Enterprise Spreadsheet Management: A Necessary Evil?

11-13 July 2007 University of Greenwich, London
European Spreadsheet Risks Interest Group

EuSpRIG 2007 Conference

Enterprise Spreadsheet Management: A Necessary Evil?

Conference Sponsored By:
Compassoft

And
AuditNet
Baker Tilly
Information Systems Audit & Control Association
(Northern England Chapter)
SecureXLS
Spreadsheet Engineering
Systems Modelling
The University of Greenwich
University of Wales Institute, Cardiff

Editor:
David Ward

July 2007
University of Greenwich, London
Proceedings of EuSpRIG 2007 Conference
Enterprise Spreadsheet Management: A Necessary Evil?
First published 2007
Published by University of Wales Institute, Cardiff
Printed in United Kingdom by Baker Tilly

Held at University of Greenwich, London; July 11th-13th 2007

Copyright: European Spreadsheet Risks Interest Group
www.eusprig.org


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 the Publisher who is the Committee of the European Spreadsheet Risks Interest Group (www.eusprig.org).

Conference Organised by
EuSpRIG Committee

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.

Cover design by David Ward,
photos : http://www.essential-architecture.com/10/14-Greenwich_Hospital_from_Thames.jpg and
EuSpRIG 2007 Conference

Enterprise Spreadsheet Management: A Necessary Evil?

ORGANISING COMMITTEE
Patrick Cleary (chair), University of Wales Institute Cardiff, UK (Chair)
David Chadwick, University of Greenwich, UK
Ray Butler, Highways Agency, UK
Simon Thorne, University of Wales Institute Cardiff, UK

PROGRAMME COMMITTEE
David Chadwick (Chair), Univ of Greenwich (Chair)
Pat Cleary, University of Wales Institute Cardiff, UK
Ray Butler, Highways Agency, UK
Grenville Croll, Spreadsheet Engineering Ltd, UK
Roland Mittermeir, University of Klagenfurt, Austria
Patrick O’Beirne, Systems Modelling Ltd, Ireland
David Ward, Baker Tilly, London, UK

EUSPRIG MANAGEMENT COMMITTEE
Patrick O’Beirne : Chairman
David Ball : Eusprig Treasurer
Grenville Croll : Eusprig Membership Secretary
Patrick Cleary : Eusprig Secretary
Ray Butler : Member and past chairman
David Chadwick : Member and past chairman
David Colver : Member
Roland Mittermeir : Member
Jocelyn Paine : Member
Simon Thorne : Member
David Ward : Member
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Page</th>
<th>Paper</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Key Note Speaker: The Regulator’s View</td>
<td>Dean Buckner, Financial Services Authority, UK</td>
</tr>
<tr>
<td></td>
<td><strong>Session 1: Good and Evil in Enterprise Spreadsheet Management</strong></td>
<td><strong>Chair: Patrick O’Beirne</strong></td>
</tr>
<tr>
<td>7</td>
<td>Enterprise Spreadsheet Management – A Necessary Good</td>
<td>R. Baxter, ClusterSeven, UK</td>
</tr>
<tr>
<td>15</td>
<td>Spreadsheet Hell</td>
<td>S. Murphy, Codematic Ltd, UK</td>
</tr>
<tr>
<td>21</td>
<td>Managing Linked Spreadsheets in a Compliant Environment</td>
<td>S. Saadat Prodiance Corporation, USA</td>
</tr>
<tr>
<td></td>
<td><strong>Session 2: Managing the Corporate Risks from Spreadsheets: Risk management Strategies</strong></td>
<td><strong>Chair: David Chadwick</strong></td>
</tr>
<tr>
<td>37</td>
<td>Categorisation of Spreadsheet Use within Organisations: A Progress Report</td>
<td>M. Madahar, P. Cleary, D. Ball, UWIC, UK</td>
</tr>
<tr>
<td>47</td>
<td>Awareness and Market Changes in the Control of End User Computing Risk Management</td>
<td>P. Bach - Sponsor's Paper CompassSoft Inc, USA</td>
</tr>
<tr>
<td></td>
<td><strong>Session 3: Managing the Corporate Risks from Spreadsheets: Errors &amp; their Impact</strong></td>
<td><strong>Chair: Ray Butler</strong></td>
</tr>
<tr>
<td>57</td>
<td>Impact of Errors in Operational Spreadsheets</td>
<td>S. Powell, B. Lawson, K. Baker, Dartmouth College, USA</td>
</tr>
<tr>
<td>69</td>
<td>Spreadsheet Errors – What the Research Says</td>
<td>R. Panko, University of Hawaii, USA</td>
</tr>
<tr>
<td></td>
<td><strong>Session 4: Towards Heaven – Development Methodologies and Controls</strong></td>
<td><strong>Chair: Prof. Ray Panko</strong></td>
</tr>
<tr>
<td>81</td>
<td>Spreadsheet Application Software: Guidelines for Source Code Control and User Interface Design</td>
<td>T. Grossman University of San Francisco, USA</td>
</tr>
<tr>
<td>105</td>
<td>Fun Boy Three Were Wrong: It Is What You Do (Not the Way That You Do It): a tutorial on modularity</td>
<td>J. Paine, Spreadsheet Factory</td>
</tr>
<tr>
<td>117</td>
<td>Concerning the feasibility of example driven modelling techniques</td>
<td>S. Thorne, D. Ball, Z. Lawson UWIC and Cardiff University, UK</td>
</tr>
<tr>
<td>131</td>
<td>Practical Experience With Excel: Modular Construction of a Big Housing Finance Spreadsheet</td>
<td>J. Paine, Spreadsheet Factory</td>
</tr>
</tbody>
</table>
## TABLE OF CONTENTS (continued)

