness so fewer errors later</td>
<td>May become blasé and so negligent</td>
</tr>
<tr>
<td>Etc ...</td>
<td></td>
</tr>
</tbody>
</table>

3.2 Improving Awareness of Errors, Their Causes and Effects

Students were encouraged to identify the events during development which could create, propagate or exacerbate errors. They were asked to tabulate what they thought were possible error-events, their causes and possible effects as in fig. 3.2.

Fig. 3.2 Sample Error-Event Table: Error Events, Causes, Effects

<table>
<thead>
<tr>
<th>Error-inducing events identified?</th>
<th>Causes?</th>
<th>Effect?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incorrect Formulae</td>
<td>1. Typographical error</td>
<td>Wrong output data</td>
</tr>
<tr>
<td></td>
<td>2. Wrong arithmetical precedent rules</td>
<td>leading to faulty decision-making</td>
</tr>
<tr>
<td>Mistakes in input data-set</td>
<td>1. Typographical error</td>
<td>Wrong output data</td>
</tr>
<tr>
<td></td>
<td>2. Wrong data-set</td>
<td>leading to faulty decision-making</td>
</tr>
<tr>
<td></td>
<td>3. Wrong data source</td>
<td></td>
</tr>
</tbody>
</table>

3.3 Improving Awareness of Business Risks

As part of their introduction to CSA, students were told that organisations often extended CSA to consider the inherent risks (Control and Risk Self-Assessment or CRSA). CRSA, in this context, involved personal awareness of the risks inherent in a spreadsheet project, the possible error situations, alternative arrangements a client may need in event of failure as well as the developer’s own response to failure and problems. Students were encouraged to identify the possible risks inherent in an application they had been asked to develop and to identify the minimum risk-set.

Fig 3.3 Minimum Risk Set Compiled By A Typical Student

<table>
<thead>
<tr>
<th>Risk Self Assessment of Errors In My Spreadsheet</th>
</tr>
</thead>
<tbody>
<tr>
<td>How important to my client is the spreadsheet I am developing?</td>
</tr>
<tr>
<td>Which parts of the spreadsheet model are most critical to my clients business?</td>
</tr>
<tr>
<td>What compensation might the client demand from me for any error?</td>
</tr>
</tbody>
</table>
3.4 Improving Awareness Using Specially Seeded Models For Training

Improving awareness of errors was accomplished by presenting students with spreadsheet models deliberately created with errors and asking them to identify the errors and assess the extent of commercial risk associated with them. An introductory example from the teaching materials is shown in 3.4.1 (there were more complicated models!).

3.4.1 Example of An Error Seeded Model

Bungee Breakspeare wants to open his own night-club and has put details of expected income and out-goings in a spreadsheet which he has sent to his bank-manager for a start-up loan. (see fig 3.4).

**Question:** Which parts of the spreadsheet carry the most likelihood of an error?

**Question:** Is there an error in this spreadsheet?

**Question:** What business risks are associated with an error in this model?

Fig. 3.4 Example of Error identification and Risk Assessment question

<table>
<thead>
<tr>
<th>Figures in £</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>June</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entrance Ticket sales</td>
<td>4,000</td>
<td>3,500</td>
<td>3,000</td>
<td>3,000</td>
<td>4,000</td>
<td>5,000</td>
</tr>
<tr>
<td>Bar sales: drink and food</td>
<td>1,500</td>
<td>1,000</td>
<td>1,000</td>
<td>1,000</td>
<td>2,000</td>
<td>2,000</td>
</tr>
<tr>
<td><strong>TOTAL INCOME</strong></td>
<td><strong>5,500</strong></td>
<td><strong>4,500</strong></td>
<td><strong>4,000</strong></td>
<td><strong>4,000</strong></td>
<td><strong>6,000</strong></td>
<td><strong>7,000</strong></td>
</tr>
<tr>
<td>Wages: bouncers and bar staff</td>
<td>1,000</td>
<td>1,000</td>
<td>1,200</td>
<td>1,200</td>
<td>1,500</td>
<td>1,500</td>
</tr>
<tr>
<td>Electricity charges</td>
<td>0</td>
<td>3,500</td>
<td>0</td>
<td>0</td>
<td>4,000</td>
<td>£0</td>
</tr>
<tr>
<td>Rent for premises</td>
<td>0</td>
<td>0</td>
<td>5,500</td>
<td>0</td>
<td>0</td>
<td>5,500</td>
</tr>
<tr>
<td><strong>TOTAL OUTGOINGS</strong></td>
<td><strong>1000</strong></td>
<td><strong>4500</strong></td>
<td><strong>6700</strong></td>
<td><strong>1,200</strong></td>
<td><strong>5,500</strong></td>
<td><strong>7,000</strong></td>
</tr>
<tr>
<td><strong>MONTHLY PROFIT</strong></td>
<td><strong>4,500</strong></td>
<td><strong>-2,700</strong></td>
<td><strong>2,800</strong></td>
<td><strong>1,200</strong></td>
<td><strong>500</strong></td>
<td><strong>0</strong></td>
</tr>
<tr>
<td><strong>ACCUMULATING PROFIT</strong></td>
<td><strong>4,500</strong></td>
<td><strong>4,500</strong></td>
<td><strong>-2,700</strong></td>
<td><strong>100</strong></td>
<td><strong>600</strong></td>
<td><strong>600</strong></td>
</tr>
</tbody>
</table>

3.5 Practical Self Assessment Check List

The use of check lists is well documented in audit literature. Their use has also been championed in educational studies with business students learning to build spreadsheet models [8]. The checklist used with students in the research reported herein was a pre-event checklist asking students to consider appropriate action prior to building a spreadsheet.

