European Spreadsheet Risks Interest Group

"Spreadsheet Risks, Audit and Development Methods"

Symposium Proceedings EuSpRIG 2000

17th – 18th July 2000
University of Greenwich, London, UK

Sponsored by

ISACA (Northern UK Chapter)

Supported by

University of Wales Institute Cardiff
ICAEW
British Computer Society CASG

the UNIVERSITY of GREENWICH
Information Integrity Research Centre
School of Computing & Maths Sciences
European Spreadsheet Risks Interest Group

EuSpRIG 2000 Symposium

"Spreadsheet Risks, Audit and Development Methods"

Editor:
David Chadwick

Symposium Sponsored By

Information Systems Audit and Control Association

Symposium Hosted By

the UNIVERSITY of GREENWICH

July 2000
Greenwich, London, UK
Proceedings of EuSpRIG 2000 Symposium
Spreadsheet Risks, Audit & Development Methods
Prepared for publication in 2000 by the Procurement and Business Services Department,
University of Greenwich

Held at the Maritime Greenwich University Campus,
University of Greenwich, London, UK; July 17th-18th 2000

© European Spreadsheet Risks Interest Group
www.gre.ac.uk/~cd02/eusprig/

ISBN: 1 86166 158 4

All rights reserved. Apart from fair dealing for the purposes of research or private study, or
criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, no part
of this publication may be reproduced, stored in a retrieval system, or transmitted in any form
by any means, electronic, mechanical, photocopied, recording or otherwise, without prior
permission of EuSpRIG.

Symposium Organised by
EuSpRIG Committee
in collaboration with
Information Systems Audit and Control Association (Northern UK Chapter)
www.isaca.org.uk/northern/
and
Information Integrity Research Centre, School of Computing & Mathematical Sciences
University of Greenwich, London SE10 9LS
www.greenwich.ac.uk/~cd02/iirc/

The use of registered names, logos, trademarks etc. in this publication does not imply, even in
the absence of a specific statement, that such names are exempt from the relevant laws and
regulations and therefore free for general use.

No responsibility is assumed by the Publisher for any injury and/or damage to persons or
property as a matter of products liability, negligence or otherwise, or from any use or
operation of any methods, products, instructions or ideas contained in the materials herein.

University of Greenwich, a charity and company limited by guarantee, registered in England
(reg. no. 986729). Registered Office: 30 Park Row, Greenwich, London SE10 9LS

Cover design by David Chadwick, graphics by Vanilla Beer at the University of Greenwich.
PREFACE

It's almost twenty years since the risks from end user computing in general and the use of spreadsheets in particular first appeared in the literature. Why are we holding the initial symposium of a Spreadsheet Risk Interest Group now, in the summer of 2000? Spreadsheet errors are still the rule, rather than the exception. In my own industry, around 80% of tax & duty spreadsheets tested by VAT inspectors contain errors. Jobs and money are lost through business mistakes based on defective spreadsheet developments.

Why? Some of the answer is in the gulf between the business and academic communities — they don’t read each other’s publications. Some is in the perception by auditors that spreadsheets are not ‘proper systems’, and by managers that spreadsheet developments are in some way trivial.

EuSpRIG's mission is to bridge this communication gap, by forging a partnership between business, academia and auditors. Indeed, the reviewers for some of the papers for this symposium have had to be quite firm in reminding authors of the intended business audience for these proceedings.

Fifteen months ago, four people who had each imagined themselves the only one to worry about spreadsheets got together to compare notes. Since that meeting we have formalised EuSprig into a properly constituted society, held this symposium and generated a fair amount of publicity in the professional press. Some of that publicity is reproduced in these proceedings.

Thanks are due to

- David Chadwick, who has carried the burden of organising this symposium and tirelessly publicising the issues and the group,
- Pat Cleary, our secretary for all the back-office work,
- The members of the EuSpRIG committee for their support,
- My colleagues on the Board of the Northern UK chapter of the Information Systems Audit & Control Association, for their financial support
- All the session chairs, authors of papers and management summaries and finally to
- Ray Panko, of the University of Hawaii, whose work and web site reassured the founders of EuSpRIG that we weren’t alone in worrying about spreadsheets, and that our worries were justified

Please - enjoy the Symposium, spread the word, share the knowledge.

Ray Butler, Chairman, EuSpRIG

June 2000

---


EuSpRIG First Symposium on

Spreadsheet Risks, Audit and Development Methods

SYMPOSIUM ORGANISER
David Chadwick (University of Greenwich and EuSpRIG Treasurer)

SYMPOSIUM CHAIRPERSONS
Professor Brian Knight (University of Greenwich and Symposium President)
Raymond Butler (HM Customs and Excise and EuSpRIG Chairman)
Patrick Cleary (University of Wales Institute Cardiff and EuSpRIG Secretary)

SYMPOSIUM MANAGEMENT COMMITTEE
Robert Li (KPMG Management Consulting)
Graham MacDonald (HM Customs & Excise)
Grenville Croll (Operis)

SYMPOSIUM SESSION CHAIRPERSONS
Raymond Panko (University of Hawaii)
Barry Pettifor (Price Waterhouse Coopers)
Leon Strous (De Nederlandsche Bank, Amsterdam)

SPONSORING BODY
Information Systems Audit and Control Association (Northern UK Chapter)

SUPPORTING BODIES
Institute of Chartered Accountants in England and Wales
British Computer Society Computer Audit Specialist Group
British Computer Society Software Quality Specialist Interest Group
# TABLE OF CONTENTS

## PREFACE

Stop the Subversive Spreadsheet!
*David Chadwick (EuSpRIG Symposium Organiser)*

## INTRODUCTORY SESSION

**Session 1: Extent of the Problem: Types of Errors, Risks**
Chairperson: Raymond Panko, Professor, Decision Sciences Dept, University of Hawaii

Spreadsheet Errors: What we Know. What we Think we Know.
*Raymond Panko (University of Hawaii)*

How Important Are Spreadsheets To Organisations?
*Patrick Cleary (University of Wales Institute Cardiff)*

Classification of Spreadsheet Errors
*Kamalesen Rajalingham, David Chadwick, Brian Knight (University of Greenwich)*

Building Financial Accuracy Into Spreadsheets
*Andrew Hawker (University of Birmingham)*

## Session 2: Development Methodologies and Techniques

Chairperson: Barry Pettifor, Director of Spreadsheet Assurance Services, Price Waterhouse Coopers

A Structured methodology For Spreadsheet Modelling
*Brian Knight, David Chadwick, Kamalesen Rajalingham (University of Greenwich)*

Detecting Errors In Spreadsheets
*Yirsaw Ayalew, Markus Clermont, Roland Mittermeir (Universitat Klagenfurt, Austria)*

## Session 3: Audit Methodologies and Techniques

Chairperson: Leon Strous, Internal Audit Dept, De Nederlandsche Bank, Amsterdam

Risk Assessment For Spreadsheet Developments
*Raymond Butler (HM Customs and Excise)*

Visual Checking Of Spreadsheets
*Hock Chuan Chan, Ying Chen (University of Singapore)*

## Session 4: Software Tools: practical demonstrations

Chairperson: Raymond Butler
*Computer Audit Unit, H.M. Customs & Excise (North West Collection)*
Stop The Subversive Spreadsheet!

David Chadwick
EuSpRIG Symposium Organiser

The Nature of Subversion

"The presence of a spreadsheet application in an accounting system can subvert all the controls in all other parts of that system". So says Ray Butler of the Computer Audit Unit, HM Customs and Excise and Ray should know. For ten years he has been investigating errors in spreadsheets used by companies for calculating their VAT payments. Over this period, he has collected useful data on types and frequencies of errors as well as on the effectiveness of different audit methods not only those used by VAT officers but also those in use by other auditors. It seems surprising that Ray would find such a problem in such a straightforward well-defined business application but as he is quick to point out "Even in a domain such as indirect taxation, which is characterised by relatively simple calculations, relatively high domain knowledge by developers, and generally well-documented calculation rules, the use of spreadsheet applications is fraught with danger and errors"

Ray is not alone in his interest of spreadsheet risks. Chris Conlong of the Business Modelling Group at KPMG Consulting is also only too aware of the problems and, when asked, frequently refers to the findings of a KPMG survey of financial models based on spreadsheets. The survey 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 and 75% had significant accounting errors.

These figures are truly astounding and if extrapolated to all major organisations throughout the world hint at potential disaster scenarios just waiting to happen. A colleague recently remarked "Spreadsheet errors are a business time-bomb waiting to go off. Like the millennium bug - nobody knew the time-bomb was there and then everybody knew and knew when it would happen. With the spreadsheet problem few know that there is a time-bomb at all and nobody knows when their particular bomb may go off"

The extent of the problem has not escaped the notice of academics. Ray Panko at the University of Hawaii has been collecting data on spreadsheet errors for more than ten years. Ray has a formidable collection of research data, papers and research contacts which together form possibly the most extensive and comprehensive resource on the problem in the entire world. In Europe, too, David Chadwick and Brian Knight, at the University of Greenwich, have been collecting errors. Their team has developed a taxonomy of spreadsheet errors which has been used to classify and catalogue practical error situations in a database as a resource for better training and for research into audit and development methodologies. Patrick Cleary, at the University of Wales, Cardiff, has also recently begun researching the extent to which businesses depend upon their spreadsheet data.
What Are The Causes?
But just what are the causes of all these spreadsheet errors?

David Finch, Head of Internal Audit at Superdrug plc, believes "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 and not sexy'. He is quite frank about what he believes is the cause of the phenomenon "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. Users who are technically capable of developing applications have not been trained in development methodology"

David's last statement about training is echoed by Barry Pettifor, Director of Spreadsheet Assurance Services at Price Waterhouse Coopers. Over the past three years Barry and his team have tracked a continuous trend towards greater model complexity, as measured by the number of unique formulae, with a corresponding increase in model size. Barry says "It seems that, given increasingly powerful tools, clients wish to model at greater levels of detail and are constrained by the availability of time and data. However, even though we are talking about major transactions with £millions or even £billions at stake we nearly always find that the modellers have no formal training in good modelling techniques, and that their organisations do not have even the most rudimentary internal modelling standards"

Chris Conlong again refers to the KPMG survey for an explanation of the causes of such errors. The survey gives much the same explanation as Barry at Price Waterhouse Coopers "Whilst these results are disturbing" says the survey " when the modelling procedures are analysed they are of no real surprise. 78% of models had no formal QA (quality assurance) procedure. This is despite 81% of model users believing that their model would provide them with competitive advantage. The ease of use of spreadsheets has led to them being used in virtually all companies, often for analysing key decisions, but the rigour that is traditionally imposed on IT development is usually not applied to spreadsheet models"

So What Is Being Done?
This brings us to the obvious question: what is being done about the situation? or more to the point, is anything being done at all? A typical City manager probably feels that there is no urgency to the problem - after all, companies have been using spreadsheets for twenty years and nobody has gone out of business yet because of them. Or have they?

But there are those attempting to defuse the hidden bombs.

Ray Butler believes that the use of a more formalised development and testing methodology for these applications would help reduce the risk of error especially when combined with supporting computer-assisted audit software. To this end Ray has been working on a software tool to aid VAT officers. The tool is already in use.

David Finch, of Superdrug, believes that much of the problem was due to users not being sufficiently trained in a development methodology and this belief is supported by much of the work done by Brian Knight at the University of Greenwich. For three years Brian has been pointing out to postgraduate students and businessmen alike that in his opinion "There is a
distinct lack of a well-founded software engineering methodology for developing spreadsheets. This continues to hold back attempts to improve quality in large and complex models. But Brian and his team have already come up with interesting methods which are currently being trialled with students at the university. He says "The sooner we can try out our methods on real-world users with real-world problems the sooner we can come up with a methodology of use to all"

EuSpRIG
A significant move to defuse the spreadsheet time-bomb was the meeting of Ray Butler, Patrick Cleary and David Chadwick who together decided to create a forum in which all interested parties could meet and exchange views. And so, the European Spreadsheet Risks Interest Group, EuSpRIG, was founded with the aim of increasing business awareness of spreadsheet errors as well as promoting research on and dissemination of methods of prevention and detection. EuSpRIG has brought together researchers and professionals in the areas of accounting, business, audit and software engineering to actively seek useful solutions.

Adapted from an article by David Chadwick and published in 'Internal Auditing and Business Risk', Institute of Internal Auditors (UK and Ireland) May 2000.
Session 1

Extent of the Problem: Types of Errors, Risks

Chairperson: Raymond Panko,
Professor, Decision Sciences Dept,
University of Hawaii

Papers in this section:

Spreadsheet Errors: What we Know. What we Think we Know.

How Important Are Spreadsheets To Organisations?

Classification of Spreadsheet Errors

Building Financial Accuracy Into Spreadsheets

"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 and not sexy"

David Finch in "Stop The Subversive Spreadsheet"
Proceedings of the Spreadsheet Risk Symposium  
European Spreadsheet Risks Interest Group (EuSpRIG)  
Greenwich, England  
July 17-18, 2000

Spreadsheet Errors:  
What We Know.  
What We Think We Can Do.

Dr. Raymond R. Panko  
University of Hawaii  
panko@hawaii.edu, http://panko.cba.hawaii.edu

Abstract

Fifteen years of research studies have concluded unanimously that spreadsheet errors are both common and non-trivial. Now we must seek ways to reduce spreadsheet errors. Several approaches have been suggested, some of which are promising and others, while appealing because they are easy to do, are not likely to be effective. To date, only one technique, cell-by-cell code inspection, has been demonstrated to be effective. We need to conduct further research to determine the degree to which other techniques can reduce spreadsheet errors.

Introduction

Spreadsheets are widely used in organizations [McLean, Kappelman, & Thompson, 1993]. Each year, tens of millions of managers and professionals around the world create hundreds of millions of spreadsheets. Although many spreadsheets are small and simple throwaway calculations, surveys have shown that many spreadsheets are quite large [Cale 1994, Cragg & King 1993, Floyd, Walls, & Marr 1995, Hall 1996]. Cragg and King [1993] audited spreadsheets as large as 10,000 cells, and when Floyd, Walls, and Marr [1995] conducted a survey of 72 end user developers in four firms, asking subjects to select a single model, the average model had 6,000 cells.

Spreadsheets are also complex, using a large number of sophisticated functions [Hall, 1996].

Spreadsheets are also important. For instance, Gable, Yap, and Eng [1991] examined all 402 non-trivial spreadsheets in one organization. Forty-six percent were rated as important or very important to the organization, and 59% of the spreadsheets were used at least monthly. In another study, Chan and Storey [1996] surveyed 256 spreadsheet developers. Each developer was asked to describe one of their spreadsheets. When asked to identify the highest-level user of the spreadsheet’s data, 13% cited a vice president, and 42% cited their chief executive officer.

Under these circumstances, if many spreadsheets contain errors, the consequences could be dire. Unfortunately, errors in bottom-line values are very likely because spreadsheet modeling is incredibly unforgiving of errors. A spelling error in a word processing document will only occasionally create a material problem; but an error almost anywhere in a spreadsheet will produce an incorrect bottom-line value. Unless the development error rate is close to one error in every ten thousand cells, most large spreadsheets are likely to contain errors.

In fact, we know that humans are incapable of doing complex cognitive tasks with great accuracy. The Human Error website [2000a] lists data from a number of studies of human cognitive errors. These studies indicate that even for simple cognitive tasks, such as flipping switches, error rates are about one in 200. For more complex cognitive tasks, such as writing lines of computer code, error rates are about one in 50 to one in 20.

Research on human error in many fields has shown that the problem is not sloppiness but rather fundamental limitations in human cognition [Reason 1990]. Quite simply, we do not think the way we think we think. Human cognition is built on complex mechanisms that inherently sacrifice some accuracy for speed. Speed in thinking was the overriding evolutionary force for our hunter ancestors, and although occasional errors caused the loss of hunters, new hunters were inexpensive and fun to make. Unfortunately, error rates acceptable in hunting and even in most computer applications cannot be tolerated in spreadsheets. For spreadsheet accuracy, we must somehow overcome or at least
reduce human cognitive accuracy limitations dramatically.

**What We Know**

Let us begin with what we actually know about spreadsheet errors and corporate practices to control spreadsheet errors. More information about the studies listed in this section is available at the Spreadsheet Research website [Panko 2000b].

**Spreadsheets Contain Errors**

The Introduction noted that human error rates in complex cognitive tasks tend to be about 2% to 5%, so spreadsheet errors must be fairly frequent or we will have to throw away decades of cognitive research.

**Data from Field Studies**

In fact, spreadsheet error rates actually *are* rather high. Table 1 shows data from seven field audits of real organizational spreadsheets. The field audits found errors in 24% of the 367 spreadsheets audited, and most older audits used audit techniques not likely to catch a majority of errors. The most recent field audits, in contrast, generally used better methodologies and found errors in at least 86% of the spreadsheets audited. These numbers are even more impressive when you consider that most audits only reported substantive errors, not all errors. In other words, given data from recent field audits, *most* large spreadsheets probably do contain significant errors.