<table>
<thead>
<tr>
<th>Page</th>
<th>Paper</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>143</td>
<td><strong>Key Note Speaker:</strong> Software solutions for Spreadsheet Risk</td>
<td><strong>Phillip Howard,</strong> Bloor Research</td>
</tr>
<tr>
<td></td>
<td><strong>Session 5: Exorcising Evil: Debugging Spreadsheets</strong></td>
<td><strong>Chair: David Ward</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Voice-controlled Debugging of Spreadsheets</strong></td>
<td><strong>D. Flood,</strong> K. McDaid, Dundalk Institute of Technology, Ireland</td>
</tr>
<tr>
<td></td>
<td><strong>An Empirical Study of End-User Behaviour in Spreadsheet Error Detection &amp; Correction</strong></td>
<td><strong>B. Bishop,</strong> K. McDaid, Dundalk Institute of Technology, Ireland</td>
</tr>
<tr>
<td></td>
<td><strong>Inclusion Analysis: Finding omission errors</strong></td>
<td><strong>D. Colver,</strong> Operis, UK</td>
</tr>
<tr>
<td></td>
<td><strong>Session 6: Preventing Evil - Education and training</strong></td>
<td><strong>Chair: Pat Cleary</strong></td>
</tr>
<tr>
<td>191</td>
<td><strong>Why Task-Based Training is Superior to Traditional Training Methods</strong></td>
<td><strong>K. McGuire,</strong> Small Spark, UK</td>
</tr>
<tr>
<td>197</td>
<td><strong>Establishing a Minimum Generic Skill Set for Risk Management Teaching in a Spreadsheet Training Course</strong></td>
<td><strong>D. Chadwick,</strong> University of Greenwich, UK/Eusprig, UK</td>
</tr>
<tr>
<td>209</td>
<td><strong>Facing the Facts: Overconfidence and Quality</strong></td>
<td><strong>Patrick O’Beirne,</strong> Systems Modelling, Ireland</td>
</tr>
</tbody>
</table>
PREFACE

You are very welcome to the Eighth Annual Conference of the European Spreadsheet Risks Interest Group.

Our contributors this year are from Australia, Canada, Ireland, the UK, and the USA. This year’s theme is "Enterprise Spreadsheet Management – a Necessary Evil?" This theme reflects the greater grip of regulation and the equal and opposite reaction of corporations.

In its eight years so far, Eusprig has fostered the linking of academic research with the skills of practitioners. We are pleased to see this networking bearing fruit in ever more thorough and imaginative research. We commend the maturing of the market for spreadsheet discovery, assessment, remediation, and control. Visitors to our website eusprig.org have increased by 50% per year over the last years; currently we have more than 3000 unique visits per month.

I would like to acknowledge with gratitude our sponsors who have supported us financially and professionally. Our prime commercial sponsor this year is Compasssoft, along with SecureXLS. We depend also on the support of ISACA Northern England chapter, Baker Tilly, the University of Greenwich, the University of Wales Institute in Cardiff, Spreadsheet Engineering Ltd and Systems Modelling Ltd. Representatives of these organisations have contributed a great amount of expertise in the organising of this conference, the publicity, the proceedings, and much more work in the background.