Fig. 3.5 The Self-Assessment checklist stages

<table>
<thead>
<tr>
<th>Before starting your spreadsheet have you considered if and when to perform:</th>
<th>1. the RADAR development stages yes/no?</th>
<th>2. modularisation of the spreadsheet if necessary yes/no?</th>
</tr>
</thead>
<tbody>
<tr>
<td>3. creation of a logical model of the spreadsheet yes/no?</td>
<td>4. use of the 2 A’s : which functions to test yes/no?</td>
<td>5. create a User-Guide yes/no?</td>
</tr>
<tr>
<td>6. create a risk assessment table identifying risks yes/no?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(RADAR life-cycle, spreadsheet modularisation, logical models, and 2-A’s approach are spreadsheet building methods developed and taught at the University of Greenwich - for further information on some of these terms refer to Chadwick et al [5] and Rajalingham et al. [6] and [7]. For risk assessment see 3.3 above)

4.0 PEER ASSESSMENT

Peer assessment is frequently used as an educational tool because “of its potential for the development of students’ autonomy, maturity and critical abilities” [10]. The advantages are cited as giving students appreciation of other students’ work and thereby identifying common mistakes. However, there are generally two main criticisms from the students’ point of view. It is commonly stated that although the peer assessor is given the opportunity to review another students work, they, the peer assessor do not:
• gain any particular new skills by so doing,
• have their own proficiency at peer assessment itself assessed,
• understand why they should perform part of the tutor’s job.

These factors have been major drawbacks to the use of peer assessment generally. However, as a possible solution to this, the approach herein described uses methods derived from the peer audit functions commonly used in industry. Peer audit is a method promoted by internal audit departments whereby application development projects are checked against corporate standards by persons not involved in the original development but of equal skill and business knowledge to the original developers. Such peer auditors are commonly trained in basic audit techniques and taught how to report their findings. Significant savings have been made by companies employing this approach as it reduces the need for a costly central audit department. It also enables far more frequent reviews to occur and at much earlier stages in the application development life-cycle. A peer assessment method used in education and based on professional peer audit was considered to possibly have certain advantages which would address some of the criticisms given by students above. For the assessing students in particular, it was considered that they might develop a set of skills commensurate with those of practicing auditors and which would be immediately transferable to a work environment. Proficiency in such skills could also be educationally assessed.

The peer audit skills were taught using lectures and practical exercises revolving around:

• improving awareness of the usefulness of peer-audit,
• a practical peer audit check list,
• practical guidance on writing a peer-audit report,
• exercises for peer audit practice.

### 4.1 Improving Awareness of the Usefulness of Peer-Audit

Students were introduced to peer-audit and encouraged to tabulate the advantages and disadvantages.

![Peer Audit Advantages and Disadvantages As Per Typical Student](PA: Advantages | PA: Disadvantages)

<table>
<thead>
<tr>
<th>PA: Advantages</th>
<th>PA: Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peers have similar spreadsheet skills</td>
<td>But may make same mistakes</td>
</tr>
<tr>
<td>Peers have similar domain knowledge</td>
<td>But it may be too limited</td>
</tr>
<tr>
<td>Etc…</td>
<td></td>
</tr>
</tbody>
</table>

### 4.2 Practical Peer Audit Check List and Peer Audit Report

Three types of practical audit methods were used to develop skills in searching for and identifying errors and omissions in other students’ work. Peer assessing students were encouraged to:

• find errors in other students’ spreadsheets using a post-event check list,
• prepare an audit report on their findings.

The post-event check list (fig. 4.2) was used by the assessing student to check that the assessed student had actually done what was required in terms of the standards of model development defined in the requirements specification given to them.
4.3 Exercises In Use of the Peer Audit Method

The peer audit method was used for practical assessment of each student’s spreadsheet. The peer exercise occurred in a university computing laboratory with a working version of the spreadsheet. This was in accordance with the work of Boud (in Heywood, 1989) who argued;

"peer assessment should be formalised in the laboratory. In this way students begin to take responsibility for their own learning and gain insight into their own performance through having to judge the work of others." [11]

Fig. 4.3 Peer Audit Sheet giving instructions on how to perform audit

| AUDITOR STUDENT NAME:……………… |
| AUDITED STUDENT NAME:……………… |

**PART A : AUDIT OF SPREADSHEET 1**

Qu 1 : Change the $ to £ exchange rate to 1.69. Change the D.Mks to £ exchange rate to 2.40. Change the Yen to £ exchange rate to 1300

The 3 Mthly Total (in I9) should change to 4773.99

**PART B : AUDIT OF SPREADSHEET 2**

Qu 2 : What is the figure for the 3 Mthly Total over all salespeople for April to June? Should be 47425.00

Qu 3 Does User Guide identify spreadsheet builder?

Qu 4 Does User Guide show the Date of Creation?

Qu 5 Does User Guide specify spreadsheet purpose?

Qu 6 Is there a lookup table for commission rates?

Qu 7 Check any 3 key functions. Are the functions correct?

**MARK GIVEN:**

Now Prepare Your Audit Report on Spreadsheet 2 model.
• no mutual-audit triples arose i.e. A audits B, B audits C, C audits A.

5.0 CONCLUSIONS

The audit approach was used on a cohort of second year undergraduate information systems students. These fairly novice students (in serious spreadsheet work) were selected for this project for two reasons. They were generally eager to participate in new assessment methods and also had not yet formed bad habits which resulted in poor spreadsheet model design. This is in keeping with the suggestion by Habeshaw et al. [10] and also by Chadwick et al. [1] that peer assessment techniques should be started early in the programme of study.

5.1 Analysis of Student Feedback On The Exercise

Anonymous feedback was obtained from 42 participating students who answered five questions and were permitted to give comments. All reported favourably on most criteria (except report writing) saying they had not only learned from the experience but also enjoyed it.

Fig.4.4 Feedback from students

<table>
<thead>
<tr>
<th>Yes</th>
<th>No/Not sure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELF AUDIT: was the exercise useful in exposing your mistakes?</td>
<td>50%</td>
</tr>
<tr>
<td>SEEDED MODEL: were these useful in increasing your ability to find mistakes?</td>
<td>65%</td>
</tr>
<tr>
<td>PEER AUDIT: was the exercise useful in developing your skills for finding errors in other people’s work?</td>
<td>72%</td>
</tr>
<tr>
<td>PEER REPORT: was the exercise useful in developing your report writing skills?</td>
<td>45%</td>
</tr>
<tr>
<td>PEER AUDIT: was being audited a learning experience for you?</td>
<td>55%</td>
</tr>
</tbody>
</table>

SELF AUDIT: was the exercise useful in exposing your mistakes?