**Table 1: Field Audits of Real Spreadsheets**

<table>
<thead>
<tr>
<th>Study</th>
<th>Year</th>
<th>Spreadsheets</th>
<th>% w Errors</th>
<th>Cell Error Rate (CER)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Davies &amp; Ikin (1)</td>
<td>1987</td>
<td>19</td>
<td>21%</td>
<td>NR</td>
</tr>
<tr>
<td>Butler (2)</td>
<td>1992</td>
<td>273</td>
<td>11%</td>
<td>NR</td>
</tr>
<tr>
<td>Hicks</td>
<td>1995</td>
<td>1</td>
<td>100%</td>
<td>1.2%</td>
</tr>
<tr>
<td>Coopers &amp; Lybrand (3)</td>
<td>1997</td>
<td>23</td>
<td>91%</td>
<td>NR</td>
</tr>
<tr>
<td>KPMG (4)</td>
<td>1997</td>
<td>22</td>
<td>91%</td>
<td>NR</td>
</tr>
<tr>
<td>Lukasik</td>
<td>1998</td>
<td>2</td>
<td>100%</td>
<td>2.2%, 2.5%</td>
</tr>
<tr>
<td>Butler (2)</td>
<td>2000</td>
<td>7</td>
<td>86%</td>
<td>0.4%</td>
</tr>
<tr>
<td>Overall NA</td>
<td>367</td>
<td></td>
<td>24%</td>
<td>--</td>
</tr>
<tr>
<td>1997 and Later NA</td>
<td>54</td>
<td></td>
<td>91%</td>
<td>--</td>
</tr>
</tbody>
</table>

NR = Not reported  
(1) "Serious errors"  
(2) Only reported errors large enough to demand additional tax payments  
(3) Spreadsheets off by at least 5%  
(4) "Major errors"  
(5) Weighted average

Hicks [1995] and Lukasik [1998] audited three large spreadsheets using methodologies that probably caught a large majority of errors. They found that the cell error rates, that is, the percentages of cells containing original errors (as opposed to errors based on earlier erroneous numbers in the spreadsheet) were 1.1%, 2.2%, and 2.5%. These cell error rates (CERs) are about what we would expect from general error research, as discussed above. The Butler [2000] data, in contrast, found a lower CER of only 0.38% for formula cells. This lower result may be the result of careful creation and simple accounting calculations, although the methodology used attempted to balance error detection rates and costs and used a technique that probably could not catch a majority of errors. Even the lower Butler [2000] values mean that even in spreadsheets of a few dozen cells, errors are likely.

**Data from Laboratory Experiments**

As shown in Table 1, we also have data from laboratory experiments that collectively had about a thousand subjects develop spreadsheets from word problems. These word problems had known solutions, allowing 100% error detection. Across these experiments, 51% of all spreadsheets contained errors, despite the fact that most spreadsheets were only 25 to 50 cells in total size. Cell error rates for whole spreadsheet were at least 1% to 2%. These lowest CERs, furthermore, were for a task purposely designed to very simple and largely free of domain knowledge.

**Table 2: Spreadsheet Development Experiments**

<table>
<thead>
<tr>
<th>Study</th>
<th>Year</th>
<th>Sample</th>
<th>Subjects</th>
<th>Spreadsheets</th>
<th>% w Errors</th>
<th>Cell Error</th>
</tr>
</thead>
</table>

2005-01-19 14:43
### What We Know; What We Think

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

These studies, by the way, 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. Overall, then, intensive research has shown that spreadsheet error rates are comparable to those in other human cognitive activities. These error rates are large enough to tell us that most large spreadsheets will contain errors.

### Errors are Like Multiple Poisons

When Panko and Halverson [1997] examined errors made by their subjects, they concluded that spreadsheet errors fell into different categories. They noted that even if all errors of a certain type could be eliminated, the remaining errors would still be fatal. They compared spreadsheets errors to multiple poisons, each of which is 100% lethal.

#### Quantitative and Qualitative Errors

First, there are quantitative errors and qualitative errors. A quantitative error produces and incorrect value in at least one bottom-line variable. In turn, qualitative errors, such as poor design, do not create immediate quantitative errors, but they may cause later problems in data entry or from incorrect modifications.

#### Mechanical, Omission, and Logic Errors

Panko and Halverson [1997] classified quantitative errors into three basic types.

- **Mechanical errors** are simple slips, such as mistyping a number or pointing to the wrong cell when entering a formula. Mechanical errors are common in all experiments that looked at them, but the most obvious mechanical error—mistyping a number—actually has been quite rare.

- **Logic errors** occur when the developer has the wrong algorithm for a particular formula cell or expresses the algorithm incorrectly in the formula.

- **Omission errors** occur when the developer omits something that should be in the model. In experiments, this meant leaving out something in the word problem that should be in the model. In real life, it means not creating
a complete model because not all factors have been considered. Non-spreadsheet research shows that humans are not good at considering all factors when considering a problem [Fischhoff, Slovic & Lichtenstein 1978].

**Errors by Life Cycle**

Like any other system, a spreadsheet has a life cycle, which may include needs analysis, design, construction, testing, and ongoing use. Errors may be introduced (and detected) at any stage in this life cycle.

Note that error generation does not end with the creation of the final spreadsheet. Errors may also occur during ongoing use after development. Users may enter erroneous data into the spreadsheet, of course. In addition, we will see later one particular type of post-development error, hardwiring that is distressingly common yet is also easy to prevent.

**Developers are Overconfident and Policies are Lax**

Given the large amount of data on spreadsheet errors, one might think that companies would implement strong policies for spreadsheet development and testing. However, that rarely is not the case.

Most studies that have audited spreadsheets or surveyed users have reported poor development practices [Cragg & King 1993, Davies & Ikin 1987, Hall 1996, Schultheis & Sumner 1994]. In addition, studies that looked at corporate controls also found a general pattern of little control and of the controls that did exist being largely informal [Cale 1994, Cragg & King 1993, Davies & Ikin 1987, Floyd, Walls, & Marr 1995, Galletta & Hufnagel 1992, Hall 1996, Speier & Brown 1996].

One reason for this lack of disciplined development practices and policies may be that spreadsheet developers are overconfident in the accuracy of their spreadsheets. Certainly laboratory studies, field audits, and surveys have shown a high degree of confidence in the accuracy of spreadsheets, even if quite a few errors were found later [Brown & Gould 1987, Davies & Ikin 1987, Floyd, Walls, & Marr 1995, Panko & Halverson 1997, Panko & Featherman 1999].

In one experiment [Panko & Featherman 1999], for instance, developers were 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. Although these subjects were students, overconfidence has also been found among experts in many fields [Johnson 1988, Shanteau 1992]. Indeed, overconfidence is one of the most consistent findings in behavioral sciences [Plous 1993] and has been linked to a lack of care in risk avoidance [Rasmussen 1990, Rumar 1990].

**What We Think We Can Do**

Now that the widespread existence of errors in spreadsheets is well documented, the next logical question is, “What can we do to reduce errors?” Notice that the word is “reduce” rather than “eliminate.” Years of human error research have shown that there is simply no way to eliminate error completely.

**The Tenacity of Error**

Human error research indicates that human error is tenacious because people are not terribly good at detecting and correcting errors. The Human Error website [Panko 2000a] shows that error detection and correction rates approaching 90% only occur in the simplest processes, such as proofreading for spelling errors in which the misspelling is not itself a valid word. If the result of the spelling error is itself a valid word, error detection rates fall to about 70%. Even this is high compared to error detection and correction for logical errors in mathematics, which in Allwood’s [1984] classic study succeeded in only about half of all errors. Error detection and correction for omission errors is much lower still [Allwood 1984, Bagnara, et al. 1987, Woods 1984].

More directly, the Human Error website [Panko 2000a] has data from a number of software team code inspection studies, in which a group of programmers goes over a program one line at a time to look for errors. These intensive code inspections only find around 80% of all errors despite the use of team code inspection by programming professionals.

Even more directly, there have been experiments in spreadsheet code inspection, in which subjects examine a spreadsheet cell-by-cell to look for errors. These studies indicate that subjects working alone only catch about half of all errors [Galletta et al. 1993 1997, Panko 1999], even when the subjects are experienced spreadsheet developers [Galletta et al. 1993 1997]. We also know that error detection rates in spreadsheet code inspections only approach 90% for mechanical typing and pointing errors in short formulas [Panko 1999]. For logic errors, mechanical errors in long formulas, and omission errors, detection rates in spreadsheet code inspection are far lower [Panko 1999].

Software developers, who are highly experienced with errors, respond to the difficulty of detecting errors by
engaging in massive amounts of formal testing. About a third of the total software development effort goes into formal testing [Grady 1994], and even after several stages of testing, errors remain in about 0.1% to 0.3% of all lines of code [Putnam & Myers 1992].

Given the tenacity of error in the face of intensive code inspection and other types of testing, we should not be very sanguine about any technique of error reduction that falls short of massive testing.

**Cell Protection**

However, there appears to be one simple thing we can do to prevent at least one type of error. Earlier, we noted that one type of error during ongoing use, hardwiring, is very common. In hardwiring, a user cursors to a formula cell and enters a number in the cell. This generally occurs because the user did not realize that the cell was a formula cell and thought they should enter the value. One survey of an Australian mining company’s spreadsheets found hardwiring errors in about 30% of the spreadsheets [Dent 1995].

When a spreadsheet is hardwired, it is likely to be correct for the current user, provided the correct current value is entered into the formula cell. However, if the spreadsheet is saved and then run again, the spreadsheet will not be correct for subsequent users.

Hardwiring errors are rather easy to prevent using cell protection. Cell protection only allows users to change pre-specified input cells, so that if a user attempts to hardwire a formula cell, he or she will be prevented from doing so. Unfortunately, although cell protection is fairly effective and easy to do, it is only done in half or fewer of all spreadsheets [Cragg and King 1993, Davies & Ikin 1987, Hall 1996].

**Re-Keying Data**

There is also a simple way to reduce data input errors. This is to have two input sections, so that all data will be entered twice. It is rather trivial to highlight differences between two blocks of input data in order to highlight errors. This method, used in traditional data processing, is called verification. Data entry errors are likely to be random, so making the same error twice is unlikely. It is easy to check if two input areas are the same and, if not, to determine where the error lies.

**Examining Results for Reasonableness**

Ineffective home remedies in the United States are called “whiskey cures,” after the old aphorism, “Of all the things that do not cure the common cold, whiskey is the most popular.” With a whiskey cure, you take a step to reduce harm, but the step is largely or entirely ineffective.

One commonly seen activity in spreadsheet development is looking over a spreadsheet’s results for reasonableness [Hendry & Green 1994, Nardi & Miller 1991]. If a few errors are found, the seeker feels that he or she is very effective at finding errors. If no errors are found, then the seeker feels that the spreadsheet is free of error.

Given the difficulty of finding errors noted above even when full cell-by-cell code inspection is used, merely looking over a spreadsheet’s numbers for reasonableness must be considered a whiskey cure. In addition, studies [Klein 1997, Rickets 1990] have shown that people are not very good at finding errors when they assess numbers for correctness. Although looking over results for reasonableness is simple and inexpensive and should be done, it must not be considered an acceptable stopping point.

**Good Development Practices?**

A number of authors have proposed good spreadsheet development practices [e.g., Kreie 1988, Rajalingham, et al. 2000, Ronen, Palley, & Lucas, 1989], often modeled after good software development practices. For instance, they usually advocate a full systems development life cycle, with definite needs analysis and design stages, both of which often are skipped by spreadsheet developers [Brown & Gould 1987, Hall 1996]. They usually also propose modular design and the placement of all input numbers in a single “data” or “assumptions” section. Finally, a few propose something like clean room development, in which equations are worked out ahead of time [Rajalingham, et al. 2000], although they usually stop short of formal proofs.

One general problem with these methodologies is that they have not been tested experimentally to see if they really reduce errors and, if so, by how much. In the terminology of the U.S. Federal Drug Administration, they have not been proven “safe and effective.” Experiments conducted to reduce errors [e.g., Janvrin & Morrison 1996, Kreie 1988] have found it very difficult to create statistically significant reductions in error rates. In an unpublished study by the author, when spreadsheets using assumptions sections and not using assumptions sections were compared, there was almost no difference in error rates. Indeed, traditional “good practice” actually may be harmful. If all input data are placed in one section and logic in another, the logic may be more difficult to read for code inspection.
Overall, although good practice in development probably is commendable, it must not be considered sufficient unless it is proven safe and effective through experimentation. Certainly, we should not listen to claims of effectiveness in this area until methods have been tested.

**Error Discovery Software**

Another promising but untested area is error discovery software designed to identify errors in spreadsheets. Error discovery software comes in two basic forms. One form graphically portrays patterns of connections between cells within spreadsheets, say by coloring or arrows, to highlight patterns that may indicate possible errors in the spreadsheet. However, it is not clear how many errors seen in experiments and field audits today would be made more visible by this type of software [Panko, 2000c]. Again, there is a need for testing before claiming that such tools safe and effective.

The other type of error discovery software acts like a spell checker or grammar checker, highlighting specific cells that may contain errors. Again, however, we would like to know the extent to which such software could actually catch the types of errors found in experiments and field audits.

Another issue is customization to specific purposes. The Butler [2000] software, for instance, has several error-detection techniques tailored to the ways in which people may be deliberately cheating on taxes, for instance making a number in a column a label and then right-justifying it, to reduce the column total.

**Code Inspection**

In software development, about a third of all development time is spent in formal testing, as noted earlier. Testing is considered the sine qua non for reducing errors in software development. Even when good practice is used throughout the needs assessment, design, and code development process, there is no substitute for testing.

In spreadsheet development, testing is fairly rare [Cragg & King 1993, Gable, Yap & Eng 1991; Hall 1996, Nardi 1993, Schultheis & Sumner 1994]. Informal comments often cite the cost of testing as being prohibitive. However, without formal testing, the cost of errors must be considered.

There are two general forms of testing. One is execution testing, in which known data are used to test the spreadsheet. However, known data are not always available because spreadsheets programs often allow far more complex analyses than a company could previously perform. In addition, execution testing is a complex craft in which out-of-bounds and extreme values must be tested, and in which test cases generally must be selected very carefully. Yet even when execution testing is done with spreadsheets, it typically avoids such niceties [Hall 1996].

The other form of testing is code inspection, in which an individual or group goes through a spreadsheet cell by cell to look for errors. Code inspection is very expensive, because software development has taught us that the yield (percentage of errors found) in software testing falls when code inspection is done rapidly [Panko 2000a]. In addition, code inspection is exhausting work, and inspectors tend to hate it. Finally, as noted above, individual code inspectors find only half or fewer of all errors, so group code inspection is necessary. Overall, group code inspection is an unpleasant medicine yet the only medicine that has been experimentally tested to date and found to be fairly effective, catching about 80% of all errors [Panko 2000a].

At the same time, there can be cost-yield tradeoffs. For instance, Butler [1992 2000] was concerned with finding incorrect spreadsheets in tax auditing. Each error found might result in income from additional taxation. However, auditing is also expensive. Given tradeoffs between yield and cost, it was reasonable to use single-person code inspection augmented by software, and was reasonable not to put limits on inspection rates. In general, the Butler [2000] analysis method represents an interesting approach for matching costs with yields.

However, it is important for firms to deliberately consider how to do code inspection. If accuracy is important, single-person code inspection appears to be a poor idea. Teams of three or more are likely to be necessary.

**Conclusion**

Research on spreadsheet errors began over fifteen years ago. During that time, there has been ample evidence demonstrating that spreadsheet errors are common and nontrivial. Quite simply, spreadsheet error rates are comparable to error rates in other human cognitive activities and are caused by fundamental limitations in human cognition, not mere sloppiness. Nor does ordinary “being careful” eliminate errors or reduce them to acceptable levels.

Now that the reality of spreadsheet error has been established, the next step is to ask what we can do to reduce spreadsheet errors. Unfortunately, only one approach to error reduction has been demonstrated to be effective. This is code inspection, in which a group of spreadsheet developers checks a spreadsheet cell-by-cell to discover errors. Even this exhausting and expensive process will catch only about 80% of all errors. Other things can be done to reduce
errors, but given the tenacity of spreadsheet error in the face of even cell-by-cell code inspection, we should be careful about expecting too much from most error-reducing approach. What is needed now is an experimental research program to determine which approaches really are safe and effective.

References


Butler, Raymond, Tax audit study in 1992, personal communications by electronic mail, August and September, 1996 and in August, 1997.


Dent, A., personal communication with the first author via electronic mail, April 2, 1995.


Hassinen, K., University of Joensuu, private communication with the first author, January 1995. Provided further data from his 1988 study, so that the cell error rate could be computed.

Hassinen, K. *An Experimental Study of Spreadsheet Errors Made by Novice Spreadsheet Users*, Department of Computer Science, University of Joensuu, P. O. Box 111, SF-80101 Joensuu, Finland, 1988.

Hendry, D. G. & Green, T. R. G. “Creating, Comprehending, and Explaining Spreadsheets: A Cognitive Interpretation

Hicks, L., NYNEX, personal communication with the first author via electronic mail, June 21, 1995.


Lukasik, Todd, CPS, private communication with the author, August 10, 1998.


HOW IMPORTANT ARE SPREADSHEETS TO ORGANISATIONS?

P. M. Cleary
Business School
University of Wales Institute Cardiff
Email: PMCleary@uwic.ac.uk

ABSTRACT
The upsurge in the development and use of spreadsheet models over the last ten years has prompted research into the risks of errors in the models. In response to research papers on this subject, reviewers have commented that ‘spreadsheets are not really vital to organisations and that they are only used for small analysis’ (Panko, 1999). This proposed project will attempt to establish the degree of importance of spreadsheets to organisations.

AIM
To quantify the importance of spreadsheets in organisational decision-making.

OBJECTIVES
1. To carry out a pilot study in one organisation to establish a suitable metric for measuring the importance of spreadsheets in organisational decision-making.
2. To carry out a study in a relevant number of organisations using the established metric to determine the importance of spreadsheets in organisational decision-making.

PROPOSED METHODOLOGY
Three approaches have been suggested by Panko (1999):
1. Critical Incident - in which managers are asked to select a recent key decision and are then queried as to where they acquired the information that supported the selected decision.
2. Managerial Tools - in which managers are surveyed to ascertain what tools use to guide their work
3. Spreadsheet Census - in which questions are asked to ascertain the highest managerial level at which each of the organisation’s spreadsheets are used.
Each of these approaches should result in a quantitative measure which can be related to the importance of spreadsheets to the organisation.

It is unclear which, if any, of the above approaches would be feasible and useful. It is therefore proposed that a pilot study be carried out in one organisation to test the appropriateness of each approach. A refined approach can then be used in a wider study to test the longer term hypothesis.