It is my pleasure to once again acknowledge the untiring efforts of our three conference and programme organisers from the University of Wales Institute, Pat Cleary, Simon Thorne, and David Ball (recently taken over from our long-time treasurer Graham Macdonald), and David Chadwick of our host university of Greenwich. The committee also depends upon the wise counsel and active support of Ray Butler of ISACA Northern England Chapter, David Colver of Operis, Grenville Croll (PR and membership secretary), Roland Mittermeir of Klagenfurt University, Jocelyn Paine of Virtual Worlds, and David Ward of Baker Tilly.

The main programme now fills two days to overflowing.

Our keynote speakers are Dean Buckner of the UK Financial Services Authority, Paul Bach of Compasssoft, Ray Panko of the University of Hawaii. Enterprise Spreadsheet Management will also be addressed by ClusterSeven, Prodiance, and Codematic.

We continue with a focus on risk management, development and control methodologies, testing, and debugging. Other speakers will present their research into error impact, detection and correction, quality improvement, Education and Training. Thank you for your interest and participation, and we look forward to a stimulating and interactive conference!

Patrick O’Beirne, Chairman 2006/07, Eusprig
http://www.eusprig.org
http://groups.yahoo.com/group/eusprig
Regulatory Update

Dean Buckner
UK Financial Services Authority

July 2007

Summary

- As is now customary I shall review progress in the areas of compliance and spreadsheet risk management, over 2006-7
The issues …

- Change of mindset (industry, senior mgt, IT)
- User training
- No “good practice”
- Accreditation
- Audit awareness
- Data standards
- Spreadsheet support

Management mindset

- Spreadsheets are increasingly accepted as strategic solutions
  - A major change from five years
  - Many firms have given up on the ‘big solution’
- Risk magazine takes note
  - The house journal of quantitative finance
    - June 2007 issue
User training

- Most problems are still the result of poor use of EUC solutions
- Good training the obvious solution
- But little budget!
  - Some changes from last year
  - But I am still seeing ‘dumb solutions’ that training would have easily prevented
- A mitigating factor is the increasing use of spreadsheet control systems

Other mitigants

- More firms using spreadsheets in a way that makes sense
  - E.g. using the strong pivot functionality in Excel with back-end relational databases
  - Commerzbank switching data aggregation from spreadsheets to computing grid
  - More technical solutions that take the compliance burden away from the user
Good practice

- Little change from last year
  - Policy tends to be very high level
  - No industry view on good practice

Accreditation

- No change
  - Accreditation seen as burdensome and risky
  - And difficult (implies generally accepted view on good practice, for a start)
Audit

- Continues to improve
  - Increasing mention of spreadsheets in audit reports
  - General progress over 2003-6
  - And EUC part of audit plans

Data

- Data processing – spreadsheets now standard across the industry
- Increasing use of compliance tools
- Limitation on spreadsheet size was only constraint (but not any more!)
Microsoft

- Biggest development of 2007
- Dialogue between firms, Microsoft and FSA
- Attempts to address:
  - Code fragmentation problem
  - Lack of audit trail
  - Version control
- However, now 1m rows, 16k columns!

Eusprig

- Getting away from early obsession with ‘errors in spreadsheets’
- Needs to understand control environment in firms
ABSTRACT

This paper presents the arguments and supporting business metrics for Enterprise Spreadsheet Management to be seen as a necessary good. These arguments are divided into a summary of external business drivers that make it necessary and the ‘good’ that may be delivered to business spreadsheet users involved in repetitive manual processes.

1. INTRODUCTION

Four years ago, ClusterSeven was the first company to use the term “Enterprise Spreadsheet Management”. We noted that despite the value of spreadsheets to organisations in terms of flexibility, familiarity, speed and cost-efficiency there were notable business deficiencies when they were used as operational applications (i.e. where business practice creates version after version). These deficiencies include the absence of auditability and change control, the difficulty of managing and analysing historical data and the time spent on repetitive manual tasks. In selecting the term “Enterprise Spreadsheet Management” we set out to define a technology sector that addressed these deficiencies, without reducing spreadsheet functionality or performance.

Mark Twain said that ‘Work is a necessary evil to be avoided’. As to whether Enterprise Spreadsheet Management should be considered a necessary evil, I will tackle the argument in two steps; firstly to demonstrate that it is ‘necessary’ and secondly that it is a force for ‘good’.