Students commented favourably on the disciplined approach forced upon them. Although they identified the time overhead of having to think about inherent risks they felt that the insight gained into possible problem areas outweighed the time overhead.

SEEDED MODEL: were these useful in increasing your ability to find mistakes?

Students were keen on this method of raising error-awareness and requested more exercises of further complexity.

PEER AUDIT: was the exercise useful in developing your skills for finding errors in other people’s work?

Student were enthusiastic that having to audit each other’s work had been a beneficial exercise. Many remarked how surprised they were to see the variety of spreadsheet models produced from what they had considered to be requirements that only had ‘one’ solution.

PEER REPORT: was the exercise useful in developing your report writing skills?

The seemingly negative reply to this question may indicate that the question was wrongly asked on the follow-up questionnaire. It may well have been that the report was not useful in ‘developing’ skills already in existence (all the students had already received much exposure to report writing opportunities).

PEER AUDIT: was being audited a learning experience for you?

There is no doubt, from the feedback received, that students were motivated to do well by the prospect of having their work examined by another.

5.2 Analysis of The Complete exercise

Crucial to the success of this exercise were:
• early explanation of the rationale for the peer and self-assessment,
• clear and simple instructions for the students to follow,
• objective criteria using prescribed test data, and
• monitoring of the exercise by the tutor through viewing the audit report.

The work continues with current cohorts of students and the methods used have been refined. The concept of peer and self audit has proved to be so useful with spreadsheet teaching that it is now being extended to the teaching of databases.

References


Spreadsheet assurance by “control around” is a viable alternative to the traditional approach

by Harmen Ettema, Paul Janssen and Jacques de Swart

PricewaterhouseCoopers

Amsterdam, The Netherlands

E-mail: harmen.ettema@nl.pwcglobal.com

ABSTRACT

The traditional approach to spreadsheet auditing generally consists of auditing every distinct formula within a spreadsheet. Although tools are developed to support auditors during this process, the approach is still very time consuming and therefore relatively expensive. As an alternative to the traditional 'control through' approach, this paper discusses a 'control around' approach. Within the proposed approach not all distinct formulas are audited separately, but the relationship between input data and output data of a spreadsheet is audited through comparison with a shadow model developed in a modelling language. Differences between the two models then imply possible errors in the spreadsheet. This paper describes relevant issues regarding the 'control around' approach and the circumstances in which this approach is preferred above a traditional spreadsheet audit approach.

1 THE TRADITIONAL AUDIT APPROACH CAN BE VERY TIME CONSUMING

1.1 For moderately sized spreadsheets, traditional spreadsheet audit efforts are in the range of 25 to 75 hours

A moderately sized spreadsheet typically involves thousands of cells. Spreadsheet Audit (SA) tools like OAK [1] and Spreadsheet Professional [2], reduce the audit to the inspection of the distinct formulas only. The number of distinct formulas strongly depends on the nature of the spreadsheet, but for moderately sized spreadsheets this number typically lies in the range of 500 to 1,500. Besides the mentioned SA tools, features in spreadsheet programs like “trace dependents” and “trace precedents” in Microsoft Excel, help the spreadsheet auditor to validate distinct formulas quickly. But still, as a rule of thumb, the inspection of a distinct formula by an experienced auditor takes on average around 3 minutes. The total effort of such a traditional spreadsheet audit, or in assurance terms control through approach, generally takes from 25 up to 75 hours. This amount of time is often out of proportion compared to the development of the spreadsheet. On the other hand, in many cases the assurance of the spreadsheet is still required by users or third parties, which decide to invest large sums of money based on the spreadsheet calculation.

1.2 For large spreadsheets, traditional SA often is not an option because throughput time is too long

If the number of distinct formulas is much larger 1,500, then not only the time spend and the audit costs are an obstacle for following the traditional SA approach, but also the throughput time. For complicated, badly structured spreadsheets, it can be hard to perform the SA by more than one auditor in parallel. An example of such a complicated, badly structured spreadsheet is a spreadsheet with on one worksheet thousands of distinct and repetitive formulas. Consequently, the throughput time for these spreadsheets will be, based on the previous assumptions, more than two weeks. Because users want to take the spreadsheet in production as quick as possible the time-pressure on the SA is very high, and therefore often limited to one or two weeks.
2 THE CONTROL AROUND APPROACH EXPLOITS ADVANTAGES OF MODELLING LANGUAGES

2.1 The control around approach relies on a shadow model that validates output in the spreadsheet

In our proposed audit approach, we build a shadow model of (parts of) the spreadsheet model in a modelling language. The shadow model consists of the formulas in the spreadsheet that need to be audited. By importing scenarios of input data in the spreadsheet as well as in the shadow model, one can compute various scenarios according to the shadow model and compare it to the output in the spreadsheet. These scenarios should reflect a relevant set of cases of input data for the spreadsheet model. Defining the right test set is crucial in a shadow modelling approach. By focusing on the input/output relationship of data the validation of all intermediate calculations in the spreadsheet can be restricted to differences found between output of the spreadsheet and the shadow model. In appendix A we describe the basic features of this modelling language, which is integrated in a modelling environment called AIMMS [6]. Appendix B illustrates the building of such a model using a simple sales forecasting example in the field of telecommunication. In the remainder of this section, we highlight the advantages of using modelling languages for SA.

2.2 Defining the right test set of input data scenarios is crucial for a successful control around approach

Assurance on the spreadsheet is given by comparing the output of the spreadsheet with the output of the shadow model for a test set of input data scenarios. Therefore it’s crucial to compose a relevant test set. The test set can for example consist of a scenario with default values, scenarios with mutations in values for individual input variables and scenarios with mutations in values for combinations of input variables. A test set can also consist of scenarios with random numbers between possible values for each input variable. A way to overcome the disadvantage of a limited test set is to perform parallel processing for important calculations. This means that the output is not only produced with the spreadsheet model but also with the shadow model.