Each approach will now be considered in turn.
1. CRITICAL INCIDENT
An open-ended interview technique should enable a view to be obtained as to whether or not spreadsheet models contributed to the decision-making process. A Likert-type scale can be used to establish, in the opinion of the manager, the size of the contribution made. This metric, combined with some metric of the hierarchical level of the manager, may be considered as representing the importance of spreadsheets to the organisation. Some discussion needs to take place regarding the number of critical incidents each individual should be asked to examine, and a measure of the importance of that decision to the company may need to be established.

2. MANAGERIAL TOOLS
A survey of the number of tools used by each manager should enable a metric to be established that will represent:
- the number of managers at each hierarchical level that use spreadsheets
- the use of spreadsheets as a percentage of the number of tools used.
This metric should be relatively easy to obtain by interview.

3. SPREADSHEET CENSUS
Establishing the highest hierarchical level at which each individual spreadsheet is used should provide a useful metric and should be relatively easy to obtain by interview.

It could be that a metric can be established which is dependent on all three approaches.

4. PROVISIONAL TIMETABLE
- Literature search - 4 weeks
- Design Likert-type and other appropriate scales taking guidance from the literature - 1 week
- Design interview questionnaires - 1 week
- Identify personnel within the target organisation who partake in the decision-making process - 1 week
- Identify all spreadsheets used within the organisation - 1 week
- Carry out critical incident and managerial tools interviews with decision-makers - 3 weeks
- Carry out spreadsheet census to establish the highest decision-making hierarchical level at which each spreadsheet is used - 2 weeks
- Analyse the data gathered - 2 weeks
- Write report - 4 weeks

Total - 19 weeks

5. BACKGROUND
A brief critique of two papers published on spreadsheet research will illustrate the problem. Floyd, Walls et al (1995), in their paper Managing Spreadsheet Models, discuss the risks associated with End-User Computing and link this discussion to risks associated with spreadsheet development. They describe how they collected and analysed data on spreadsheet development practices in organisations and follow this with a discussion on the management methods observed and management models developed by others. They conclude with some recommendations of how spreadsheet development can be managed to
reduce errors. However, at no time do they attempt to quantify the importance of
spreadsheets in organisational decision-making.

In another paper, Panko (1998) discusses at great length 'What We Know About
Spreadsheet Errors'. He points out that 'Over the years a number of embarrassing
spreadsheet development incidents have been reported', and refers to a number of
spreadsheet error audits that have established that spreadsheets in use today are very likely
to contain errors. Yet again, there is no attempt to quantify the importance of these error-
prone spreadsheets to organisations. Clearly, if spreadsheets are not important to
organisations, then there is little point researchers spending valuable time investigating the
problem and attempting to develop methods for reducing error rates.

An investigation which will quantify the importance of spreadsheets to organisations
therefore seems appropriate and useful to the spreadsheet community.

6. POTENTIAL COLLABORATIVE ORGANISATIONS
There are a number of organisations that have offered to collaborate in this pilot study:
- Gilesports Ltd
- University Hospital Of Wales, Cardiff
- Hyder plc

Gilesports Ltd is a distributor and retailer of sports goods. It runs a warehouse and
distribution centre in Cardiff and has a chain of 70+ retail outlets across the country. A
traditional and until recently family-run business, it is expanding rapidly and is adopting
new technology in order to enable it to continue to compete.

The University Hospital of Wales is the largest UK National Health Service Trust in
Wales and is a large district general hospital. In recent conversations with the author,
senior managers in the Trust have expressed concern about major public funding decisions
being made by the Welsh Assembly based on spreadsheet information supplied by the
Trust. As a result, they welcome the opportunity to collaborate with UWIC in this
research.

Hyder plc is the largest plc in Wales. Owners of Welsh Water and South Wales Electricity
Company, they also have a civil engineering division.

7. REFERENCES

Journal of Systems Management, 46, pp 38-43:68

Panko, R. (1998). What We Know About Spreadsheet Errors. Accepted for Journal of End
User Computing

ABSTRACT

This paper describes a framework for a systematic classification of spreadsheet errors. This classification or taxonomy of errors is aimed at facilitating analysis and comprehension of the different types of spreadsheet errors. It is far more comprehensive than any presented or published before. The taxonomy is an outcome of a thorough investigation of the widespread problem of spreadsheet errors and an analysis of specific types of these errors. This paper also contains a clear description of the various elements and categories of the classification. It is also accompanied and supported by appropriate examples.

1. INTRODUCTION

A host of publications over two decades have clearly described the seriousness of spreadsheet errors and their adverse or potential impact on businesses. A financial model review by KPMG Management Consulting, London confirms the frequency and seriousness of spreadsheet errors. Their report states that in 95% of the financial models reviewed, at least 5 errors were found. The review also reveals alarming statistics pertaining to defects in spreadsheet development, addressing the project management, technical and analysis aspects.

An article in New Scientist has reported that a study conducted by the British accounting firm Coopers & Lybrand found errors in 90% of the spreadsheets audited. This is an extremely high figure and if the errors went undetected, it could have had a devastating effect on the business. It is evident from these cases that the occurrence of spreadsheet errors is a major problem for businesses and needs to be addressed urgently.

A thorough review of literature relevant to spreadsheet development and errors reveals that very little research has been done on studying specific errors that occur in spreadsheets. The outcome of a thorough analysis of specific types of spreadsheet errors from a wide variety of sources is a more comprehensive classification or taxonomy of spreadsheet errors than ever presented before. It reflects an improvement to the version of the classification presented previously by the authors.

2. THE SPREADSHEET ERROR TAXONOMY

2.1 Introduction

In a broad sense, taxonomy is the science of classification, though more strictly, it refers to the classification of living and extinct organisms. The term is derived from the Greek taxis ("arrangement") and nomos ("law"). It is important to note, however, that there is no special theory which lies behind modern taxonomic methods.

In attempting to define taxonomy within the context of spreadsheet errors, it would be appropriate to investigate the definition of this term in other fields of study. In biology, taxonomy refers to the establishment of a hierarchical system of categories on the basis of presumed natural relationships among organisms. The goal of classifying is to place an organism into an already existing group or to create a new group for it, based on its resemblances to and differences from known forms. To this end, a hierarchy of categories is recognised.

2.2 Taxonomy of Spreadsheet Errors

Based on the definitions borrowed from other disciplines, we can extend the concept of taxonomy to include the classification of spreadsheet errors. For our purposes, the spreadsheet error taxonomy can be described as the hierarchical system of categories of spreadsheet errors on the basis of presumed common characteristics and relationships.

Based on the principles of classification adopted in botany and zoology, taxonomic methods for spreadsheet errors depend on:

a) Obtaining a specific type and example of a spreadsheet error
b) Comparing the error with the known range of variation of spreadsheet errors
c) Correctly identifying the error if it has been described, or preparing a description
   Showing similarities to and differences from known categories, or, if the error is of a
   new type, assigning it to a new category.
d) Determining the best position for the error in the existing classifications and determining what
   revision the classification may require as a consequence of the new discovery

3. RATIONALE FOR DEVELOPING A SPREADSHEET ERROR TAXONOMY

There are various reasons for developing a taxonomy of spreadsheet errors. The most important
probably is that it forces us to clearly understand the characteristics of an error as well as the nature
of its occurrence. A comparison can also be made with other related errors belong to the same
category or level.

An insight into the features and nature of an error is critical for any effort to devise a solution or
method of detection. In general, a similar approach can be taken to address errors within the same
category of the classification. The knowledge of the characteristics of an error also enables analysis
of its potential impact and frequency. It is also highly probable that other errors in the same
category would have the same degree of seriousness.

4. FRAMEWORK FOR THE CLASSIFICATION OF SPREADSHEET ERRORS

Two different approaches to the classification of spreadsheet errors were experimented. The
following frameworks for the classification of spreadsheet errors appear feasible based on an
examination of the process of spreadsheet development and the characteristics of spreadsheet
errors and the nature of their occurrence:

i) Based on the nature and characteristics of the error
ii) Based on the spreadsheet development life cycle

Having used both frameworks, it was found that the classification based on the characteristics of
the error was far more appropriate due to its structure and rigidity. The main criterion for selecting
the better framework was the possibility of minimising the recurrence of the same category or type
of error in different parts of the taxonomy. In other words, to minimise the overlap of different
categories of spreadsheet errors.

In order to produce the taxonomy of spreadsheet errors, the binary tree approach is used in
conjunction with the analysis of spreadsheet errors based on their nature and characteristics. At
each stage of the taxonomy, this approach uses dichotomies or divisions into two disjunctive
groups, to classify spreadsheet errors.

5. CLASSIFICATION OF SPREADSHEET ERRORS

Figure 1 shows the model of the classification of spreadsheet errors constructed by adopting the
framework described in the previous section.

SYSTEM-GENERATED

System-generated errors are errors made by the spreadsheet software or bugs in the software. Their
occurrence is generally beyond the control of users, although they can, when aware, take corrective
action.

Example: Century Error

In MS Excel 97 for instance, for any entry of a date (without the century) before 01/01/30, the
century is assumed to be the 21st century while for any entry of a date (without the century) after
01/01/30, the century is assumed to be the 20th century. This problem of course, can be avoided if
the year is explicitly entered with the century e.g. 09/02/1915, 03/12/2060 (dd/mm/yy).

USER-GENERATED

User-generated errors are errors committed by the user, as opposed to being software/system-generated and can be prevented, detected and corrected by the user. They can be divided into two major categories at the highest level, namely qualitative errors and quantitative errors.
QUANTITATIVE

Quantitative errors are numerical errors that lead to incorrect bottom-line values.6

SPREADSHEET ERRORS

- SYSTEM-GENERATED
- USER-GENERATED
  - QUANTITATIVE
    - ACCIDENTAL
      - DEVELOPER (workings)
        - Omission
        - Alteration
        - Duplication
      - END-USER
        - DATA INPUTTER (Input)
          - Omission
          - Alteration
          - Duplication
        - INTERPRETER (output)
          - Omission
          - Alteration
          - Duplication
      - REASONING
        - DOMAIN KNOWLEDGE
        - REAL-WORLD KNOWLEDGE
        - MATHEMATICAL REPRESENTATION
      - IMPLEMENTATION
        - SYNTAX
        - LOGIC
  - QUALITATIVE
    - SEMANTIC
    - STRUCTURAL
    - TEMPORAL
    - MAINTAINABILITY

Figure 1. Taxonomy of Spreadsheet Errors

ACCIDENTAL

Accidental errors are mistakes and slips caused by negligence, such as typing errors. Though quite frequently occurring, they have a high chance of being spotted and corrected immediately by the person committing the error. Some, however, do go undetected and could lead to incorrect values in other cells. It is important to state here that most of the errors described under this category can also be intentional or deliberately caused with malicious intent.

After a close examination of various types of accidental errors, it has been found that they can be further divided into two distinct categories. They are developer-committed errors and end-user-committed errors.

DEVELOPER-COMMITTED ERRORS

Developer-committed errors are errors produced by the developer of the spreadsheet model. These errors usually occur in the workings (as opposed to input or output) section of the model. They can belong to any of three categories, namely omission, alteration and deletion.

OMISSION

Here, omissions are things accidentally left out of the model by the developer. Human factors research has shown that on-fission errors are especially dangerous, because they have low detection rates.6 It could be that a key factor or variable is omitted from the spreadsheet model and therefore, an important relationship is missing from the model.
Example: References to corresponding input data in the workings/output section are omitted from the model.

KPMG, in one of their client models, found that increase in vehicle cost was blank until 2001, even though the source of data from that date (from another worksheet) contained values for the earlier years.  

ALTERATION  
This error occurs when the developer of the model accidentally makes a change to an existing model, that produces a defect in the model. An example of such an error is the use of cell protection on the wrong cells accidentally, making it impossible for users to enter data.

DUPLICATION  
The developer of the model accidentally re-creates elements of the model, causing data duplication or redundancy.  

Example: A variable is defined twice.  
When developing a model, it's easy to make a forecast for a growth rate of X%. X is written into the equations that compute growth but is written in as a constant, for example, =cell above x 1.04. In a later stage of model development, the user might do a what-if analysis and writes an equation such as = [cell above] x [growth rate cell]. During debugging, the two growth rates might be identical or similar. During use, they might be different.

END-USER-COMMITTED ERRORS  
End-user-committed errors are mistakes or slips made by end-users that merely manipulate or interpret the spreadsheet model/system. The end-users can consist of two distinct groups, namely the data inputters and the data interpreters.

DATA INPUTTER  
The data inputter is the end-user who enters the data required by the model. It is these values which are fed into the workings and output sections. The data inputter may also produce errors as a result of omission, alteration or duplication of data.

OMISSION  
These errors are typically caused by the data inputter who fails to enter a piece of data required by the spreadsheet model.

ALTERATION  
These errors usually take the form of data input or overwriting errors. These are errors made by users while adding to or modifying existing data in the spreadsheet model.  

Example: Rows are added to spreadsheets but not the "bottom line" totals.  
The modeller has written an equation to find column totals, writing the equation in row seven. Data are to be entered below. The equation is written =SUM (B8:B99). It works fine until a user adds data in row 100. Because this row is beyond the range of the equation, the data is not included in the addition.

DUPLICATION  
Duplication errors by data inputters are mainly caused by accidentally re-entering data in the wrong part of the spreadsheet.

DATA INTERPRETER  
The data interpreter is the end-user who extracts useful information from the model and presents it in a more convenient form. This is the output section of the spreadsheet model. The data interpreter may perform various actions to obtain the desired information. In the process, they may commit errors that can be classed as either omission, alteration or duplication based.
OMISSION

The data interpreter accidentally leaves out certain elements from the output section of the model.

ALTERATION

The data interpreter may incorrectly alter the model and consequently misinterpret the results. For instance, they may sort particular columns of data in a table, accidentally leaving out the corresponding columns. This makes the table inconsistent and unreliable.

ERRORS IN REASONING

These errors involve entering the wrong formula because of a mistake in reasoning. The formulae may be incorrect as a result of either choosing the wrong algorithm or creating the wrong formulae to implement the algorithm.

DOMAIN KNOWLEDGE

Domain knowledge errors are produced due to lack of knowledge required to analyse the business function in order to design the data model which is to be represented electronically by the spreadsheet model. These skills enable the user to identify business functions which are suitable for modelling with a spreadsheet and how this modelling is to be done. This requires thorough knowledge of business functionality and requirements for both the present and the future.

REAL-WORLD KNOWLEDGE

These errors involve creating an incorrect formula by selecting the wrong algorithm.

Example: Calculation of depreciation
The reducing balance method is used instead of the straight line method or vice versa.

Example: Absence of distinction between leap and non-leap years
For instance, year 2000 is a leap year, but calculations divide by 365 not 366.8.

MATHEMATICAL REPRESENTATION

These errors involve incorrect or inaccurate construction of a formula to implement a correctly chosen algorithm.

Example: The PERCENTAGE problem
This error occurs when the formula to calculate percentage is incorrectly written, either due to lack of knowledge of what a percentage is or BODMAS (Brackets, Of, Division, Multiplication, Addition, Subtraction) by which the spreadsheet identifies precedence in calculations e.g. B2/A2*100, B2*100/A2 or B2*A2/100 instead of A2/B2*100 or A2*100/B2. This is based on figure 2 below.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Night Wages £</td>
<td>Total Wages £</td>
<td>Night Wages %</td>
</tr>
<tr>
<td>1400.00</td>
<td>4690.00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2

IMPLEMENTATION

Implementation errors are produced due to lack of knowledge on the full use of the functions and capabilities of the particular spreadsheet package in use, with an understanding of the spreadsheet principles, concepts, constructs, reserved words and syntax. Implementation errors can be divided into syntax and logic errors.

SYNTAX ERRORS

A syntax error occurs when the formula contains characters and symbols which are not recognised by the spreadsheet software to perform the desired function. Syntax errors can be easily detected as the spreadsheet immediately indicates that an error has occurred.
LOGIC ERRORS

A logic error is a form of implementation error which occurs when the formula is incorrectly constructed due to a lack of understanding of the features and functions of the spreadsheet software in use. As a result, the formula produces a wrong value.

Example: Relative and absolute copy problem

The relative copy causes cell references in a copied formula to alter row and column references relative to the original cell copied. People often make the false assumption that the software will automatically adapt the cell references wherever they happen to copy '0.

Example: Misconception of the AVERAGE function

Users see the word 'Average' in the column heading and immediately apply the average function without questioning whether it was appropriate. Based on figure 3, Over 80% of students in a survey entered =AVERAGE(C6:D6) in cell F6. But this gives the average of Basic Wages and Overtime Wages when, given the context, surely it is the 'average wage per person' and the formula should be =E6/B6.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Lazy Days Staff Budget Costs 1995-1996</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Staff Basic Overtime Total Average</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Numbers Wages £ Wages £ Wages £ Wage £</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Managers</td>
<td>1 17700 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Grade 1</td>
<td>3 45540 1400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Grade 2</td>
<td>9 122340 2000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Grade 3</td>
<td>12 102350 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Grand Totals</td>
<td>25 287930 3400</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3

Example: Circular references

This error frequently occurs in totals where the formula uses its own value in its calculation. This error will give a run-time error message and so probably occurs infrequently. A common example of a circular reference arises when calculating bank overdraft interest, and can be corrected as follows:

With a circular reference, i.e., the incorrect way:

<table>
<thead>
<tr>
<th>Cashflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>£</td>
</tr>
</tbody>
</table>

Opening bank balance (overdrawn) (x)
Add: Receipts x
Less: Payments (x)
Less: Overdraft interest based on closing balance (x)
Closing bank balance (x)

Figure 4a

Each time the spreadsheet is recalculated the overdraft interest will change and update the closing bank balance ad infinitum. Without a circular reference, i.e., the correct way:

<table>
<thead>
<tr>
<th>Cashflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>£</td>
</tr>
</tbody>
</table>

Opening bank balance (overdrawn) (x)
Add: Receipts x
Less: Payments (x)
Balance before overdraft interest (x)
Less: Overdraft interest on balance before interest (x)
Closing bank balance (x)

Figure 4b
QUALITATIVE

**Qualitative errors** are errors that do not immediately produce incorrect numeric values but degrade the quality of the model. The model also becomes more prone to misinterpretation on the part of the user. As a result, it also becomes more difficult to update and maintain the model. A more detailed investigation into qualitative errors reveals that they can be generally divided into two different types, namely, **semantic** and **maintainability errors**.