2. NECESSARY OR UNNECESSARY?

While the calculation risk inherent in spreadsheets has been extensively discussed since Panko and Halverson (1996), their presence as a systemic part of the operational fabric of businesses has received less research. Croll (2005), Buckner (2004, 2005, 2006) and Pettifor (2005) are notable exceptions. All focus on the pervasive presence of operational spreadsheets and their inherent risks. Moreover, they highlight that businesses using operational spreadsheets typically apply much higher standards of risk and asset management in almost everything else they do – somehow spreadsheets have slipped through the management net. The following changes in business risk and its assessment have now brought this issue to a tipping point in perception.

1.1 Volume and value of spreadsheets.

In our experience any material financial services business will have a few hundred to a few thousand operational spreadsheets (excluding any version saves). They range in size from small (1MB or less) to very large (in excess of 150MB). The total volume of operational spreadsheets will commonly exceed 1GB. The most intensive departments (e.g. trading of ‘exotic’ derivative instruments) have more than 5GB in daily use. Typically each spreadsheet is used over and over again for many months and probably years. This creates multiple version saves with an aggregate data volume many times that of the operational spreadsheets.
This explains why files scans of servers for .xls files in large departments may locate hundreds of thousands and sometimes millions of spreadsheet files.

All of these operational spreadsheets have an impact on the business (as business assets in their own right, as retainers for intellectual property, as consumers of man hours and as a record of business input to decisions). These spreadsheets are often associated with some of the most innovatory and profitable parts of the business (Ashton, 2005). Despite this value, most businesses have no explicit recognition or management of this asset base. Ironically, this is probably because the activity is so prevalent that it has become an unremarkable and implicit part of business.

The spreadsheet asset base within a business clearly requires explicit identification and monitoring as much as any other information resource. For most organisations the scale of the challenge means that manual solutions will be impractical. Enterprise Spreadsheet Management provides a technological solution to the problem, enabling organisations to wrap a monitoring environment around this asset base establishing a basis for all the standard IT policies such as security, change control, disaster recovery and data management.

1.2 Tactical or Strategic?

The most common reason for businesses to reject the establishment of a formal enterprise spreadsheet management process is that operational spreadsheets are short-term tactical solutions. As such, there is little reason to invest in managing them, it being better to conserve funds for a future long term solution. This is an example of what Buckner (2003) termed the budget paradox.

As confirmation of this ‘denial’ it is interesting to note that we have found binary code dating from very early versions of Excel (even Excel 95) in current operational spreadsheets. This longevity suggests clear strategic value.

The reference by Buckner (2006) that some banks are stating that ‘spreadsheets are now accepted as strategic solutions’ changes this dynamic. The switch in perception from tactical to strategic is a key step in justifying corporate resources (in human, financial and technical terms) to be spent supporting Enterprise Spreadsheet Management.

1.3 The Expectations of Employees

Until ten years ago there was an unwritten assumption about computing power: an individual joining a big business would get far more capability inside a business than they would get at home. This has changed – an individual with domestic broadband has enough computing power to do almost anything they want. Now when they join an organisation they will commonly be offered reduced power (e.g. by restrictions on web usage, file downloads, email filtering and through locked-down desk top machines). Much of this restriction comes from concerns regarding security or liability.

The spreadsheet is another example of this end-user power. On the one hand it provides the ability for capable individuals to build enormously powerful models to exploit their understanding of a particular financial environment. At the same time there may be a large business exposure caused by the use of this freedom. Clearly the employer cannot remain ignorant of such exposure and, as with the other end-user applications, decisions must be made as to how it should be managed.

An example of this was the £13.9million fine levied on Citigroup (Conceicao, 2005) where the bank was penalised for pursuing a particular trading strategy. Whilst the use of the spreadsheet was not the reason for the fine, the fact that the bank’s technology (both trading...
systems and spreadsheets) was used to perpetrate these unapproved activities receives specific mention in the list of criticisms.

Enterprise Spreadsheet Management technology delivers an important option to managers faced with this decision – allowing users to retain the accustomed value of spreadsheet speed and flexibility, but monitor what they do.

1.4 Regulators and Personal Exposure

One of the greatest changes in the business world in the last decade is the rise in corporate regulation. The Sarbanes-Oxley (SOX) act in the US probably represents the fiercest element in a series of regulatory announcements. If SOX can be boiled down to one sentence it is that it removes the ‘I didn’t know what was going on’ defence for senior executives. When it comes to anything to do with corporate finances that means executives have to know where their numbers came from.