2.3 Separation of data and formulas is a key advantage in a modelling language

Although a decently designed spreadsheet should make a clear distinction between the regions in the spreadsheet that contain data manipulation and those that contain pure data, many spreadsheets encountered in practice lack this property. The separation of data and relationships is the cornerstone of a modelling language, in which only symbolical relations are defined, and the loading of the input data into the model parameters is a separate process (see Appendix A). Another property of properly built spreadsheets is that they use named ranges that make the formulas more readable and easier auditable. In practice, one often violates this design rule, because it is too time consuming to update the range names after adjusting the spreadsheets. In a modelling language, the formulas typically take the form of verbose English phrases that are easy to audit, because they are even understandable for people who are not familiar with the concept of modelling languages. An adjustment of the input data typically does not require the adjustment of the formulas.

2.4 Top-down approach is a logical consequence of a modelling language

Most spreadsheet builders try to follow some sort of top-down approach. This often comes down to arranging the workbook so that each worksheet contains a logical functional module and that data is “pushed” to the right and down through cells and worksheets within the spreadsheet. It is clear that such an approach greatly simplifies the work of the auditor of the spreadsheet. However, adjustments to the spreadsheet often lead to violating this principle. In a modelling language the model is of a hierarchical nature, which results in clear insight into the dependency structure.
2.5 Incremental software development approaches are hard to establish in a spreadsheet, but easy in a modelling language

2.5.1 Rapid Application Development is the state of the art incremental approach

Rapid Application Development (RAD) has become a popular way of designing computer applications in the past decade. Before RAD became the trend, one typically first made extensive, time-consuming specifications of what to build. The programming was only started after the approval of the specification. However, the shortcomings of the specification often appeared after the end-user had worked with the first version of the software. To deal with these shortcomings, one started the iteration loop specification adjustment – specification approval – programming – inspection by end-user. This iteration turned out to be very time consuming. RAD focuses on a rapid development of a first version that only contains the basic features of the application. The end-user inspects this version, comments on the basic features and indicates with which other features the application should be enriched. Then the iteration adjustment of existing features and programming of new features – inspection by end-user is started. In this iteration, the specification of the application, which is often difficult to understand for the end-user, is only used for internal use by the programmer. For a more elaborate discussion of the advantages of RAD, we refer to [3].

For SA, the use of RAD can be very beneficial, because in the first iteration one can focus on the building of a shadow model that computes the key output data in the spreadsheet. This could be done for a small data set only. For example, if the spreadsheet involves a financial forecast for several countries for several years, one could start to establish a shadow model for the first year and one country only. In the second iteration the data set could be enlarged and the computation of other, less important output data could be added to the shadow model.

2.5.2 Rapid Application Development is hard to establish in a spreadsheet

Due to the simple user interface, spreadsheets are easy to make. Even without or with only limited training it is possible to develop large spreadsheets with many features. However, adjustments and supplements, which are the basic ingredients of the RAD approach, are relatively hard/risky to incorporate in spreadsheets. An adjustment often requires the error-prone and time-consuming revision of many other cells. Adding an element to a dimension, e.g. adding an additional year to the time horizon, typically involves inserting additional rows or columns. As a consequence, all time dependent formulas and graphs have to be extended.

2.5.3 Rapid Application Development is a natural concept in a modelling language

Due to its hierarchical structure, a modelling language is well suited for RAD, because it is easy to add branches to the model structure without adapting other branches. Enlarging a dimension is as easy as adding new records to database tables. Adding a dimension is equivalent to adding a index to all relevant parameters. E.g., making the model time dependent is just a matter of adding a time index to all relevant parameters, whereas for a spreadsheet, this might lead to rebuilding the spreadsheet from scratch. Since the Graphical User Interface (GUI) is orthogonal to the model, new features in the GUI do not interact with the model structure and are therefore not subject the model assurance. For more details on RAD in a modelling language, we refer to Appendix A.

2.6 Control around approach exploits a library of common formulas

2.6.1 Even common relationships have to be audited again for every spreadsheet

A large fraction of a typical spreadsheet consists of a set of common relationships which are used in other spreadsheets as well. For example, many spreadsheets use the same financial forecast formulas. However, since the data structure is different per spreadsheet, the auditor has to check these common formulas again
for every new spreadsheet. This makes the SA work along the traditional approach not only quite boring, but also inefficient.

2.6.2 The shadow model is built using a library of well-tested, robust, state of the art, common relationships

Instead of checking the same formulas over and over again, an auditor who uses the control around approach typically builds these common formulas for the shadow model once. In a modelling language these formulas are in readable English and are therefore easily confirmed using a standard text book from the literature. The auditor collects these formulas in a library. This library can be linked to the shadow model for the new spreadsheet that has to be audited. At PricewaterhouseCoopers, we already have a wide collection of frequently occurring formulas, which makes the formulation of shadow models very efficient. In many cases the library formulas can be used directly. In the worst case, the data structures in the spreadsheets are so different, that the auditor has to change the index domain of the parameters occurring in the formulas. However, this does not affect the basic structure of the formula.

3 ESPECIALLY FOR HIGH-DIMENSIONAL MODELS THE CONTROL AROUND APPROACH IS BENEFICIAL

3.1 High dimensional problems create many distinct formulas in a spreadsheet

A spreadsheet is most convenient for two-dimensional models. The first dimension uses the rows, the second the columns. If the model contains a third dimension, then the first option is to create worksheets that contain individual slices of the third dimension. Relationships that depend on three dimensions are from a modelling perspective not more complex than those that depend on two dimensions. However, in a spreadsheet every slice of the third dimension generates a distinct formula, which has to be audited separately. Therefore, the larger the cardinality of the third dimension, the less efficient the traditional SA approach.

A formula that aggregates over the third dimension needs to contain references to all individual worksheets. For example, if the third dimension represents time in years, then every worksheet represents one year. The formula for the Net Present Value (NPV) would be a long expression with references to many sheets.