**SEMANTIC ERRORS**

Semantic errors are qualitative errors that occur due to a distortion of or ambiguity in the meaning of data. It consequently leads to incorrect decisions, choices or assumptions. As far as qualitative errors are concerned, semantic errors are relatively very difficult to detect. They can be divided into **structural** and **temporal** errors.

**STRUCTURAL ERRORS**

These errors usually take the form of flaws in the design or layout of the model, incorrect or ambiguous headings, and situations in which the documented assumptions are not reflected in the model, causing confusion.

**Example:** Formatting error

If you format to one digit to the right of the decimal (F1), and then enter values having greater precision, the spreadsheet will round off the numbers. Thus 1.44 will round off to 1.4; the sum of 1.44 and 1.44 will round to 2.9 from 2.88. Such additions will appear incorrect.

**Exam**

**ple:** **SUM Incorrect Use Problem**

A common error is to enter any formula within the SUM brackets as though the SUM was mandatory for defining a formula, for instance, in the spreadsheet model in figure x, the formula in cell H7 might be wrongly entered as =SUM(G7/D7) when it should really be =G7/D7. Although the calculation is correctly done, this is logically wrong and could cause confusion.

**TEMPORAL ERRORS**

Temporal errors are described as qualitative errors produced due to the use of data which has not been updated. They can lead to unreliable decisions or interpretation of the situation.

**Example:** **Qualitative error resulting from the referencing of non-current Data**

This is an example of a qualitative error produced as a result of referencing a piece of data that has become invalid due to time lapse. In the example given below (figure 5), this piece of data is the exchange rate from British Pounds (£) to Ringgit Malaysia (RM) contained in cell F2. If the exchange rate undergoes acute fluctuations and the changes are not reflected in cell F2, the calculation in cell A8 produces a value that is invalid. This is a qualitative error and any decision made based on this value would be unreliable.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tea (£)</td>
<td>Milk (£)</td>
<td>Coffee (£)</td>
<td>Exchange Rate (£ to RM)</td>
<td>1</td>
</tr>
<tr>
<td>1st Quarter</td>
<td>450</td>
<td>560</td>
<td>467</td>
<td>7.3</td>
<td>2</td>
</tr>
<tr>
<td>2nd Quarter</td>
<td>904</td>
<td>900</td>
<td>352</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>3rd Quarter</td>
<td>872</td>
<td>800</td>
<td>233</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>4th Quarter</td>
<td>123</td>
<td>234</td>
<td>901</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>Total Sale of Tea &amp; Coffee (RM)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>=SUM(B2:B5,D2:D5)*F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

**Figure 5**
MAINTAINABILITY

Maintainability flaws are features of the spreadsheet model that make it difficult to be updated or modified. They can potentially cause inconsistency in the model. A common and typical example of a maintainability error is hard-coding.

Example:  

*Hard-coding*

The hard-coding of a formula is another example of a qualitative, decision error. This error decreases the quality of the spreadsheet by making it much less flexible. Referring to figure 6, if the formulae in column H were hard-coded e.g. \( \text{=G8}/9 \) (in cell H8) instead of \( \text{=G8}/\text{D8} \), and if any of the values in column D (number of staff) changed, the formula in column H of the same row would have to be re-written. This is just a simple example to illustrate the concept of hard-coding being a source of error.

### Staff Budget Costs 1995-1996

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Number of Staff</td>
<td>Day Wages £</td>
<td>Night Wages £</td>
<td>Total Wages £</td>
<td>Average Wage £</td>
</tr>
<tr>
<td>Grade 1</td>
<td>1</td>
<td>17700.50</td>
<td>0.00</td>
<td>=SUM(E6:F6)</td>
<td>=G6/D6</td>
</tr>
<tr>
<td>Grade 2</td>
<td>3</td>
<td>45540.00</td>
<td>1400.55</td>
<td>=SUM(E7:F7)</td>
<td>=G7/D7</td>
</tr>
<tr>
<td>Grade 3</td>
<td>9</td>
<td>122340.00</td>
<td>2000.00</td>
<td>=SUM(E8:F8)</td>
<td>=G8/D8</td>
</tr>
<tr>
<td>Grade 4</td>
<td>12</td>
<td>102350.25</td>
<td>0.00</td>
<td>=SUM(E9:F9)</td>
<td>=G9/D9</td>
</tr>
<tr>
<td>Grand Total</td>
<td>=SUM(D6:D9)</td>
<td>=SUM(E6:E9)</td>
<td>=SUM(F6:F9)</td>
<td>=SUM(G6:G9)</td>
<td>=G10/D10</td>
</tr>
</tbody>
</table>

**Figure 6**

It should also be noted that some numbers, which at first sight appear to be constants, are often in fact variables. For example, the rate of inflation or the percentage value for employees' pension contributions.

6. CONCLUSION

The classification of spreadsheet errors has been found to be very useful in analysing specific types of spreadsheet errors. It also enables users to gain a better understanding of the different types of errors that can occur in their spreadsheet models. Appropriate tools, techniques and methods can subsequently be developed to prevent their occurrence in the first place or enhance the chances of detecting these errors after they have occurred. In addition to that, when a new specific type of error is identified, it can be placed in the appropriate category within the taxonomy. In the process of classifying the error, spreadsheet developers and end-users are bound to gain a much deeper understanding of the error. This is because they will be forced to examine and compare its characteristics with those of other spreadsheet errors.
REFERENCES


(3)  Encyclopaedia Britannica Online


BUILDING FINANCIAL ACCURACY INTO SPREADSHEETS

Andrew Hawker
Department of Accounting & Finance, University of Birmingham, UK
Email: A.Hawker@bham.ac.uk. Fax: 0121414 6678.

ABSTRACT

Students learning how to apply spreadsheets to accounting problems are not always well served by the built-in financial functions. Problems can arise because of differences between UK and US practice, through anomalies in the functions themselves, and because the promptings of Wizards' engender an attitude of filling in the blanks on the screen, and hoping for the best. Some examples of these problems are described, and suggestions are presented for ways of improving the situation. Principally, it is suggested that spreadsheet prompts and 'Help' screens should offer integrated guidance, covering some aspects of financial practice, as well as matters of spreadsheet technique.

1. INTRODUCTION

When spreadsheeting first came into the accounting curriculum, students spent a lot of time learning to deal with some awkward and temperamental features of the software. Everyone looked forward to the day when it would become more user-friendly. Perhaps this was a little naive. There is no doubt that, with today's products and the dominance of Microsoft's style, students can get to grips with spreadsheets much faster than they used to. Unfortunately, this can also give them a false sense of confidence, as they discover and activate all the powerful built-in functions.

Over the past decade or so, considerable evidence has accumulated of the dangers of poorly-constructed spreadsheets, thanks to the efforts of Brown, Panko, Janvrin, and others. A number of useful guides to good practice and audit are now available. Most of these publications suggest ways in which those who create spreadsheets should be more systematic and vigilant. This clearly has to be the first line of attack in promoting good practice. However, there are still some occasions when the products themselves could help to pre-empt errors.

This paper is concerned with one particular aspect of spread-sheeting (the use of financial functions) in one particular niche in the education system (accounting courses in a university). If students are going to move on to use the financial functions in a business environment, it is important that they know what they are doing. Their coursework includes practice in the use of functions for calculating present values, interest rates, depreciation, and so on. A recurring problem is that the technology works smoothly enough, but the functions do not behave exactly as might be expected.
The comments which follow are based on experiences with Microsoft Excel, up to and including Excel 97. This is not intended to suggest that these problems are by any means confined to Microsoft's product.

2. THE PROBLEM AREAS

The problem areas can be grouped under three headings:

2.1. Unexpected results

A number of arguments have raged about arithmetic in spreadsheets when large numbers of significant figures are involved, but this is rarely an issue for accountants. Sometimes, however, the accuracy turns out to be less than might be expected. For example, the declining balance function in Excel calculates and uses a factor which is accurate to three decimal places. This means that if you are dealing with an asset worth millions of pounds, the final few figures of each depreciation figure will be suspect.

Depreciation is not a very exact art, and this is fine so long as the user converts the results to round sums. Furthermore, it is unlikely that those doing such calculations manually in the past ever bothered to work out the factor beyond a few decimal places. Spreadsheets, though, are expected to deliver precision. Anyone who adds up depreciation values over the life of the asset, and expects that the salvage value will always equate to the initial value minus the accumulated depreciation, could be in for a shock. (A quick way of verifying this is to add together the values provided in the example given on the Help screen for the DB function). They could be further perplexed if they used the optional "month" argument in this function. This makes rather primitive adjustments for an asset bought part way through the first accounting year. Users of this argument have to bear in mind that (a) it only makes sense if you are working in periods of one year, (b) you need to add an extra period at the end (not mentioned in the Help text) and (c) your results are even less likely to balance than when you leave out this optional argument altogether. All this is fairly difficult for a novice student to take on board.

2.2. Misleading or ineffective "help"

Efforts to make Help facilities more “Intelligent” have not always been well received. Unsolicited interventions, particularly by cartoon paper-clips, can be extremely irritating. However, there are many areas of Help text which remain unchanged from early Excel releases. Where changes have been made, they often have the feel of amendments to a software specification, rather than real attempts to guide and inform the behaviour of users. For example, an extra option has been added to the ACCRINT and related functions, to allow users to choose a "European 30/360" method of calculation. In Europe and the US, the same basic formula is used to calculate interest on a "30/360" basis, but there are some potentially crucial differences in the way months are treated if they are not actually 30 days long. Originally, the Excel function only offered the US method, as its default.
This is an example of a situation where it would be very useful if users could, if they wished, be offered more explanation of the financial as well as the technical aspects of the functions they are using. They could be advised, for example, on the types of investments for which each option is appropriate, and the key differences between the different methods of calculation. Such explanations are to be found on the web (for example at bondchannel.bridge.com). In theory, we could build some appropriate hyperlinks into the spreadsheet. However, since many of the issues are basic and unchanging, it would be far better if they were incorporated into the routine Help material, alongside the normal guidance on technique. When actually using the spreadsheet it is, after all, necessary to consider all these different aspects of the problem at the same time.

Interest rates can cause other problems, again resulting mainly from the Atlantic divide. Help screens routinely advise that annual interest rates can be converted to monthly ones simply by dividing by 12. This advice is not tempered by any reference to effective versus nominal interest rates. It accords with much American practice, but causes problems if, for example, students are invited to replicate a loan repayment table issued by a UK bank. This is because UK practice is to quote effective annual rates (based on compounded monthly interest). Divide the annual rate by 12, and the results you obtain will be too high.

Excel has the effective and nominal functions to help get round these problems, but makes no effort to steer users towards them at the critical moment. Furthermore, recent supplementary advice from Microsoft on using different compounding periods completely ignores this as a potential pitfall.

This leads off into some other interesting territory, since UK lenders are permitted, by law, to quote annual rates rounded down to a single decimal point (so that many of them routinely set rates such as 11.995%, which can legitimately be advertised as 11.9%). It is quite a good test of students' audit skills to have them replicate the calculations, based on the advertised figures, and pick up on anomalies of this kind. In one particular loan scheme, for example, customers were advised that they could defer paying back the loan during a repayment "holiday". What the advertising omitted to mention was that interest on the full amount of the advance was charged during the "holiday" period, and was then added to the capital. Again, it is a useful exercise in scrutiny to ask students to try and explain the figures, as published, for the scheme. For such exercises to succeed, it is important that the students should be given consistent and appropriate advice on interest calculations throughout.

2.3. Elephant traps

Much of the emphasis in modern interactive design is on enabling users to work quickly by making choices, and entering data via structured input panels. This is fine, for users who know exactly what they are doing. For those who do not (i.e. by definition, most students) there are some dangers. The arrival of the Insert Function facility, for example, has led to various forms of 'function surfing". Students presume that there is a function for everything. It is generally easier to hunt around for a function to try, than to bring up the Help screen, (or even to keep your fingers off the mouse, and think it through for yourself). Having found what seems to be a suitable candidate, students will start to stuff all kinds of data into the panels of
the wizard, in a determined (and usually increasingly bad-tempered) effort to persuade it to deliver the outcome they want.

There are two versions of the elephant trap. Firstly, the function may yield up a result which looks fine, but which is actually wrong. For example, INTRATE looks like a useful contender for finding an interest rate, given initial and final values for an investment, but only works for simple interest. Alternatively, the function may indeed be the right one, but if inappropriate data is fed into it, the rather restricted feedback may not alert the user to this. Indeed, it is all too easy to assume that the wizard is giving its approval simply by accepting the input. Unfortunately, quite simple errors may be overlooked. For example, it may be that 01/01/80 is being calculated as .125, rather than being interpreted as a date, or a percentage value is being taken at a hundred times the value intended. Of course students make exactly the same mistakes when entering values into cells. Their shift in perception is that the wizard is somehow all-knowing. They expect it to pick up errors for them.

3. IMPLICATIONS

All the instances mentioned above can be dismissed as being relatively trivial. But in financial calculations, quite minor misjudgements or misapprehensions on the part of the user can cause havoc in the final results. Consideration of these minor flaws also prompts a more worrying question. Even if we wish to correct some of the defects in existing functions, is it actually feasible to do so?

At this point, some sympathy is due to the software developers. Take the familiar Net Present Value function, for example. Ever since its inception (by Lotus, in this case) it has been a trap for the unwary. A textbook published more than ten years ago advises: It is important to note that the Lotus @NPV routine applies discounting to all the cells in the specified range. This convention was followed by the designers of Excel. Any initial investment, which does not need discounting because it is already at today's values, should be omitted.

This does not seem at all intuitive to students, as can be seen by the number of them who take an entire series of cash flows, and feed them all into the NPV function. (They could of course use the more recent XNPV function, which discounts from the second payment onwards. However, this requires the user to enter both cash figures and a matching set of dates. Apart from the complexity of the function, they may also be puzzled by an enigmatic statement in the Help screen, to the effect that the first payment is optional).

It would be nice to re-invent the NPV function to be more intuitive, but this is a QWERTY-type problem. Experienced users expect NPV to work in a particular way, and it is buried in the calculations of countless existing spreadsheets. So we would already seem to be locked in to a particular way of doing things.

Similar issues arise in respect of conventions which vary between countries or between industries. A "UK Edition" of Excel might ensure that all calculations complied with UK practice. The Help screens could provide exact guidance on when to use particular ways of calculating interest or depreciation. However, such spreadsheets would not travel well.
Someone loading the spreadsheet into a US Edition would immediately have to be alerted to its original "nationality".

A final issue to be confronted is one which arises in a great deal of accounting software. Practices which have evolved in paper-based systems do not always make sense in electronic ones. Thus many accounting packages advertise themselves as "double entry", even though some of the error checking inherent in double entry methods is no longer done or required. Similarly, the survival of calculations based on 30 and 360 days, rather than true elapsed time, can no longer be defended on grounds of practicability. As Bhandari points out, what now seems to happen very often is that computer power is used to hoodwink the consumer, by using 360 as a divisor where it helps to top up the revenue of the loan issuer\(^8\). If combined with variable rates\(^9,10\), the interest charges can become extremely difficult to audit. Logic suggests that everyone should use an identical "true" approach to interest periods, but this is an area where spreadsheets can only reflect, not lead, the real world.

4. THE WAY AHEAD

In the meantime, there are two strands of policy which perhaps should be considered. In education, it would help if more recognition were given to the necessity of spending time with students, discussing and explaining the nature of the tools they are using. This can only pay dividends in the way they apply them in their subsequent careers. Simply because spreadsheet technique can easily be learned from books and interactive tutorials, it does not follow that good practice follows close behind. At a time when Higher Education is constantly being urged to be more "efficient", it should be noted that cutting back on staff contact time, and relying substantially on automated teaching, will probably be a false economy in the longer term.

Secondly, it would help to have more constructive discussions between the designers of specialised spreadsheet facilities, such as the built-in financial functions, and those who set the standards in the relevant industry sectors. This implies that suppliers should look beyond merely checking that functions work exactly "as per specification" or adding in extra ones if there seems to be demand. We should be urging them to integrate the technical and the financial advice being given to users. This would mean calling on the expertise of outsiders such as accounting's professional bodies, and possibly developing different "dialects" for Help and user documentation for different countries. This would be a practical acknowledgement that errors are likely to follow not just from failures in technique, but from failures of alignment between the spreadsheet functions, and the conventions and perceptions of the world they serve.

REFERENCES


Session 2

Development Methodologies and Techniques

Chairperson: Barry Pettifor
Director of Spreadsheet Assurance Services, Pricewaterhouse Coopers

Papers in this section:

A Structured Methodology For Spreadsheet Modelling
Detecting Errors In Spreadsheets

"...we nearly always find that the modellers have no formal training in good modelling techniques, and that their organisations do not have even the most rudimentary internal modelling standards"

Barry Pettifor in "Stop The Subversive Spreadsheet"
A Structured Methodology
For Spreadsheet Modelling

Brian Knight, David Chadwick, Kamalesen Rajalingham
University of Greenwich, Information Integrity Research Centre,
School of Computing and Mathematics, 30 Park Row, London SE10 9LS UK
{b.knight, d.r.chadwick, k.rajalingham} @greenwich.ac.uk,

ABSTRACT

In this paper, we discuss the problem of the software engineering of a class of business spreadsheet models. A methodology for structured software development is proposed, which is based on structured analysis of data, represented as Jackson diagrams. 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. The capability of the methodology to provide a modular overview in the model is described, and examples are given. The potential for a reverse-engineering tool, to transform existing spreadsheet models is discussed.