Another option for storing the third dimension is generating separate blocks on the same sheet, each block representing a slice of the third dimension. This option has to be selected if the model involves more than three dimensions. Again formulas with parameters defined over one or more dimensions that are represented by different blocks, generate more than one distinct formula.

3.2 In a modelling language every relationship is represented by one distinct formula independent of dimensions

Instead of auditing all distinct formulas in a spreadsheet used to represent one single relationship, the control around approach proposes to replace all these formulas by one formula in the shadow model. After feeding the input data from the spreadsheet in the shadow model, the shadow model is evaluated and its output compared with the output in the cells of the spreadsheet. This comparison process is often efficient, because it normally only involves the key output data, which are at an aggregated level.

Based on the previous discussion, it is possible to compare the efficiency of the traditional approach with the control around approach. Suppose that the model has $d$ dimensions, and that every dimension has the same number of levels. Furthermore, we assume that, due to the copying of the data to the modelling environment, the cost of auditing one relationship via the control around approach is twice as expensive as auditing one distinct formula via the traditional approach. The following graph shows the average effort of both approaches for auditing one relationship that involves all dimensions versus the number of levels for $d = 1, 2, 3$ and $4$, under ceteris paribus conditions. From this graph it is clear that for higher values of $d$, the
control around approach becomes less time consuming than the traditional approach for increasing values of the number of levels per dimension.

4 THE CONTROL AROUND APPROACH DELIVERS MORE THAN JUST ASSURANCE

4.1 The shadow model is a good starting point for strategic scenario analysis

Once the spreadsheet has been audited, one may decide to use the shadow model for the further development of the model. An interesting enhancement of a model is the study of the impact of different business strategies on the key performance indicators under several scenarios. In the field of Operations Research one often formulates an optimisation problem, in which the task is to find the optimal setting of the decision variables (the optimal business strategy) for different settings of the environmental variables. Once the model has been formulated in a modelling language, such optimisation problems can be easily defined and solved using state of the art solver like CPLEX [4] and CONOPT [5], which are available in the modelling environment.

4.2 The shadow model can be visualised efficiently

The Graphical User Interface of the modelling environment is well-suited for a quick development of a natural user interface. Moreover, it is easy to create a pdf report containing all conclusions and intermediate output.

4.3 The control around approach makes SA more exciting than the traditional approach

Instead of the tedious, monotone and often boring work involved in the traditional SA approach, the control around approach offers a more challenging type of work, for which it is required to think about the structure of the model on a conceptual level.

4.4 The control around approach gives an overall assurance, while traditional assurance is focused on accuracy of formulas

The functionality of the spreadsheet is rebuilt in a shadow model, based on the spreadsheet, conversations with spreadsheet users and available documentation. With this shadow model the spreadsheet is audited.
Therefore, the control around approach gives an overall assurance on the relevant input/output relationships as covered by the test set, while the traditional method is focused on individual calculations. The extra element of interpreting the formulas, rebuilding the model and comparing the results gives not only assurance on correct formulas but also assurance on the accuracy of the model as a whole: does the model do what it is supposed to do.

4.5 Should spreadsheets be used at all?

After presenting the list of advantages of using the control around approach above the traditional SA approach for complex multidimensional spreadsheets it’s justified to ask the question if we should use spreadsheets at all? It is clear that from the perspective of modelling strength we prefer modelling languages above spreadsheets. The separation of data and formulas, the hierarchical structure of the formulas and the scalability and therefore ability for incremental software development methods are key advantages of modelling languages. Although the strength of modelling is of course just one of the software selection criteria, the importance of this criteria is often underestimated. For complex models the modelling strength of the underlying software is the key success factor for the prevention of modelling and user errors. Software for supporting company critical decisions should therefore be reviewed twice on their modelling strength.
APPENDIX A: BASICS OF MODELLING LANGUAGES

In this appendix we describe a few basic features of the modelling language AIMMS [6]. Other modelling languages have similar features. For a more elaborate description, we refer to [6].

In translating real-life problems into AIMMS models, several conceptual steps are required. First, the input and output data are described using *sets* and *indexed parameters*. *Sets* are entities that determine the size of the problem. For example, in a transport problem, the cities between which transport takes place form such an entity. With these entities a number of instances is associated. For example, one of the cities could be ‘Amsterdam’. However, to keep the model generic and maintainable, it is customary and desirable to translate the problem into a symbolic model and not to make any explicit reference to individual instances. Such high-level model specification can be accomplished through the use of sets, each with an associated index for referencing arbitrary elements in that set. AIMMS allows for hierarchical set structures. E.g., origin cities and destination can be subsets of one set *Cities*. *Indexed parameters* are used to store data that can be associated with a particular element or tuple of elements. For example, if \( o \) is an index in the set *Origins* and \( d \) is an index in the set *Destinations*, then the distances between origin cities and destination cities can be stored in the parameter \( \text{Distance}(o,d) \). Data can be fed into the input parameters via the AIMMS Graphical User Interface (GUI) or via links with ASCII files, databases, spreadsheets or internal procedures that construct the input data.

Second, the mapping between input and output parameters is described using AIMMS definitions or procedures. Procedures are used for output parameters that cannot be expressed in terms of other parameters in closed form. For example, there is no closed form for the Internal Rate of Return (IRR). AIMMS definitions can be entered together with the declaration of the output parameters in a concise and verbose form and are often powerful enough to describe very complex relationships. As an example we give the definition of the largest national distance:

\[
\text{LargestNationalDistance} := \max((o,d) \mid \text{Country}(o) = \text{Country}(d), \text{Distance}(o,d)).
\]

Here, the input parameter \( \text{Country}(c) \) specifies in which country the city is situated, where \( c \) is an index in *Cities*, and the expression \( (o,d) \mid \text{Country}(o) = \text{Country}(d) \) assures the maximum is taken over distances between cities which lie in the same country.

Third, the model is executed. For output parameters that are computed via a procedure, this means that a call to this procedure has to be made. Parameters with a definition are automatically evaluated as soon at least one input parameter on which the definition (in)directly relies is changed.

The AIMMS model can be defined in a point-and-click manner via a GUI, which assures that the user has to learn hardly any syntax.

APPENDIX B: EXAMPLE OF SA BY CONTROL AROUND

The following case is based on a client case. For the purpose of illustration and competition considerations, the data is fictitious and some case elements have been strongly simplified.

Consider a company that studies the possibility of investing in a data telecommunication network. To assess the feasibility of this project, the Future Cash Flows (FCFs) generated by the network have been modelled as follows: For the next 20 years, estimates of the volume of communications over the network for three different scenarios (worst case, base case and best case) have been made. In the proposed business model, users of the network pay a fixed amount per transmitted byte, independent of the trajectory in the network. However, due to regional differences in tax legislation, the gross margin for the company depends on the specific trajectory. The major investments in the network are the acquisition and engineering costs for the glass fibre cables. These investments are done in the first year and come down to a fixed price per mile. For simplicity of illustration, we neglect other investments here.

The company has constructed a spreadsheet to implement this model. In this spreadsheet, the communication network is depicted as a matrix in which the rows represent the origins and the columns the
destinations. One worksheet contains a distance matrix\(^1\), a matrix that specifies the gross margin as function of the trajectory in the network and the following constants: investment costs per mile, Weighted Average Costs of Capital (WACC) and the sales price per byte of transmitted data. For every year in the time horizon of 20 years, a separate worksheet contains the three scenarios for the communication volumes. In the same worksheet, a formula is included that computes the net revenues per scenario for the corresponding year. The Net Present Value (NPV) is computed from a formula, which involves cells from all of these 21 worksheets, in a separate results worksheet.

Since one of the most important Key Performance Indicators to the company is the Net Present Value (NPV), the SA focuses on auditing the formulas for the NPV. Using the traditional SA approach, the auditor first has to validate the formulas for the net revenues for each year individually. Next, he/she has to validate whether the NPV formula contains the correct references to the individual worksheets. Using our proposed control around approach, we build a shadow model as follows. First, we identify four dimensions: time \((t)\), Origin \((o)\), Destination \((d)\) and Scenario \((s)\). Second, we declare the following parameters:

\[\text{Volume}(o,d,t,s), \text{GrossMargin}(o,d), \text{Distance}(o,d), \text{InvestmentPerMile}, \text{WACC}, \text{PricePerByte}\]

and link the data in the corresponding matrices and cells in the spreadsheet to these parameters. Third, we define the following parameters:

\[\begin{align*}
\text{Investment}(t) & := \text{IF } (t = \text{FIRST(Time)}) \text{ THEN SUM}((o,d), \text{Distance}(o,d) \times \text{InvestmentPerMile} \text{ ELSE 0 ENDIF}; \\
\text{FCF}(s,t) & := \text{Volume}(o,d,t,s) \times \text{GrossMargin}(o,d) - \text{Investment}(t); \\
\text{NPV}(s) & := \text{SUM}(t, \text{FCF}(s,t)/(1 + \text{WACC})^{(t - 2001)}); \\
\end{align*}\]

Finally, we compare the data in \(\text{NPV}(s)\) with the three corresponding cells in the results worksheet.

From this example we see that in the control around approach, there are only three formulas, which are easy to audit due to their verbose nature, as opposed to at least 23 distinct formulas in the traditional approach, which are harder to audit because of their less verbose nature and considerable size (e.g. for the NPV). Furthermore, the visualisation features in AIMMS make it easy to draw the network in the form of a graph, in which the origins and destinations are represented by nodes, and the trajectories by arcs. Especially when the geographical coordinates of the origins and destinations are known and used for positioning the nodes of the graph, this type of visualisation is very powerful. An example of such a Graphical User Interface (GUI) is shown in the figure below. In order to inspect the model data further, the user can select for which year and which scenario the data is displayed along the arcs of this graph via the drop down lists in the upper right corner.

\(^1\) In the distance matrix, only cells that refer to direct trajectories, i.e. trajectories which do not involve other points in the network than the origin and the destination, are non-empty. Furthermore, to avoid double counting, the matrix is lower triangular.
REFERENCES


Session 4.

Audit and alternative approaches (continued)

Chairperson: Patrick O’Beirne
Systems Modelling Ltd., Ireland

Papers in this session:

“A real alternative to spreadsheets”
*Dane Knight, Brixx, UK*

Quote: “*When spreadsheets were becoming widely adopted in the early 80’s, (when the desktop PC first became available), they were regarded as the new paradigm in business modelling. There has been no significant advance in the technology since that time.*”
(Dane Knight)
A REAL ALTERNATIVE TO THE SPREADSHEET

Dane Knight,
39 Skylark Meadows,
Fareham,
HANTS, UK PO15 6TJ
Danek@Brixx.com

ABSTRACT

In the search for alternatives to spreadsheets, questions are raised as to the whether the user community really desires an alternative and whether both the users and the software authors are prepared to pay the price in order to find an acceptable solution. There seems to be at least one alternative solution available. The combination of the latest object technology combined with a new approach offers a genuine alternative to spreadsheets that goes some way towards satisfying the identified resistance. While challenges still remain, the future could hold an exciting new power and capability for all business modellers to meet the more complex modelling challenges of the future.

BACKGROUND

When spreadsheets were becoming widely adopted in the early 80's, (when the desktop PC first became available), they were regarded as the new paradigm in business modelling. There has been no significant advance in the technology since that time.

Most companies are still using spreadsheets for most forms of business modelling. Models can range from simple project plans through to budgets and complex transaction models for mergers & acquisitions or financial re-structurings. There are many large international corporations which still have their business plans prepared in Microsoft Excel. A study in 1999 by a large high street bank in England identified over 27,000 different spreadsheet models in use within their organisation for decision support. The resulting risks in terms of duplication of effort, exposure to error and lost business knowledge are well understood and documented.