1. INTRODUCTION

This paper describes an outcome from research done by the authors at the Information Integrity Research Centre at Greenwich over the past 3 years, concerning the problems of the quality of spreadsheet models. The research has focused on the class of business models, including functional formulae, referencing and replication of individual cells and ranges. Surveys have shown that the frequency and severity of errors in spreadsheets is now reaching dangerous proportions. A KPMG survey of financial models based on spreadsheets 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 and 75% had significant accounting errors.

There is much evidence that these errors are caused by untrained or badly trained modellers and, that even those who are technically capable of developing applications have not been trained in any development methodology Development is in many ways comparable to the days for main-line software development before the advances due to structured programming and design.

The approach of this research has been to examine the applicability of main-line software-engineering techniques to the very special needs of spreadsheet developers. These needs are partly determined by the visual nature of spreadsheets and their heavy reliance on referencing and intermediate data, and partly by the likely acceptance of techniques within the industry. However sound a methodology is, we cannot expect modellers to undergo much training in software engineering. Object orientation may be technically ideal, but not if modellers have to learn the Unified Modelling Language first.

The aim of the research was to create a methodology for spreadsheets which improved the quality of models, whilst not imposing an extra burden of modellers. To this end, we have looked for a support tool to assist in spreadsheet structuring. Ideally, the tool should be able to take existing models, and transform them to the appropriate form.
Several structured programming and design methodologies originated during the 60s and 70s, with goals to systematise the process of analysis and design of software. The goals were to increase productivity, reduce errors, ease problems of maintenance, and where possible to automate the development process. Amongst these, several important "data-oriented" methods were proposed, amongst which were the Warnier-Orr\textsuperscript{1,2} methodology, M.A. Jackson's JSD\textsuperscript{3} and Chen's E-R data modelling\textsuperscript{4}. These methodologies concentrate primarily on the logical structure of the data, which is likely to be more stable than the software functions. It is argued that this provides a good basis for comprehensible software, which is able to support change and maintenance over time.

In this research, the suitability of a methodology based on Jackson charts 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 are may be summarised as:

- 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.

In section 2 of this paper, we show explain the methodology with illustrations. In section 3 we explain the modularisation principle and the relation to Jackson charts. In section 4 the possibility of automatic re-engineering of existing spreadsheets is discussed.

### 2. APPLICATION OF JACKSON CHARTS TO A SINGLE MODULE

The essence of JSD is the structure diagram and its relationship to block structure, with its three key constructs of sequence, repetition and selection. Figure 1 shows a structure diagram, representing a typical block structured module. Here asterisked blocks are repeated, and blocks marked with an O are selections (mutually exclusive). The diagram shows that A consists of a repeated block B, and each B is made up of either C or D. C is a sequence of block E followed by block F.

The indented structure on the right of figure 1 is the structured programming equivalent of the structure diagram. The philosophy of structured programming, as outlined in \textsuperscript{3} promotes the indented form for code. This form has led to huge improvements in the comprehension of code, leading to improvements in productivity, auditing and maintenance. Later work \textsuperscript{6} proposed methods for the translation of data structure into structured form. Jackson proposed that the form of the data structure diagram should be extracted from the natural structure existing in the data to be processed.

Some of these techniques can in fact be transferred to the production of spreadsheets, and that this can give a more comprehensible format for spreadsheets, based on indented format. The derivation of the structure charts can be based on the natural data dependencies within the spreadsheet. This is an analytical exercise which depends on a close examination of the semantics of the data involved, to build a logical model in chart form. However, it will be noticed that structure diagrams bear resemblance to the graphs obtained using auditing tools on existing spreadsheets. This reflects the fact that the logical structure is in fact embedded in existing spreadsheets, and may be extracted from them automatically.
We first illustrate how these principles can be used to structure a single spreadsheet, leaving a discussion of module formation to the next section. We take as illustration the example of a profit and loss account and shown in Figure 2 below. From knowledge of the meaning of the data, we may construct the chart shown in Figure 3.

![Figure 1 An example structure diagram](image)

![Figure 2 An example unstructured spreadsheet](image)
To maintain this structure in the spreadsheet view, we can use the indentation principle both on the row labels and on the data values themselves. In fact, we can also insist that data values are indented by assigning a spreadsheet column to each level of indentation. If this is done, the spreadsheet takes on the form shown in Figure 4. Notice 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 two figures: Net Profit add unappropriated profits from last year and Total appropriations. On the data level we see that 24,219 is made up from 36,019 and 11,800. Likewise, we see immediately the constituents of Total expenses are a total of eight different expense types, and the data level. 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 3 shows net profit, unappropriated profits from last year, proposed dividend, general reserve, and foreign exchange. These figures give a valid impression of the state of the trading account at this level of detail. If we were to include a figure from a different level, e.g. purchases (from column 7), it would confuse the picture, since it has already been included in net profit.
3. MODULARISATION OF SPREADSHEETS

Modularisation is the key to successful software engineering, allowing complex systems to be broken down into manageable sub-systems, for ease of comprehension and maintenance. Indeed, the basic principle guiding modularisation can be said to characterise different software engineering methodologies. Object-oriented software engineering is characterised by Parnas's information hiding principle, and Stevens, Constantine and Myers' structured approach is characterised by the concept of code cohesion. In the spreadsheet methodology described here, modules are defined by graphical properties of data structure diagrams.

In section 2, we looked at a structure diagram which took the form of a tree, and showed how this could lead to a structured spreadsheet form for a single module. However, not all spreadsheets are of this simple form, but have structure charts in the form of a more general graph. shows an example of such a chart. The chart is different to that in Figure 1 in that there is a loop in the relationships connecting A B and Q so that we do not any longer have a tree form. In this chart, data block C contributes to block A and to block B. We can of course turn the graph into a tree by duplicating the structure C D E, as shown in figure 5. However, the resulting structured spreadsheet will then have to include the rows

Figure 4 A structured spreadsheet form

Figure 5: Chart in the form of a graph
for C D and E in two different places - as a constituent of A, and as a constituent of B.

The duplication problem can be overcome simply by defining the structure C, D, E, as a separate module, which will occur once in the spreadsheet model. The chart of figure 5 now takes the form of 2 structured modules.

In general, we can always reduce a chart to tree structure by this method, which conveniently produces a unique modularisation of the spreadsheet module, each individual module being expressible in indented form. The modularisation itself, and the relationships between the modules can give a useful overview of complex modules. Figure 6 shows part of an example modular overview for a re-engineering of a normal spreadsheet model. The labels attached to the modules were added after the re-engineering.

Figure 6 Part of a Module level overview of the model

4. DATA INPUT MODULES

Data input represents a special problem in spreadsheet design, with its own special requirements. There are reasons why cells for data input should be grouped together in data input modules, separate from the structured modules described above. One reason is to do with the utmost importance of obtaining accurate data entry. 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. A second reason is that input cells are often referred to by more than one calculated cell. In this case, according to the discussion of the previous section, they should each have the status of a module.

We are however, 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 data input cells. If we do this we end up with the architecture exhibited in figure 7.
The structured spreadsheet modules represent the calculation and display modules. They are the interface accessible with read/write access to the model builder and maintainer, and with read access for the user and auditor. The data entry modules are accessible to the builder, maintainer, auditor with test authorisation, and user with data entry authorisation.

![Architecture of a structured spreadsheet](image)

**Figure 7 Architecture of a structured spreadsheet**

5. CONCLUSIONS

This paper has described progress on a research project to investigate the use of structured techniques in spreadsheets. It has concentrated on an outline of the main theoretical results obtained, and has indicated their possible use in the construction of sound spreadsheet models. The main results are that structured techniques based on Jackson diagrams may be used with advantage to produce well structured spreadsheets. The techniques give rise to a modularisation principle allowing a decomposition of spreadsheets. The paper shows how individual modules can be structured to advantage, and how an overview of module interactions can be visualised.

The paper has presented an outline only, and has not entered into a discussion of related problems, such as recursive dependency relationships, and practical problems of frequent addition and deletions. We intend to publish a discussion of these problems in a follow up article.

Future work on this project is envisaged on two issues. The first is an investigation of the potential of the structured form for improving the quality of spreadsheets software. The second is work towards an automatic re-engineering tool which can extract information on structure from existing spreadsheets, and translate models into structured form.
REFERENCES


10. Chadwick, D., “Stop the Subversive Spreadsheet”, Internal Auditing and Business Risk magazine, Institute of Internal Auditors United Kingdom, May 2000,

DETECTING ERRORS IN SPREADSHEETS

Yirsaw Ayalew, Markus Clermont, Roland T. Mittermeir
Institut für Informatik-Systeme, Universität Klagenfurt
Klagenfurt, Austria
Email: {yirsaw, mark, mittermeir}@ifi.uni-klu.ac.at

ABSTRACT
The paper presents two complementary strategies for identifying errors in spreadsheet programs. The strategies presented are grounded on the assumption that spreadsheets are software, albeit of a different nature than conventional procedural software. Correspondingly, strategies for identifying errors have to take into account the inherent properties of spreadsheets as much as they have to recognize that the conceptual models of “spreadsheet programmers” differ from the conceptual models of conventional programmers. Nevertheless, nobody can and will write a spreadsheet, without having such a conceptual model in mind, be it of numeric nature or be it of geometrical nature focused on some layout.

1 INTRODUCTION
Spreadsheet systems are the most widely used and the most popular end user systems. Hence, spreadsheets (we might refer to them as “spreadsheet programs”) are an important basis for far reaching decisions in almost any field of a modern society. Studies on the quality of spreadsheets resp. spreadsheet based decisions show, however, that there is a substantial divergence between significance and care in this area [3, 6, 12, 13, 14, 15, 16].

Nardi and Miller [10, 11] discussed the characteristics spreadsheet languages provide for end-user programming. Among them is the property that spreadsheets shield users from low-level details of traditional programming. They allow users to think in terms of tabular layouts of adequately arranged and textually designated numbers. Thus, they appear to users as analogous to pencil and paper. The computing model upon which they finally rest is so hidden from the users that the term “programming” seems inappropriate and the term “testing” simply inapplicable.

As professionals we have to recognize, though, that below the surface, spreadsheets are programs. They are even special programs from the perspective that the placement of code is dependent on the layout of the result. – A fact that seems to reduce complexity at first sight (and it does so in simple cases), but that might become a burden in complex situations and specifically during modification. Thus, given the factual importance of spreadsheets due to the importance of the decisions based upon spreadsheet computations, very conventional considerations for software quality need to be considered. These considerations might encompass testing as much as they might encompass design or maintenance and configuration management. However, in using these technical terms, we must not forget that the spreadsheet user does not consider him-/herself as a programmer. (S)he is an end-user who does not want to be bothered with technicalities of the world of programming. In order to become successful, approaches to improve quality control for spreadsheets have to avoid conventional programming- or software engineering jargon. They rather have to link directly to the conceptual structures, spreadsheet users have readily available.
In this paper we will therefore first try to highlight some commonalities and some differences between spreadsheets and conventional algorithmic software. We then give some definitions of the basic terms needed for a focussed discussion about errors in spreadsheets. In section 4, a framework for the classification of spreadsheet faults is described on the basis of some prototypical errors. Finally, in section 5, two complementary approaches to alleviate quality problems in spreadsheet programs are outlined.

2 SPREADSHEETS AND SOFTWARE: WHAT'S DIFFERENT?

Software is written in a professional manner by Professionals; Spreadsheets are written by End-Users! While this statement is true on some face value, it raises wrong connotations. Software professionals, if working professionally, will build their products based on design that is based on some conceptual model or specification linking the application problem to an algorithmic solution with the algorithm usually considering also certain computer idiosyncracies (input/output being not the least among them). Spreadsheet-writers are end-users. As such, they are not programming professionals. However, they are professionals too, professionals in their application domain. In this capacity, they – like anybody else who writes something meaningful – do express themselves based on some conceptual model whenever they express their problems/solutions in writing. Of course, this applies also when they express themselves in writing a spreadsheet. The only difference to the software professional is that their model is not related to programming concepts. It relates application aspects to two-dimensional (tabular) arrangements of numbers interspersed with explanatory text. The numbers are further conceptually interrelated by either one of the following situations:

- The given number is the result of a computation of some other numbers placed (or to be written later) at a given location.

- The given number is part of a set of numbers playing conceptually the same role. This “same role” is generally expressed by geometrical proximity (physical area). However, we will later identify cases where this conceptual embracement cannot be expressed by geometrical proximity (logical area).

Spreadsheet experts will recognize that the two cases mentioned are not comprehensive. However, we claim that they cover most of the territory, at least most of the territory “non-expert” end-users are familiar with.

As spreadsheet systems are easy to use, do not require much training in formal methods of designing and programming, and show – in contrast to conventional programs – the results of the effort while the development effort is still in progress, they are also written in a style different from conventional software. There is a notion of immediate feedback [5] once the contents of a cell is specified. This easy way to quick feedback leads to a development style of trial & error, cutting & pasting, copying & modifying; a mixture that has to be horrifying for an orderly software methodologist.

Figure 1 shows these aspects. Given these considerations, it becomes obvious that irrespective of the true nature of spreadsheet-“software”, conventional wisdom on software testing (c.f. [2, 9, 17]) does either not apply at all or applies only to a limited extent. It applies specifically from the perspective though that spreadsheet computations are basically numerical computations. We will come back to this property in section 5.2.
Hence, rather than banking too much on preaching the gospel of “designing first” and establishing a quality improvement cure on observing this commandment, we base our approach on the very nature of existing spreadsheets and existing processes of how spreadsheets are written. In this paper, we focus specifically on “spreadsheets as they are”. This leads us to discuss in the sequel model visualization and plausibility testing. With model visualization, we bank on end-users transforming their problems/solutions into two-dimensional structures. Visualization will highlight irregularities in this transformation. For plausibility testing we rely on the end-users gut feeling for meaningful boundaries of the data (numbers!) treated in a spreadsheet. Discussing strategies for ensuring spreadsheet quality dynamically (focusing on spreadsheet evolution) would be beyond the scope of this paper.

Before delving into both of these areas, we will proceed by defining some key terms needed in the further discussion and mentioning some prototypical faults in spreadsheets and their related categories.

3 SOME TERMINOLOGY

Since the term “spreadsheet” itself is overloaded, we explain below the semantics attached to spreadsheet related terms in this paper:

A cell is the atomic unit of a spreadsheet and can have five states: (a) it can be empty, (b) it can hold a constant value that is supplied by the programmer of the spreadsheet, (c) it can hold an input value that is supplied by the user of the spreadsheet, (d) it can hold a value that is calculated by a formula, or (e) it can hold a label that describes the contents of a set of other cells.

A formula is a mathematical expression, containing cell references, operators, functions¹, and constant values. At least one cell-reference is expected to be included in the computational expression of the formula. A formula yields exactly one result and is free of side-effects.

¹A function is a built-in formula supplied by the spreadsheet system
A cell reference is a reference to another cell’s value which is either relative or absolute. The address of the referenced cell is given with a pair of coordinates, in the first case the origin is the referencing cell, in the latter the upper left corner of the spreadsheet.

An area is a set of related cells. If the cells are spatially neighbors and the area is marked by the programmer, we refer to it as physical area. A physical area usually serves as the input for a grouping function, like SUM, MAX or AVG. If the relation originates from similarities of the data-manipulation or from the way of creation (i.e copy and paste), we use the term logical area. We require the cells in a physical area to be also spatially adjacent, for cells in a logical area, no such criterion is defined. The logical area is used to describe a kind of conceptual cohesion between cells. If we cannot figure out the way the cells were created (e.g. copy and paste of same source), we have to employ certain heuristics that are based on the similarity of the references and formulas to group cells into logical areas.

A spreadsheet is an n-dimensional matrix of cells. Each cell is uniquely identified by n-coordinates. If n=2, as in the standard case, a cell is uniquely identified by its row and column address.

A Spreadsheet Core Language (SCL) is a set of language constructs to describe the data-flow (cell references) and the data-manipulation (formulas) in the spreadsheet program. Functional properties of the spreadsheet are expressed by SCL. The copy and paste primitives are also considered to be part of the SCL, if they are used in a context with logical areas.

A Spreadsheet Language (SL) is the SCL together with constructs for manipulation of the layout of the spreadsheet.

A Spreadsheet Program (SP) is the specification of data-flow between cells, data-manipulation in cells, and of the values of constant cells.

A Spreadsheet Instance (SI) is a spreadsheet program, where all input cells have certain values. A spreadsheet program can be instantiated multiple times. By changing one of the input values, the spreadsheet instance of a certain spreadsheet program is transformed into another spreadsheet instance of the same program.

A Spreadsheet System is an integrated environment, where spreadsheet programs can be created, instantiated and edited. The spreadsheet system interprets a specific spreadsheet language.

4 FAULTY SPREADSHEETS AND ERROR CATEGORIES

In this section we describe a framework that enables us to categorizes errors by their association to spreadsheet concepts. We define three categories of errors that are associated with physical areas, logical areas, and general errors. Each of the categories will be briefly described and the reasons for the assignment of a certain error to its category will be given. The examples shown try to demonstrate how an error originates. Of course, all the problems shown could have been solved in another way, without an error occurring.

A classification scheme should address the types of most frequent important errors. In addition, the effectiveness of error prevention and detection techniques can be evaluated, pro-
vided that there is a taxonomy of errors which indicates the types, frequency, and possible causes. However, as Beizer [2] indicated, there is no universally correct way to categorize faults. A given fault can be put into different categories depending on the view of the tester and the source of the error. For example, typing “+” instead of “-” in a given formula might be a typographical error or result from misunderstanding the necessary arithmetic.