Despite the wide availability of customised software products to perform specific applications, e.g. departmental budgeting, company valuation, cash-flow projections etc, the only product widely regarded today as having the flexibility to build any type of financial model, is the spreadsheet.

It would be easy to suppose therefore that the spreadsheet is so technically superior, that nothing can replace it. Technology however, is not the problem. The problem facing spreadsheet alternatives is that spreadsheet alternatives are not as approachable or flexible from the user point of view.
In addition, there are some highly charged emotional issues surrounding spreadsheet use, such as product ownership, design & control, "feel good" factors, personal achievement, perceived speed, the ability to make reports look good, etc. Whether these issues are realistic or not, any real spreadsheet alternative faces the challenge of dealing with a high degree of user cynicism and emotional turmoil when they evaluate or use a technically enhanced solution.

Furthermore, more serious spreadsheet modelling requirements (generally known as business modelling requirements) are starting to emerge as a market segment in their own right. The business modelling segment requires more sophisticated solutions and flexibility to fulfil demand and meet expectations. For comparison of potential business modelling sector growth, the budget solution sector (Figure 1), over the last six years has been fuelled by database type applications which give a degree of flexibility with regard to reporting but have restrictions on structure and functionality when modelling any significant business scenarios.

![Graph showing MSUS data](image)

**Figure 1.**

**THE PROBLEM**

The real problem is that any realistic alternative to the spreadsheet cannot be merely technically better, it must *feel* better for users too. This is an emotional imperative since most users (despite being aware of spreadsheet shortcomings) continue to suffer significant biases due to emotional conditioning when evaluating technical alternatives. The spreadsheet is a 'feel good' tool. So much so that over the last ten years the spreadsheet has become second nature to many in business. The spreadsheet gives instant feedback and instant gratification to both novice and sophisticate alike. Notwithstanding their technical capabilities:

- Spreadsheets seem easy to use.
- The 'programming language' is almost a second language to many users.
- Users can see results very quickly in a format of their own design.
- It feels like initial progress is being made quickly.
- The user is in total control.

120
Spreadsheets are very personal. Users take such extraordinary pride in their models that they can put as much effort put into formatting a spreadsheet as they put into developing the underlying financial structure.

In reality, many corporate users do not want to adopt a more disciplined approach to spreadsheet modelling because it removes some of the emotional ‘comfort’ factors described above. Indeed, there is an even greater resistance to a disciplined approach when a structured approach is dictated by someone else - through a corporate spreadsheet modelling standard or through the intervention of an advisor for example.

“I’ll do it my way” is a familiar expression in this context.

Users will only be able to benefit from better modelling solutions providing two principles are accepted. One by the user and one by the software development industry:

1. Users must accept that the short-term trade-off of initial speed and ‘feel good’ factors in order to adopt a better discipline and structure is worth it. A change in approach will only be acceptable to the user community for demonstrable returns. Users must also accept that a new approach will also require training, which is something most spreadsheet users have previously avoided.

2. The software development industry must accept that the spreadsheet is not going to go away – at least for the time being. The key is to develop an approach that will provide synergy, by allowing tactical models to be done in spreadsheets, where they are strong, but enable strategic, complex and workgroup-based models to be easily created using a more rigorous approach. Any solution will need to be capable of holding a sensible dialogue with the spreadsheet, despite its limitations.

Any new solution designed to replace the spreadsheet must move the modelling capability forward to such a degree that it rapidly becomes a standard for use in future years. In the short term it is probable that the modelling market will fragment into two tiers. The traditional spreadsheet will continue to dominate ad-hoc desktop models, which are part of the generic ‘office’ capability. A new Business Modelling tier will emerge which requires a disciplined capability to meet robust and complex model development requirements driven by the increasingly dynamic business environment of the new economy.

NEW TECHNOLOGY

Object technology is becoming increasingly mature. The technology is now being adopted in application software rather than as a separate development or reporting tool. Several object oriented application software packages, which improve existing software solutions, are now available or are in development. They are suitable for diverse markets ranging from general ledgers to ERP systems. These of course, are all specific applications. The spreadsheet however, is a generic tool with no pre-designed structure. Can object technology support the degree of flexibility and free form capability of the spreadsheet?
Brixx is an object-based modelling package, which is both generic and comprehensive. Although Brixx has been available for over two years, it is still unique in its capability. It is only possible therefore to discuss object based modelling as an alternative to spreadsheets by focusing on the Brixx package, since Brixx is the only software package that has these capabilities. Moreover, Brixx addresses the technological and user emotional issues raised earlier in this paper.

![Brixx Multiple Screenshot](image)

Three attributes of the Brixx technology in combination establish it as a genuine paradigm shift in modelling:

1. It is entirely object based and uses the Microsoft standard com based objects. This enables the positive attributes of the object approach to be delivered. Whilst being easy to use, it provides better integrity and enables the creation of any mathematical or logical expression. Brixx supports automatic consolidation, large and complex models and is extremely flexible.

2. The Brixx PlanningObjects™ are unique in that they understand time. As every modeller knows, there are many ways in which time can impact a business model. Model builders have, however, become so used to ‘bending’ software to take account of time that they never really stop to think of the difficulties and consequences. Examples of how Brixx comprehends time include:
   - Brixx automatically supports changing periodicity from weeks to months to quarters etc and automatically updates consolidations as required.
   - Brixx supports the movement of a series of calculations from one quarter or year to another whilst maintaining the integrity of all reports.
   - Brixx permits the setting of time-based rules. For example, banking covenants can be based on net receivables or property covenants can be based on occupation.
   - Accounting reporting periods can be easily changed to review the impact, and then can be instantly changed back.
Data can be entered in one periodicity – monthly or yearly and then viewed in a different periodicity – weekly or quarterly. In Brixx the user can instantly view the same report for any time period and in any periodicity with no modification of the data or the report.

All the above issues can only be supported in spreadsheets with substantial difficulty and in many cases frequently require duplication of data and effort. PlanningObjects™ provide the ability to achieve any of the above in seconds and with integrity.

3) The application utilizes a chart of accounts structure that can be used for any application – even non-financial applications. This structure provides rigour and auditability to any model and also allows extreme flexibility in constructing complex models. Furthermore the overall structure of a model can be radically changed without affecting the integrity of results or reports.

But there is a catch.

As stated earlier, spreadsheet models give instant results and instant gratification. Models are frequently built in an ad-hoc fashion. Planners come back to models at a later date and add new sub-structures or modify sections to meet new requirements.

Spreadsheets give initial satisfaction followed by greater pain as the model becomes more mature or more complex. Of course more seasoned and experienced users will invest time in planning ahead and designing a more solid structure to begin with, but eventually the limitations of the spreadsheet will create difficulties. Frequently, the more solid and complex a spreadsheet structure, the more difficult it is to modify at later date.

Object technology requires the planning of the model up front to ensure correct equations and assumptions - i.e. the pain is up front. There is no instant gratification. However, it is possible to create a library of frequently used objects and this cuts down model development time considerably. The real gain is that modifying an object based model is easier and many of the issues surrounding spreadsheet structural changes are resolved automatically. Both the technology and the methodology are different and they both require patience to learn.

Do you remember the first time you tried to use a spreadsheet?

**TIME FOR CHANGE**

The challenge for Brixx, or any other emerging alternative, is whether users will be prepared to give up the quick and easy route to modelling. Users will not initially want to give up the easy approach and the complete freedom and control that spreadsheets give them.

In order to help overcome this natural resistance to change, alternative approaches to the spreadsheet must continue to allow some of the flexibility that has historically been taken for granted. New versions of the spreadsheet in the future are going to add little tangible improvements to the way spreadsheet modelling is done. The technology is now very old - some 18+ years. Microsoft has neither the need nor the motivation to invest in a total re-
design and re-write of the Excel spreadsheet, the global standard. Furthermore, any such attempt to re-engineer Excel would of course have to address the same challenges raised in this article. It is difficult to take something as important as the spreadsheet away from users and any alternative must work with spreadsheets for the foreseeable future.

A new paradigm must therefore provide a bridge for users to move across to new technology by making it approachable. A new paradigm must leverage the investment most organisations have made in spreadsheets. Furthermore, since much data has to be shared, potentially with users who only know how to use a spreadsheet, the ability to move or copy data to a spreadsheet must be present. With an easy transition to a new approach and a link to the existing investment in spreadsheets, the over-riding benefits of new technology should eventually convince even the most luddite of modellers.

CONCLUSION

There is a real alternative to the spreadsheet.

But are users prepared to be more disciplined in order to benefit from a better technology?

By analogy, if we want to be fit, we adopt a fitness regime and go to the gym. i.e. No Pain, No Gain. But how many users remember the pain of learning the spreadsheet the first time around? Users and organisations must be prepared to invest in some initial pain in order to reap the benefits of the next generation of modelling technology.

The design and capabilities of many business models - created by good and bad modellers alike - are driven and restricted by spreadsheet technology rather than being driven by business requirements. A spreadsheet model’s structure frequently meets the needs of the spreadsheet rather than the needs of the problem at hand.

Brixx uses object technology to implement a unique capability that delivers a real alternative to the spreadsheet, and meets both technical and emotive challenges.

The task for the future is to stretch the imagination such that object technology is effectively used in models which solve ever more complex and demanding problems. The power, breadth and depth of object technology will allow this to happen. Widespread use of new technology such as Brixx will generate more feedback. New ideas and capabilities will be incorporated into the further development of the technology. More user experience will also move object technology forward in exactly the same way that spreadsheet modelling moved forward nearly twenty years ago.

We are just at the beginning of a new era in modelling technology. The only limitations are our own imaginations and our natural resistance to change.
**AUTHOR'S INDEX**

<table>
<thead>
<tr>
<th>Author</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butler, Raymond</td>
<td>7</td>
</tr>
<tr>
<td>Chadwick, David</td>
<td>39, 95</td>
</tr>
<tr>
<td>Ettema, Harmen</td>
<td>107</td>
</tr>
<tr>
<td>Janssen, Paul</td>
<td>107</td>
</tr>
<tr>
<td>Knight, Brian</td>
<td>39</td>
</tr>
<tr>
<td>Knight, Dane</td>
<td>119</td>
</tr>
<tr>
<td>Nixon, David</td>
<td>79</td>
</tr>
<tr>
<td>O'Beirne, Patrick</td>
<td>3</td>
</tr>
<tr>
<td>O'Hara, Mike</td>
<td>79</td>
</tr>
<tr>
<td>Paine, Jocelyn</td>
<td>17</td>
</tr>
<tr>
<td>Raffensperger, John</td>
<td>61</td>
</tr>
<tr>
<td>Rajalingham, Kamalasen</td>
<td>39</td>
</tr>
<tr>
<td>Sue, Rodney</td>
<td>95</td>
</tr>
<tr>
<td>Swart, Jacques de</td>
<td>107</td>
</tr>
</tbody>
</table>
This volume contains the proceedings of the second European symposium on spreadsheet risks, audit and development methods, EuSpRIG 2001, held in July 2001 at Vrije Universiteit, Amsterdam, Netherlands.

The objective of this symposium is to promote discussion and co-operation amongst those concerned with authorising, auditing and developing spreadsheet models and by so doing, improve the reliability and integrity of information portrayed on spreadsheets.

The papers cover a broad spectrum of practical experience and research. The topic areas include the types of errors, development methodologies, audit practice and tools.

ISBN: 1 86166 179 7

Hosted and sponsored by:

vrije Universiteit amsterdam
Postgraduate programme EDP-Audit
Faculty of Economics and Business Administration

Main Sponsor:

PRICEWATERHOUSECOOPERS

Supported by:

University of Wales Institute Cardiff
University of Greenwich
British Computer Society CASG

Dutch Computer Society
ISACA NL and BeLux Chapters
NOREA

Sponsored by:

Information Systems Audit and Control Association
Northern UK Chapter

Supported by:

NgI
Information Systems Audit and Control Association

NOREA