Some classification schemes are available for spreadsheet errors. Panko and Halverson [15] offer a taxonomy that consists of three major categories of errors: mechanical, logic, and omission errors. Mechanical errors refer to typographical and positioning errors. Logic errors are misunderstandings of the logic of the necessary algorithm to be used in a formula. Omission errors are a result of leaving out something needed in the program. This classification is mainly based on the causes of the errors. A more general classification scheme containing Panko and Halverson’s scheme is given by Rajalingham et al. [20].

In their experimental study, Saariluoma et al. [22] categorized spreadsheet errors in two basic types: location and formula errors. Location errors are what is commonly termed as misreference errors. Saariluoma et al. indicated that these errors are typical in spreadsheet programs. Formula errors contain typographical errors in formula components and what they call mathematical mistakes. Mathematical errors are a result of the inability to define the necessary mathematical expression in a formula. The main errors in this scheme are typo, misreference, and mathematical errors.

Unlike other classification schemes, we do not want to categorize the errors by their cause, but rather by the spreadsheet concept they seem to be associated with. In our further considerations we do not make a difference between logical, mathematical, or typographic errors, because from the error itself we cannot resolve its cause.

4.1 Category 1: Physical Area Related Errors

Errors that are typical to physical areas normally deal with missing values in the area or values of the wrong type somewhere in the area. We call this error reference to a blank cell resp. reference to a cell with a value of wrong type. In some cases such values are entered on purpose, to achieve a better structure and/or readability of the spreadsheet program. In other cases, these values result from errors.

Example 1: Reference to a blank/wrong typed cell

In Figure 2 the range for the sum spans from label 1. Quarter down to the last cell of the list. The two label cells are not considered in the sum yet, but there is no hint for the user/programmer that they might influence the sum, if they are changed to a number (e.g to 1 instead of 1. Quarter).
Another typical problem of the physical area is the impact on the results if new values are added to the area. If a new value is inserted somewhere in the middle of the physical area, it automatically expands, such that the new value and all old values are still within the area. If the new values are added by appending them to the area, the area does not expand. This leads to the error type of incorrect range specification.

Generally, the incorrect range specification problem exists if there are cells outside the physical area that should be part of it. For the user it is not clear that those cells are not part of the physical area anymore and it is common for him/her to assume that those cells influence the result of the function applied to the physical area, too.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Salesman</td>
<td>Date</td>
</tr>
<tr>
<td>2</td>
<td>Miller</td>
<td>014,000</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>044,000</td>
</tr>
<tr>
<td>4</td>
<td>Total</td>
<td>044,000</td>
</tr>
</tbody>
</table>

**Figure 3**: Formation of an incorrect range specification error

**Example 2: Physical Area Specification Error**

In Figure 3 the user defines a sum over an area of cells. During the lifespan of the spreadsheet program it turns out that more cells are needed for specifying the revenues of the salesmen. This is not a problem for extending Miller’s range. But the row appended for Smith is not part of the physical area anymore. The sum-cell C8 does not yield the correct result. However, the reason why the final spreadsheet instance is wrong is not obvious for the user.

A third class of typical errors is the accidental deletion of a cell within a physical area. This leads to the already identified reference to a blank cell error. In addition, adding something that should not be present, will have similar consequences.

A fourth class of errors is the physical area mix up error. While the previous error categories are grounded on the fact that users hardly distinguish between spreadsheet programs and spreadsheet instances (input has not the distinct role as in conventional programming), this error class is due to the spreadsheet program’s property of being a mixture of a problem solving tool and a presentation tool. The problem arises, when two separate physical areas get mixed up. In this case one of them cannot be defined as a physical area by the user anymore. The grouping functions have to be replaced by expressions (i.e SUM by multiple +). For the user it is not obvious that (s)he can specify two physical areas in two columns (see left-hand-side of Figure 4), but that it is not allowed to merge them in one column resp. that the result of the grouping function applied to one of the physical areas is not correct any more.

**Example 3: Physical area mix up problem**

As shown in Figure 4, the salesman spreadsheet program has to calculate a final sum over all sales and a subsum for each salesman. If the user wishes to place the final sum, the subsum and the sales in one column (i.e. for layout reasons), the final sum has to be replaced by an expression which adds the subsums. If the subsum moves to another cell or another salesman (with a new subsum) is introduced, the user has to maintain the final sum expression. If (s)he forgets it, the final sum becomes wrong.
4.2 Category 2: Logical Area Related Errors

As defined in section 3, a logical area represents some kind of cohesion between cells. Normally a logical area originates from copying the same source multiple times and the user is not aware of the logical area, which a cell belongs to.

A typical error is overwriting a formula with a constant value. This error can have many reasons, like rounding errors or unexpected results of the formula. The user simply overwrites the formula result in the cell with a constant value. Of course, this value remains there, even if the values in the formerly referenced cells change.

Another error that is common to logical areas is copy misreference. In this case, a constant value or an absolute reference is specified in a formula, instead of a relative reference. This error is generally not noticed until the cell’s formula is copied into another cell. If a constant cell is referenced with a relative reference, a similar problem will occur when the cell’s formula is copied.

4.3 Category 3: General Errors

General errors are not explicitly associated with a physical or logical area. Few of them are made when entering values into input cells while most of them are made during formula definition. An error associated with input cells is only typographical; e.g., a user might type 75 instead of 750. Incorrect use of formats also affects the way a value is displayed. One might format a value as 0.2 % while the intended meaning could have been 20 %. This can happen to both input cells and formula cells. In addition, if a numeric data is formatted as label data, then it might affect the computed value of a formula.

Another group of general errors is made during formula definition. As stated in section 3 a formula may involve cell references, operators, functions, and constant values. An error can be made in any of these components due to typographical errors or inability to formulate the necessary mathematical expression. These errors include operator errors, boundary errors, parentheses errors, and function errors.

5 QUALITY IMPROVEMENT APPROACHES

Here, two concepts to increase the quality of spreadsheet programs are presented. The approaches deal with the different classes of errors discussed in the above categorization. Prevention and detection of the variety of errors described above requires different methodologies that are currently not available for spreadsheet systems.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Salesman</td>
<td>Date</td>
<td>Sales</td>
<td>Salesman</td>
<td>Date</td>
<td>Sales</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Miller</td>
<td>04.4.2000</td>
<td>500</td>
<td>Miller</td>
<td>01.4.2000</td>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>16.4.2000</td>
<td>1000</td>
<td></td>
<td>16.4.2000</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>18.4.2000</td>
<td>900</td>
<td></td>
<td>18.4.2000</td>
<td>900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Subtotal Miller</td>
<td></td>
<td>2400</td>
<td>Subtotal Miller</td>
<td></td>
<td>2400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Smith</td>
<td>04.4.2000</td>
<td>600</td>
<td>Smith</td>
<td>01.4.2000</td>
<td>600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>06.4.2000</td>
<td>900</td>
<td></td>
<td>06.4.2000</td>
<td>900</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>16.4.2000</td>
<td>1000</td>
<td></td>
<td>16.4.2000</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Subtotal Smith</td>
<td></td>
<td>2500</td>
<td>Subtotal Smith</td>
<td></td>
<td>2500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Total</td>
<td></td>
<td>4900</td>
<td>Total</td>
<td></td>
<td>4900</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4: Physical area mix up problem
We first discuss *model visualization*. This gives the spreadsheet programmer resp. the spreadsheet user more insight into the structure of the spreadsheet, which is expected to help shortening the trial and error process of creating the spreadsheet and to understand and debug spreadsheets in use. The other approach deals with *interval testing* spreadsheets. It tries to overcome the difficulties resulting from the lack of specification of spreadsheets by introducing interval arithmetic as basic device.

### 5.1 Model Visualization

The fact that spreadsheet models\(^2\) are “buried in the formulas” [6] obviously makes it very hard to understand and to reconstruct the spreadsheet model.

The buried model has to be reconstructed, to enable the developer or tester to see beyond the formulas to the underlying logic and structure. To achieve this we must consider both the dataflow in the spreadsheet (as suggested by [21]) and the static aspects, such as logical and physical areas. The generation of such a representation of the spreadsheet model should be automatic with little or no intervention of the programmer. Once generated, the spreadsheet model can be used for visualization and for the automatic comparison of spreadsheet programs.

The visualization should support different resolutions, from coarse to fine grained, to give the user resp. programmer the possibility to have a look at the spreadsheet program on the level of physical and logical areas and the dataflow between those areas. In a further step there should be a possibility for the user to zoom into certain areas and to get a more detailed overview on formula or cell-reference resolution.

We plan to realize the graphical visualization of the model in a way that is based on the data-flow graph of the spreadsheet (see [1,8]), but also visualizing logical and physical areas. The user should be enabled to navigate in the visualized model as suggested in Storey et al. [23]. These authors suggest a representation which allows zooming into specific areas of a graph, without loosing the overview about the context, using a *fisheye view* (see 4).

\(^2\)Abstract representation of a spreadsheet program

![Figure 5: Shortening the trial and error process](image-url)
Our visualized model should serve as a tool for three purposes:

1. Shortening the trial and error process to develop solutions for real-world problems (see Figure 5). We assume that problem understanding is supported by the graphical representation of the spreadsheet model.

2. Understanding of spreadsheet programs that were developed by another programmer.

3. Enabling comparison of spreadsheet programs at the level of the spreadsheet model. This comparison should abstract from values and consider only the model properties, like data-flow, physical and logical areas.

The visualized model will give a representation of physical areas. This gives visual feedback to the user, if there are cells of different types or cells of different conceptual content in the area. A physical or logical area might be visualized as a box, interruptions are indicated using lines of a different color. This visualization should help to control the reference to a cell with a value of wrong type problem.

It has also to be checked if there are adjacent cells to the physical area which have the same type as the cells of the area. This might be a hint for the incorrect physical area specification problem which can be properly visualized by drawing the required border in a different color.

The physical area mix up problem can be resolved by separating the overlapping areas again in the graphical representation. The detection of such overlapping areas, however, is not a trivial problem and further research has to be done on this topic.

By identifying and visualizing logical areas, a concept that is not visually expressed for the user resp. programmer in modern spreadsheet systems, a lot of the problems presented in section 4.2 are already alleviated. Logical areas will often be spatially adjacent, although that is not a necessity. Sporadic interruptions by only a few cells might be a hint for the overwriting a formula with a constant value problem. The way of visualization should be similar to visualizing the reference to a cell with value of wrong type problem.

5.2 Interval Testing

After creating a spreadsheet program for a particular application, it is natural to check its correctness. Spreadsheet programs derive mainly to perform numerical computations. What do people expect to be correct? Usually, one has a gut feeling of the range of reasonable values for each given cell.

Spreadsheet development is based on cells which are to be filled with input values and formulas for computation. For the correctness of a spreadsheet program, every input value as well as every formula should be correct. Actually, many spreadsheet errors are made during formula definition. To judge the validity of the value of a formula cell, we check whether the computation is in the range of expected results. However, the expected behavior of a spreadsheet program is not explicitly specified.

The main duty in testing a program is to detect the existence of a fault in the program. To achieve this one needs systematically designed test cases (using an appropriate test

---

3If there are cells with values of different types in the area, the correct type can be resolved from the grouping function, which is applied to the area.
strategy) that reveal faults in the program. By running the program with the test cases and comparing the result with the expected outcome described in the specification or generated by a test oracle\(^4\), the existence of a fault can be detected.

Generating a powerful oracle, however, presupposes the existence of a specification \([7,18,19]\). Here, we neither have the specification required, nor would spreadsheet developers have the patience and expertise to run a lengthy suite of test cases. Hence, mechanisms need to be devised to approach the power of a test oracle while putting minimal strains on the developers’ diligence and insight into complex dependencies. Thus, we must recognize that “testers” of spreadsheet programs are end-users who are not aware of testing theory and hence they are not expected to do testing in the traditional sense. Rather, users of spreadsheet systems are highly dependent on the system’s assistance. The fact that control structures are confined to cell contents (and in general used rather rarely if compared to algorithmic programs) allows us to use interval arithmetic as proxy for the services of powerful test oracles.

Figure 6 depicts the test process for a spreadsheet program. Based on the goal of computation and by looking at the input values of cells referenced in a formula, the user, assuming the role of a human oracle, specifies the expected range of computation of a formula in the form of an interval for permissible/expected values.

Each actual value assumed by a cell is a discrete value, either entered by the user as input or obtained as result of a computation by the spreadsheet program. For each of these cells, a range of permissible values has to be given. This is much simpler than generating test cases (which is a very complex process especially for end users) as seen in imperative programs.

The user specifies intervals for those input cells which may assume different values. Those cells which do not assume different values can be represented by an interval of length zero. Therefore, for a formula cell under test, there are two values to be computed and compared (c.f. Figure 6): a value \((d)\), computed by the spreadsheet program based on the values of the cells referenced in the formula; a bounding interval \((B)\) computed by interval program based on interval arithmetic using the interval values of the cells referenced. The interval program is an equivalent of a spreadsheet program where the values of cells are represented as intervals and the computation is performed based on interval arithmetic. The third value referred to on the right side of Figure 6 is the interval \((E)\), the user expects as range for result values.

In order to infer the existence of a fault in a formula cell, the three values \(d\), \(E\), and \(B\) which are generated by different sources should be compared. There are two cases to consider.

**case 1: \(d \in E\) and \(E \subseteq B\)**

As the computed interval value of a formula is bounded by minimum and maximum values of the possible computation (by definition of interval arithmetic), the expected interval \(E\) should lie within the computed interval \(B\). Further, the value \(d\), computed by the spreadsheet program, should be within the expected magnitude of computation \(E\). Hence, in this case, we can say that there is no symptom of fault.

---

\(^4\)A mechanism that predicts the expected behavior of a program based on a specification
case 2: \( d \notin E \) or \( E \subset B \)

In this case, there is an indication of symptom of fault. The fault may be in the formula or in the user’s perception of expected results. Of course, testing is performed based on the assumption that there is a correct behavior of a program against which the actual result is compared. However, we can not always take for granted that the expected behavior is correct. In the situation where \( d \notin E \), due to some misreferences of cells in the formula or some other errors, the actual result is outside of the range of the expected result. In the second possibility where \( E \subset B \), faults affect the bounding interval computed for the formula and create a misalignment between \( E \) and \( B \).

This approach is mainly targeted to misreference and incorrect range specification errors. These errors are a result of specifying or selecting a group of cells incorrectly to achieve the desired goal of computation. Generally, we can say that these errors are failures in specifying a plan for a given computational goal. Misreference and incorrect range specification errors are likely to create a misalignment between the values computed by the spreadsheet program, interval program and the expected interval specified by the user. In addition, other errors may also create a discrepancy between the values \( d \), \( E \), and \( B \) and could be detected in the process. Once the existence of a fault in a formula is known, the source of the fault may be traced using the data dependency relation between cells established through the formula.

It has to be acknowledged that this form of interval testing plays a dual role. On one hand, it identifies faults in spreadsheet instances, whenever actual values \( d \) fall outside of the permitted range. On the other hand, the comparison between \( E \) and \( B \) is rather a check on the consistency of the user’s arithmetic model. Thus, this check can be quite powerful on a much more general level than on the level of a specific spreadsheet instance.

6 CONCLUSION

This paper attempts to overcome the tension between the statements “Spreadsheets are Software too” and “spreadsheet-authors are no Programmers” in order to improve the quality of spreadsheet software.

It is shown that there seems to be no single answer serving as silver bullet. However, a mix of approaches, close enough to the end-users’ conceptual model of plausible ranges for values of items as well as visualization of the mapping of conceptual structures to cell arrangements might help to highlight errors of frequently occurring nature.
REFERENCES

Session 3

Audit Methodologies and Techniques

Chairperson: Leon Strous,
Internal Audit Dept,
De Nederlandsche Bank, Amsterdam

Papers in this section:

Risk Assessment For Spreadsheet Developments

Visual Checking Of Spreadsheets

"The presence of a spreadsheet application in an accounting system can subvert all the controls in all other parts of that system".

Ray Butler in "Stop The Subversive Spreadsheet"
Risk Assessment For Spreadsheet Developments: Choosing Which Models to Audit

Raymond J. Butler, CISA
H. M. Customs and Excise, UK
Email ray.butler@hmce.gov.uk

© Crown Copyright reserved, published by permission of the Commissioners of H M Customs & Excise

ABSTRACT

Errors in spreadsheet applications and models are alarmingly common (some authorities, with justification cite spreadsheets containing errors as the norm rather than the exception). Faced with this body of evidence, the auditor can be faced with a huge task - the temptation may be to launch code inspections for every spreadsheet in an organisation. This can be very expensive and time-consuming. This paper describes risk assessment based on the "SpACE" audit methodology used by H M Customs & Excise's tax inspectors. This allows the auditor to target resources on the spreadsheets posing the highest risk of error, and justify the deployment of those resources to managers and clients. Since the opposite of audit risk is audit assurance the paper also offers an overview of some elements of good practice in the use of spreadsheets in business.

1. INTRODUCTION

There is a huge body of evidence (summarised in Panko, 2000¹ and Creely, 2000²) that errors in spreadsheet applications and models are alarmingly common (some authorities, with justification cite spreadsheets containing errors as the norm rather than the exception)

Spreadsheet users, developers, and auditors need to define controls in order to manage and reduce this risk.

Preventive Controls

Sound development methods, standards and user education are the obvious preventive controls, These have been described for almost as long as electronic spreadsheets have been available (e.g. Nevison, 1987³ and Batson & Eyles, 1995⁴)

The many examples of errors found in both field audits and experiments (Panko, 2000¹, Butler, 2000⁵) and studies such as that of Galletta & Hufflagel, 1992⁶ 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.

Detective and Corrective Control

Detective and corrective controls over spreadsheet errors are principally provided by detailed code inspection to check the formulas and where necessary the input data. This can be
• an entirely manual process, or

• One of the several computer-assisted audit tools for spreadsheets may be used.

Code inspection may be performed by

• one individual (not particularly effective - Galletta, Abraham, Louadi, Lekse, Pollailis and Sampler, 1993 found it to be less than satisfactory, detecting only around 50% of errors)

• by teams of 2 or more (More effective, finding around 80% of errors - Panko, 1996).

The Auditor's Problem

Faced with this body of evidence, the auditor can be faced with a huge task – the temptation may be to perform a full code inspection on every spreadsheet encountered. However, effective code inspection, even when aided by Computer-assisted audit tools, can be extremely resource intensive. Given limited resources, even with the anecdotal evidence of the inevitability of errors in spreadsheet model development, auditors must prioritise their work and justify the expense of code inspection to management and clients have to determine

• the potential impact of errors in a model on the organisation and

• The likely incidence of errors in the model.

This paper describes a risk assessment methodology used (with some success) by officers of H M Customs & Excise to determine which spreadsheet applications they need to test in depth in order to gain assurance that they calculate taxes and duties correctly.

A note on Terminology

The terminology used in this paper reflects the fact that Microsoft Excel is the spreadsheet programme used by H M Customs & Excise. However, the concepts and methods set out below apply to any electronic spreadsheet programme and (with minor modifications) to other areas of end-user computing. In this paper

• spreadsheet is used to mean the electronic spreadsheet program (e.g. Microsoft Excel, Lotus 1-2-3) used to develop a

• model - a generally complex single use development for financial or other planning purposes or an

• Application, which may be simple or complex but is generally used as a regular part of a business' operation.

2. THE RISK ASSESSMENT

The methodology described is a multi-stage process, which allows the auditor to make a “stop / go" decision at each stage Figure 1 illustrates this. The first two stages of Risk Assessment considers the environment within which the model or application is developed. The subsequent steps move on to consider the development itself.
2.1 OVERALL RISK ASSESSMENT

All audit planning is governed by the need to deploy resources to address quantified risks. The auditor must therefore determine both the impact and possible incidence of risks before setting out on a testing programme.

The first two stages of risk assessment - the overall assessment - are performed without looking at the detail of the model or application at all. All the steps, and the evaluation, may be taken relatively quickly, and the time saved by avoiding unnecessary code inspections amply justifies the effort of risk assessment.
Impact of Errors

The auditor must determine the:

- amount of money or other resources handled by or at risk from the application or model
- regulatory consequences of any errors
- impact of any errors on the organisation's public image, shareholder etc. confidence

The span of this will cover either the current instance or (if the current instance is a template for future re-use) a year or some other convenient period. The re-use of an application as a template for business operations will multiply both the resources handled and the impact of any errors. At this point the auditor can decide whether the amount at risk from the application justifies the work required to ascertain the likely incidence of errors.

Likely Incidence of Errors

When the Auditor has that the impact of an error is likely to be significant, the auditor must make a judgement as to the likelihood of that exposure occurring. To inform this judgement, the auditor must consider the answers to the following questions:

Organisation Questions

Does the organisation for whom the development is being made have an adequate policy governing development, testing and use of spreadsheet models and applications?

What evidence is there that this policy is observed and enforced?

Domain Questions

How complex are the business or revenue issues that the model or application purports to address?

Is there evidence that the developer of the model or application has

- An adequate understanding of those issues?
- Access to a clear, accurate, written explanation of the business issues?

If domain knowledge is partial or absent, errors in the base calculations or of omission are much more likely. Further, it follows that the developer will be less able to detect errors through a “reasonableness test” of the output from a model or application.

Specification Questions

- Is there a clear statement of the inputs, processes, outputs and results required for the model or application?
- Is it complete and accurate?
- Has the user agreed it?
• Does it include agreed measures of the success of the development?

• Does it include a testing plan?

If no specification exists, then domain and arithmetical errors are much more likely, and there will be no control against which the completeness and accuracy of the results can be judged. Developing a complex spreadsheet application or model without a specification is akin to walking across a swamp without a map - the only measure of success is surviving the experience.

Testing Questions

What evidence is there that the application was thoroughly tested before being brought into use? And thoroughly tested again each time a material change was made? If a model or application can be shown to have given

• sensible answers when tested with simple numbers (for example, if the answer to 
  \[+5+5\] is 7 then there's a problem somewhere) and

• the results predicted when realistic test data are processed and

• the results predicted when running the model in parallel with previous systems

Then there is a good level of assurance that the incidence of errors will be low.

Documentation Questions

Has the developer documented the application adequately? Good documentation should make clear statements of:

• the application's purpose, what it does and how it does it

• any assumptions made in its design

• what standing data constants (e.g. tax, duty, interest and exchange rates) are used and where they are held

• who developed it and when, and

• When and how it has been changed since being brought into use.

• How the application or model should be used.

The absence of documentation has been a factor in a number of well-documented spreadsheet errors. Where a developer is not the end-user of an application, where any application is more than utterly simple, and wherever the developer will not be maintaining and updating the application documentation is absolutely essential. Failure to document the inner workings of a model simply stores up trouble for whoever has to amend or maintain it. Failure to document its correct operation by the end-users can lead to serious errors, especially in an environment where a model or application is passed from user to user.

Questions about the complexity of the application
How complicated is the application? Is it laid out logically, with data and calculations in separate areas, and with complex calculations broken down into stages? Is the arithmetic of the calculations clear from the visible information, allowing it to be checked manually for accuracy and completeness?

Data Control Questions

In common with all computer applications, the accuracy of the results of processing depend on the completeness, accuracy, timeliness and authorisation/appropriateness of the data. Even when a properly validated specification has been verified as implemented correctly, with all domain and arithmetic issues correctly handled, the GIGO (for younger readers, Garbage in, Garbage Out) principle still applies.

The auditor must therefore ask what controls are built into the application to ensure that:

- all relevant data are input,
- no irrelevant or inappropriate data are input
- data are input accurately, and
- data are input for process at the correct time

Evaluation

The auditor must consider the answers to these questions to inform a decision whether or not it is worth proceeding to the next stage. It is important that the quality of documentation, test plans, etc is taken into account in this decision - good practice in these areas is very rare, and auditors must beware of being led to a false sense of security by the very existence of documentation, user instructions, etc.

Given that

- the amount of resource potentially at risk, and
- the evaluation of the above factors

Shows that the impact and likelihood of errors justifies further work, the auditor can progress to the next stage of risk assessment. This stage determines whether a full code inspection may be required, how much effort may be involved in it, and whether the effort will be justified by the risk.

2.2 RISK IDENTIFICATION & SCOPING

Given that the impact and likelihood of error justify further work, the auditor now needs to establish

- the size and complexity of the application (to help plan the time needed to test it),
- Which parts of it pose the highest risk (to help direct the tests to those risk areas).
This step requires access to the model or application, since its composition and set-up have to be assessed in order to inform the stop - or- go decision for code inspection. It is greatly eased by the use of spreadsheet audit software, which can quickly reveal the inner workings of an application.

**Size of the task**

To determine the complexity of the checking task, and the amount of time that may be needed to perform a code inspection, the auditor needs to know:

- How many physical files are involved in the application or model?
- for each file, how many worksheets are present
- if data is passed from one file or worksheet to another, are adequate controls in place to ensure the completeness and accuracy of the transfer?

Transposition of digits and other keying errors can easily corrupt the receiving file or worksheet. Similarly, automatic links between files must be subjected to controlled for completeness, accuracy and appropriateness of data transfer. This allows the auditor to determine the boundaries of the audit. Within these boundaries, the auditor then needs to establish for each worksheet within each file:

- how many formulas are present,
- how many numbers are manipulated,
- how many labels are present and
- How many links to other worksheets exists.

Many of the audit support software products allow this information to be produced automatically. This information is used to inform decisions on

- time management (how much time is code inspection likely to take?) and
- Resource to risk (does the money at risk justify spending that time?).

**How complex is the task?**

Assuming the auditor has justified expending further resources, a better indication of the time that has to be expended on testing is needed. The auditor now has to establish how many

- external references,
- unique formulas (i.e. those which are not replicated in a worksheet), and
- original formulas (i.e. those which are copied within a worksheet)

Arepresent in each file. The degree to which similar worksheets are used within a file or across a series of (ostensibly) identical files is also a factor in determining the resources needed for a code inspection. Once a master original worksheet has been tested and if necessary corrected and documented, to establish a norm for the audit, automated comparison of worksheets can quickly
identify any divergences from that norm in copies. This can significantly reduce the amount of actual code inspection testing that has to be performed.

The auditor has to consider how complex the business issues the application addresses are, and how complex its structure and logic are. Drawing a map or flowchart of the application at this point can help comprehension of the structure and interaction of its components.

**Identification of Set-up Risks**

Identification of the use of high-risk functions or features, and an assessment of the use of security features and the way the application is set up helps the auditor to judge the amount of risk and the amount of work needed to test the application. The auditor needs to establish:

- The recalculation settings of each file
  - Manual or automatic?
  - If manual, is it set to recalculation before save set?
  - How are iteration & calculation rules set?
- Whether macros and user-defined functions are present, or if indications are present of any traces of their use - with modern spreadsheets, macros are often an attribute of the user's individual set-up rather than the spreadsheet file itself,
- Whether Hidden Rows, Columns or sheets are present in the file,
- Whether protection against unauthorised changes is present
- Whether advanced features such as consolidation, scenarios, goal seeking, solver, pivot tables, report or view manager and equivalent features are used.

If the results of a model or application depend on the use of these features, the auditor will have to consider whether the techniques are appropriate to resolution of the issues being dealt with and whether they are being used correctly.

- Whether range and variable names are being used in formulas - this can indicate a developer's use of good practice. However, the auditor will have to consider whether names are being used correctly.

**2.3 THE TESTING DECISION**

At this stage, the auditor will know:

- the amount at risk and the likely incidence of risk from the model or application
- the amount of effort that is likely to be required in order to manage that risk by substantive testing
- whether the balance of the amount and incidence of risk justifies that further work, and (as a by-product of the risk assessment/compliance testing); and
- Which areas of the application may require detailed scrutiny
2.4 RISK IN THE CODE INSPECTION PHASE

Code Inspection

Code Inspection will, if supported by adequate software, be targeted on risks that –

• Original formulas copied around the worksheet or workbook are arithmetically and or logically incorrect.

• Copies of those formulas are used inappropriately.

• Unique formulas are arithmetically and logically incorrect.

• Formulas have been over-written by numbers or other data.

Additional Risks in the Code Inspection

Even if they appear arithmetically and logically correct, further checks will need to be made on formulas that present a high risk of error, i.e. those which

• look up named ranges, e.g. standing data;

• contain constants (e.g. net * 17.5% instead of net * a named variable or range "VAT rate");

• contain absolute references (which will not automatically respond to changes to the sheet);

• reference a block of cells (e.g. SUM A1:B7 may indicate errors in input of the formula);

• Have no precedents (e.g. additions of numbers within a cell, which invariably gives rise to audit trail problems).

Depend on

• numbers formatted as text (which may cause errors or unpredictable results), or

• blank cells (which may reveal errors in construction or in data input),

• have no dependant cells (if not the end result, may be an error),

• address hidden cells, rows or columns,

• address cells which fail or return an error message,

• address linked sheets and workbooks,

• have an inherently high risk of user error (e.g. =NPV) or

• show as an apparent break in the pattern of formulas copied from a single source -
Most test support software will help the auditor to identify these.

**Data Checking**

Given the high risk of errors in formulas, it can be easy to overlook the issues of errors in the data - as stated above, the GIGO principle applies to spreadsheets as much as to any computer application. The auditor will already have risk analysed the procedures and controls over data, and depending on the presence (or absence) and quality of systems in place to assure the completeness and accuracy of data may need to substantively test the base numbers and standing data used. In particular, the auditor must consider the risks that:

- incorrect or inappropriate raw numbers could be introduced into the model;
- numbers may be incorrectly input in place of formulas;
- numbers may have been introduced and not used by any formula; and
- numbers may have been incorrectly formatted as text, leading to their omission from totals, etc.

All these circumstances are known to pose a high risk of error.

**3. CONCLUSIONS**

Risk assessment is at the heart of all auditing, whether of manual accounts, complex enterprise resource planning systems, or of spreadsheets.

The risks of error arising from poor practice in the use of spreadsheets are known to be high, and the incidence of good practice in developments using spreadsheets are known to be low. Despite this, users are blissfully unaware of these risks and are using potentially faulty decision support machinery every day to take vital business decisions.

Given this, and that an error in a spreadsheet application can subvert all the controls in all of the systems which feed data into it, risk assessment in spreadsheets is vital if good practice is to be encouraged, bad practice detected and corrected, and appropriate resources put into auditing by data and code inspection.

This paper is offered as a starter for further research into the effectiveness of error prevention and detection methods and to inform future audits by IS and other auditors.
## REFERENCES


2. Creely, Paul "Review of Errors in Spreadsheets", *Final Year BSc Paper, University of Salford* 2000


5. Butler, R J 'Is This Spreadsheet a Tax Evader? How H M Customs & Excise test Spreadsheet Applications" *Proceedings of the 33rd Hawaii International Conference on System Sciences, Maui, Hawaii, 2000*


VISUAL CHECKING OF SPREADSHEETS

Ying Chen and Hock Chuan Chan
School of Computing, National University of Singapore, Singapore
(Contact author: Hock Chan, email: chanhc@comp.nus.edu.sg)

ABSTRACT

The difference between surface and deep structures of a spreadsheet is a major cause of
difficulty in checking spreadsheets. After a brief survey of current methods of checking (or
debugging) spreadsheets, new visual methods of showing the deep structures are presented.
Illustrations are given on how these visual methods can be employed in various interactive
local and global debugging strategies.

1. INTRODUCTION

Spreadsheets are easy to use and very hard to check. Large percentages of spreadsheets in
actual use have been found to contain errors, and users do not have effective means of checking
spreadsheets. The main difficulty that users have in understanding spreadsheets is the difficulty
of establishing cell connections - how each cell depends on other cells. These connections form
a structure. The usual tabular layouts for texts and numbers do not show the actual connections.
They may even suggest a wrong structure. The difference between the surface structure (what
the cell arrangements and values suggest to the user) and the deep structure (what the cell
connections really are, as described in the formulas) can add to the difficulty in understanding
spreadsheets.

Some systematic ways of creating spreadsheets have been recommended\(^1\). This approach
specifies some design requirements so that the spreadsheet will be easier to maintain and be
understood by someone other than the developer. The possibility of making errors may be
reduced. However, errors can still occur, even if users are generally careful. Spreadsheet
checking is also necessary at the development stage.

Currently, deep structures can be understood only by examining formulas one by one. This
process is extremely tedious and error prone. Researchers are trying to make the deep structures
visible, through visual drawings. Various visual tools and strategies for using these tools are
presented, so as to help spreadsheet users understand their spreadsheets faster and more
thoroughly.

2. SPREADSHEET STRUCTURES

Cells in a spreadsheet are organised as tabular form. Related cells are often put together.
Naturally, users deduce the spreadsheet structure through relative positions of cells as they
visually cluster them together. The spreadsheet structure is obtained through the visual
scanning of cells. So it is called visual/surface structure. However, spreadsheet
calculations are based on the formulae in cells. Cells are connected through formulae and form another kind of structure, called computational/deep structure. It reflects data flows in a spreadsheet.

Table 1(a) shows a spreadsheet with some values. The "visual" relationships among cells, identified by users through the relative spatial positions of cells in the spreadsheet provide a surface structure. At first sight, a user may form an impression that column A is collapsed into column B, column B into column C and column C into column D. The dotted arrows represent the supposed surface level of this spreadsheet. Table 1(b) shows the deep structure, with formulas determining the actual cell connections. An arrow goes from a precedent cell to a dependent cell (a dependent cell's formula refers to a precedent cell). This structure is different from table 1(a). The inconsistency between surface level and deep level can happen in two major ways: (1) References in a single formula is non-unitary and non-organised, as in E6 * F9 * A5 * C4; and (2) References in several formulas are interwoven in “the cell dependency network”.

<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>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1(a). The surface structure

<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>1</td>
<td>1</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1(b). The deep structure

Table 1. the structures of a spreadsheet

It is difficult to understand a spreadsheet when the two structures in the spreadsheet are very inconsistent, because information search is controlled by the surface structure. The memory load that is used to extract the deep structure will increase largely. As a
consequence, users will spend more time on learning and memorizing the deep structure than doing spreadsheet calculations.

This suggests that surface structure could be used effectively as mnemonic aids if the layout allows the direct spatial mapping between the elements of surface structure and deep structure. Some rules can be used to create well-organised formulas (the links in networked formulas are not crossed and their direction are uniform). Then the memory load goes down.

It is worth noting that some specialized spreadsheet programs have been developed to make the deep structures explicit during the model development stage. For example, Spreadsheet 2000 (http://www.emer.com/s2k/) avoids the typical tabular spread of cells, and thus totally avoids giving any surface structure. Models are built with predefined metaphors, such as operators, grids, charts and forms. This avoids the problem of surface structure misunderstanding. Another software, Storeys (http://www.profunda.com), allows users to built 3-dimensional spreadsheets in a more explicit manner, compared to the usual 2-dimensional worksheets. This again avoids to some degree the surface structure of a 2-dimensional spreadsheet when the model is really 3-dimensional.

Given that the most popular spreadsheets (from VisiCalc to Lotus 1-2-3 to Microsoft Excel) adopt the general tabular format without explicit structures, a very important approach is to dispel the surface structure by displaying the deep structure directly. The visibility of deep structures can help spreadsheet users put most of their attention to checking rather than memorizing the structures.

Structures can be viewed at two levels of granularity: Cell and Module. At cell level, the basic elements in the structure are cells. The structure represents the relationships among cells. Any change in the structure is reflected at cell level. In contrast, the module level is used to describe spreadsheets at the macro level. A module is made up of a set of cells that have the same function. The structure describes relationships among modules. It is a high level view of how a spreadsheet model operates.

At module level, each module element is a block of cells. The definition of a block at surface level is different from that at deep level. At deep level, a block is defined by its function, e.g., Input, Output, Decision, Parameter and Formulae[^1], or simply Input, Process and Output. The relationships among the blocks represent the logic flows in a spreadsheet model. In contrast, at surface level, blank cells divide a spreadsheet into different blocks. The cells in a block may not represent a separated function of a spreadsheet. However, the spatial position of a block may represent corresponding functions: Input part is usually put on the upper or at the left of a spreadsheet while Process part is put in the middle and Output part is put at the bottom or the right.

The same problem of inconsistency between surface and deep level exists at module level as well as at cell level. The usual method to solve this problem at module level is to follow the requirements of structured design: Put all relevant cells into the corresponding spatial position of one functional part.

Accordingly, debugging tools should let users check spreadsheets locally (at cell level) and globally (at module level) so that errors can be detected more easily.
3. CURRENT METHODS

This section reviews the current dominant methods to find errors in spreadsheets.

**Presentation**

Lacking good visualization tools, presentation is an obvious method for manual error detection. Spreadsheets can be shown on-screen, or printed on paper, and with or without formulas. Experiments suggested that spreadsheets are better understood with formulae on paper than with numbers on screen. However, detection rates for all these presentation methods are very low.

**Program Code inspection**

Theories from error detection in program codes were adopted to discover errors in spreadsheets. However, spreadsheets are quite different from program codes. Firstly, program codes are represented in a one-dimension way while cells in spreadsheets are listed in a two-dimensional form. Secondly, the models (formula and deep structure) of a spreadsheet are buried in the surface data, unlike a program where program codes are organised according to some kinds of semantic and syntax. Some restrictions are imposed on the program codes to make them meaningful and easy to be understood. Thirdly, few organizations have comprehensive policies for spreadsheet development while software engineering has built good rules for programmers' use. The development and detection phases are so tightly correlated that good development will make detection far easier. So the ways to detect errors in programs cannot be completely applied to error detection in spreadsheets. The average error detection rate for individuals is 63%. Some errors are still left in spreadsheets. It is necessary to develop more suitable ways to detect errors in spreadsheet.

**Spreadsheet Analysis Reports**

A number of software packages have been developed to analyse spreadsheets and present reports, mainly for auditing purposes. Early products, such as Cambridge Spreadsheet Analyst and Spreadsheet Auditor, to help analyse spreadsheets have appeared since 1986. More recent products include the following. The Spreadsheet Detective (http://www.gg.net.au/detective/) analyses a spreadsheet and adds cell shadings and comments to help users identify errors. Operis Analysis Kit (http://www.operis.com/) is another add-in for Excel 97, and helps to identify various kinds of spreadsheet errors. Spreadsheet Professional (http://www.eastern-software.com/) also provides analysis to help identify spreadsheet errors.

**Interactive tools**

Recent technological advances allow researchers to visually explore the discrepancy between surface and deep structures of spreadsheets. Tools have been developed to help users identify data flows at cell and module levels. Davis described two interactive tools for auditing spreadsheets: an on-line flowchart-like diagram and a tool which represents “cell dependency network” by drawing arrows cells on the spreadsheet. Both tools were found to be helpful in investigating cell dependencies and debugging. Both tools show better error detection than the traditional screen / paper presentations. The arrow tool was
the best. It was also found that users prefer visual interactive tools rather than paper printouts or screen scanning.

At a higher level, Isakowitz et al. developed an algorithm to extract logic structures from spatial layouts automatically. Four principal components are identified from the original spreadsheet: schema, data, editorial and binding. Then users can easily construct logic flows from schema. It is an easier way to check spreadsheets. Users can debug spreadsheets with schema and data and avoid cell-by-cell checking. One drawback related to this technology is that there is no direct mapping between spreadsheets and schema. Users need to try to recognise the corresponding components of schemas in spreadsheets. So these schemas impose a heavy memory load on users.

To make interactive tools more convenient, Igarashi et al. described ways to visualise spreadsheet structures at varying levels of connections. Initially, first-level precedents and dependents of a cell are shown. Subsequently, the entire structure of a spreadsheet related to one cell is displayed. Furthermore, displays can be static or animated. However, it has some limits: spreadsheets cannot be checked area-by-area or module by module; all precedents or dependents of a cell cannot be obtained at once and display is based on only one cell at a time. Microsoft Excel spreadsheet provides the same functionality in its set of auditing tools. Various levels of precedents and dependents for a selected cell can be displayed, with arrows.

4. NEW VISUAL METHODS

A set of new visual tools for understanding spreadsheets is briefly presented here. These tools help to highlight different functional parts, data flows and levels of a spreadsheet. These may allow users to debug their spreadsheets with proper strategies. These have been implemented in Excel Visual Basic for Applications, and can be added in to any Excel installation.

1. Functional Identification Tool

All cells in a selected region are classified into four kinds: Input, Output, Processing and Standalone. Input cells have dependents but no precedents while output cells have precedents but no dependents. Cells with precedents and dependents belong to processing parts. Standalone cells have neither precedents nor dependents. Different colours are used to fill these four types of cell so that users can see the function of cells immediately.

2. Multi-precedents and Dependents Tool

The multi-precedents/dependents tool shows the precedents/dependents of all cells in a selected block. It can illustrate relationships among cells beyond one level at one click.

3. Block-precedents Tool

This defines a set of similar cells as a block. Their precedents in different neighbouring areas make up different blocks. This block-precedents tool shows the relationships wrong blocks, not cells. One level is added at one click.

4. In-block-precedents-dependents Tool

In-Block-Precedents-Dependents shows cell connections within a selected region, without incoming and outgoing arrows connected to this region. It can be applied when users are only concerned with cell relationships in a particular region. All distractions outside this region can be avoided.
5. **Separated-blocks Tool**

This tool identifies groups of connected cells. There are no connections across groups. Different cell and arrow colours are used to mark different groups. Each group represents a separate model in the worksheet. So different models are easily identified.

6. **Level Label Tool**

Level label assigns a level number to cells in a selected block in terms of the longest distances between the cells and input cells. Input cells can be checked first while output cells can be checked last. The order to check contents in cells can help to minimise repetition and disorderliness.

5. **STRATEGIES**

A new tool should ideally be coupled with a set of strategies that defines the best way to use it. These strategies could be developed by experts familiar with the new tools, and can form a basis for other users to apply the new tools in their understanding of spreadsheets and detection of errors. Some strategies for using the new tools are presented below.

Understanding a spreadsheet relies on a good understanding of the "Cell dependency network". Data effects flow from the input to the output cells. Due to the flow direction, "Cell dependency network" can be regarded as a directed graph. The value of a cell in the flow can only be affected by its precedents. So if users are sure where the problem falls or which functional parts the problem belongs to, users can concentrate on a specific area and local debugging strategies are enough. However, if users have no idea about what their spreadsheets do, global debugging strategies are more suitable to help users establish thorough debugging steps. Next, these two debugging strategies are illustrated.

![Table 2. An example of a broken link](image)
Local debugging strategy I: Broken Link

One common error occurs when a user accidentally types a value instead of a formula. It is very difficult to catch because the value in this cell is correct, until such time when the values of precedents of this cell are changed. Two methods can be used to catch this type of error: functional identification and in-block-precedents-dependents.

For example, table 2(a) shows a simple spreadsheet. Table 2(b) and 2(d) show the correct visual result after applying the functional identification tool or the in-block precedent dependent tool. If a user accidentally enters a value instead of a formula in cell 133, then applying the same tools will result in the visual effects shown in table 2(c) and 2(e). The pink coloured cells (cells B1 and B2) in table 2(e) alert the users to errors, since pink coloured cells are standalones, and not used by other cells. In a well-structured spreadsheet, all cells should be connected. Some exceptions could occur, such as in column headings. So it is easy to find that the correct form of 133 should be B1 - B2 and not 10000. Table 2(e) shows a shortened chain of arrows, starting from the middle of the model. Hence, it is easy for users to spot this error.

Local debugging strategy II: Unwanted Link

Due to similarity within a set of formulas, relationships among several sets of cells may be regular. Any irregularity indicates possible error. Multi-precedents/dependents may help users to catch irregularities. For example, the wrong cell reference of C4 in Table 3(a) is easily caught by applying multi-precedents to the block of cells C2 to CS. A diagonal downward arrow is easily spotted.

In Table 3(b), users find that values for % change for different regions contribute to the overall % change after multi-dependents is applied to cells B4 to B7. From logical consideration, it is somewhat strange since interest rates are seldom aggregated. So the logical error is caught.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deposit</td>
<td>Rate</td>
<td>Payment</td>
</tr>
<tr>
<td>2</td>
<td>3000</td>
<td>0.03</td>
</tr>
<tr>
<td>3</td>
<td>5000</td>
<td>0.035</td>
</tr>
<tr>
<td>4</td>
<td>8000</td>
<td>0.05</td>
</tr>
<tr>
<td>5</td>
<td>10000</td>
<td>0.06</td>
</tr>
</tbody>
</table>
Local debugging strategy III: Localised Links

A spreadsheet may be large and complicated. To check many cells in a large region at a time may include many irrelevant details. One strategy is to examine the links within a small selected region. Users can use the in-block-precedents-dependents tool to delve into details of a selected block, not distracted by connections to cells outside the block.

Global debugging strategy I: Localization

Local checking can allow users to delve into the details. However, users may want to know the data flow at a higher level or do not want to miss any part of a spreadsheet. Global checking strategies may help users identify the logical structure or to plan thorough debugging steps. The first strategy helps user identify sub-regions within a region.

In the example in table 4, which shows a correct model, the functional identification tool is first applied to identify input, output, processing and standalone parts. The light green, dark green and grey cells show the different cell functions. Users can have an initial impression of the spreadsheet structures. The functional modules of the spreadsheets are also identified by the different colour regions. Output modules are to the right and bottom of the spreadsheet (most of column H, and rows 20 and 21 in the table); input modules occupy the centre (columns B to G, in rows 5 to 9, and 13 to 16); and processing modules are in two rows (row 10 and row 17 in the table).

<table>
<thead>
<tr>
<th>2</th>
<th>Fixed Assets</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Cost</td>
<td>Freehold</td>
<td>Leasehold</td>
<td>Vehicles</td>
<td>Equipment</td>
<td>Computers</td>
<td>Electrical</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Balance at 1.4.80</td>
<td>2,105,165</td>
<td>52,422,655</td>
<td>2,500,444</td>
<td>880,355</td>
<td>2,000,500</td>
<td>120,500</td>
<td>60,029,619</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Additions</td>
<td>188,367</td>
<td>1,826,545</td>
<td>388,567</td>
<td>455,676</td>
<td>380,600</td>
<td>1,290,500</td>
<td>4,440,255</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Disposals</td>
<td>(55,650)</td>
<td>(75,800)</td>
<td>(205,900)</td>
<td>(100,800)</td>
<td>(9,050)</td>
<td>(9,050)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Transfer to other assets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Translation difference</td>
<td>23,165</td>
<td>12,565</td>
<td>12,885</td>
<td>12,565</td>
<td>51,171</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Balance at 31.3.81</td>
<td>2,316,845</td>
<td>54,249,240</td>
<td>2,845,828</td>
<td>1,273,116</td>
<td>1,218,715</td>
<td>1,220,200</td>
<td>64,083,845</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Localization
Users can use the multi-precedents tool to check similar output cells to detect any irregularity or critical output cells to identify data flows. The arrows in Table 4 are the results of applying the multi-precedents tool to B20 to G20. The logical flows in B5 to G20 can be easily identified at a glance. More importantly, this shows that each column is in effect independent. Column values are summed in column H, but otherwise they are independent. Thus, the overall region has been decomposed into single columns. An alternative tool to use is the block-precedents tool. There are fewer arrows, providing a clearer view.

**Global debugging strategy II: Separation**

Table 5 illustrates the strategy of separation. Unrelated groups of cells are identified using the separated-blocks tool. Different colour arrows visually identify four models in the worksheet. There are no connections among them and no standalone parts. All cells are properly connected.

With the identification of separated groups, the user can now focus on each group without being distracted by other cells. In some cases, separated groups mean errors if the groups are not meant to be separated. For example, some links could be missing.

**Global debugging strategy III: Stratification**

The arrows linking an output cell and all of its precedents form a tree. The output cell is correct if the formula and the first level precedents of this cell are correct. It is same for all cells in a spreadsheet. So if a cell is checked after all of its precedents have been checked, the repetition of checking can be avoided. The checking efficiency is improved. Hence, all cells are stratified with the level label tool. All cells are assigned to a level number according to the longest distance from them to the input cells. In Table 5, the corresponding level label for each cell is given. Checking should proceed from level one onward. In this sequence, output cells are checked last while input cells are checked first. Then users have a visual plan to examine the spreadsheet level by level without repetition.

**6. CONCLUSION**

Spreadsheet errors are common, and can have serious decision consequences. In order to help users understand spreadsheets and identify errors in a visual manner with lower cognitive loads, various local and global strategies with a new set of interactive tools are presented. Local debugging strategies help users delve into details while global debugging strategies help users establish thorough and more systematic debugging steps.

Compared to the earlier error detection methods, this new set of tools can make deep structures more visible and obvious by the arrows connecting precedent and dependent cells. The tools provide more convenience by enabling all precedents and dependents of a group of cells to be shown at once, instead of requiring many clicks from the user. Cell relationships can also be limited to a selected region, so as to allow more focused understanding and detection of errors. Visual comprehension is facilitated through the use of different colours to represent different functional parts in a spreadsheet. Visual aids by the interactive tools can make users concentrate on the important task of verifying the model and not on remembering and recalling cell connections.
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Share Capital and Reserves</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Share capital</td>
<td>45000</td>
<td>45000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Capital reserves</td>
<td>99200</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Retained profits</td>
<td>17768</td>
<td>35788</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Represented by:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Fixed Assets</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Other investment</td>
<td>268241</td>
<td>9846</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Deferred Taxation Asset</td>
<td>22026</td>
<td>7667</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Current Assets</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Stocks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Trade debtors</td>
<td>13893</td>
<td>9996</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Other debtors and deposits</td>
<td>3822</td>
<td>423</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Fixed deposits</td>
<td>6353</td>
<td>102882</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Cash</td>
<td>132748</td>
<td>100404</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Total current assets</td>
<td>14996</td>
<td>5386</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Less current liabilities</td>
<td>191312</td>
<td>219091</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Trade creditors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Provision for claims</td>
<td>108593</td>
<td>10221</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Provision for upgrades</td>
<td>10960</td>
<td>74963</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Fuel price equalization</td>
<td>5604</td>
<td>14691</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Loan</td>
<td>53127</td>
<td>52227</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Income tax payable</td>
<td>72000</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Proposed dividends</td>
<td>12000</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Total current liabilities</td>
<td>13048</td>
<td>2700</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Net Current Assets</td>
<td>275532</td>
<td>158931</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Less: Non-current liabilities</td>
<td>-38326</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Provision for replacement</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Provision for service benefits</td>
<td>3442</td>
<td>4188</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Deferred taxation liabilities</td>
<td>28837</td>
<td>3000</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Total non-current liabilities</td>
<td>12180</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Net Assets</td>
<td>252759</td>
<td>9188</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5. Separation and Stratification
REFERENCE


Session 4

Software Tools: practical demonstrations

Chairperson: Raymond Butler
Computer Audit Unit,
H.M. Customs & Excise

EuSpRIG is committed not only to the academic pursuit of working methodology and good practice but also to the dissemination of knowledge concerning useful software tools and commercial products. There are no papers in this section as it consists entirely of practical demonstrations of software.

For many organisations, and for many years, Gavin Potter’s 'Spreadsheet Professional' has been the main tool of the spreadsheet auditor’s software arsenal. Today, there are several more products available to the auditor.

Anthony Berglas has had success with his 'Spreadsheet Detective' tool and Operis have entered the market with their Operis Analysis Kit (OAK). Not to be outdone, HM Customs and Excise have produced their recent SpACE product (SPlreadsheets Audit for Customs & Excise).

"The sooner we can try out our methods on real-world users with real-world problems the sooner we can come up with a methodology of use to all"

Brian Knight in "Stop The Subversive Spreadsheet"
## Author’s Index

<table>
<thead>
<tr>
<th>Name</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ayalew, Yirsaw</td>
<td>51</td>
</tr>
<tr>
<td>Butler, Raymond</td>
<td>iii, 65, 87</td>
</tr>
<tr>
<td>Chadwick, David</td>
<td>1, 23, 43</td>
</tr>
<tr>
<td>Chan, Hock Chuan</td>
<td>75</td>
</tr>
<tr>
<td>Chen, Ying</td>
<td>75</td>
</tr>
<tr>
<td>Cleary, Patrick</td>
<td>19</td>
</tr>
<tr>
<td>Clermont, Markus</td>
<td>51</td>
</tr>
<tr>
<td>Hawker, Andrew</td>
<td>35</td>
</tr>
<tr>
<td>Knight, Brian</td>
<td>23, 43</td>
</tr>
<tr>
<td>Mittermeir, Roland</td>
<td>51</td>
</tr>
<tr>
<td>Panko, Raymond</td>
<td>7</td>
</tr>
<tr>
<td>Rajalingham, Kamalesen</td>
<td>23</td>
</tr>
</tbody>
</table>
This volume contains the unedited proceedings of the first European symposium on spreadsheet risks, audit and development methods, EuSpRIG 2000, held in July 2000 at the University of Greenwich and sponsored by the Information Systems Audit and Control Association.

The objectives of this symposium are to promote discussion and co-operation amongst those concerned with authorising, auditing or 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 expertise and research. The topic areas include the type of errors, development methodologies, audit practice and tools.

ISBN: 1 86166 158 4