Modern centralised systems pass such transparency tests with ease – audit, security and integrity are part of the design specification. Change control is enforced through IT policies. But for spreadsheets it is a different matter. Who knows what has happened to a spreadsheet? If you are an executive facing incarceration for signing off on these items it becomes crucial that you know who did what. This pressure makes Enterprise Spreadsheet Management an executive necessity and has been the single greatest driver behind the growth in the sector.

All major financial institutions now maintain loss registers to support compliance with the Basel II legislation. Examination of these shows multiple losses (occasionally reaching seven figures) attributable to spreadsheet errors that would have been prevented had Enterprise Spreadsheet Management been in place.

Spreadsheet risk has also become more widely recognised in the general business community. In the last two years there have been regular mentions in the press including the Financial Times (2006), the Wall Street Journal (Gomes, 2006), and the Telegraph (Miller, 2005) as well as numerous trade journals (Laurent, 2006 and Albinus, 2006). This has been supplemented by work from Microsoft (2006) who specifically outline a file management approach based on the new version of their product Sharepoint (a file repository). In addition the IT analyst house Gartner have now formally recognised the emergence of a new technology sector called Spreadsheet Control (Heiser, 2006). They anticipate that mainstream adoption will be relatively fast, estimating a period of 2-5 years.

Any future executive defence based on ignorance of the issue is clearly becoming steadily more difficult.

1.5 What Other Solutions Exist?

The primary test of necessity is whether there are other solutions that could render a technology-based approach to enterprise spreadsheet management unnecessary. In our experience four alternatives are most commonly suggested:

i. **Forbid the use of business-critical spreadsheets.** Many companies have exactly this policy. This fails for two reasons; the confidence of users in their own capability (and the opportunity to deliver unique insight or make money) means that they do so anyway. Secondly, the transition from non-business critical to business-critical usage is not a sharp line; the point at which the policy is transgressed is unclear and hence difficult to enforce.

ii. **Manually document changes** e.g requesting users to self-document changes. The problem with this approach is the level of typical business spreadsheet usage. Furthermore, the desire for speed (that makes spreadsheets so attractive) is counter-
productive to manual documentation. Our own experience also shows that users
genuinely don’t remember all their changes as a ‘few’ changes always leads to a few
more. We have seen examples where a signed commitment to make four specific changes
have led to another forty – in the name of ‘tidying things up’.

iii. **Prevent changes** by locking the spreadsheet beyond input values. This can be achieved
by ‘compiling’ the spreadsheet into a rigid application or by applying cell and sheet
protection. This approach creates two user communities; those without the authority to
make changes (typically the beneficiaries of the application) and those with authority
(typically the maintainers of the application). When well regimented this process can
achieve a first level of control and Enterprise Spreadsheet Management can assist in
supporting the integrity of this environment. However, in practice it has two weaknesses:
firstly, that the pressure to make changes in time for business deadlines means the
authorised community gradually grows to include all the regular business users (thus
defeating the attempt to segregate the two communities) and secondly, it fails to confirm
that those authorised to make changes have done what was requested (and no more).

iv. **Replace operational spreadsheets** with formal applications. This should be the preferred
form of retirement for any mature spreadsheet application with proven business value.
However, as noted by PWC (2005) spreadsheets will always be utilised to fill the gap
between business needs and installed systems. The materiality of this gap is dependent on
many factors such as the rate of evolution of the business, the available IT budget to
invest in new systems and the certainty that the business being conducted in spreadsheets
merits migration. An example of the latter is that of weather derivatives. A few years ago
trading weather derivatives and credit derivatives were both promising new areas of
business. Today credit derivatives are everywhere and weather derivatives are rare.
Money spent migrating weather derivatives spreadsheets would have been wasted.
Replacement strategies therefore have a proper role in the termination of individual
spreadsheets, but they do not solve the underlying issue.

2. **GOOD OR EVIL?**

Section 1 focused on the environmental pressures to adopt Enterprise Spreadsheet
Management. So it may be necessary, but is it good? Since the perception of whether
something is ‘good’ is largely subjective, I will consider it from the value offered to a typical
business user (i.e. not a compliance or technical perspective).

My arguments for user-benefit from Enterprise Spreadsheet Management are derived from
seeing how spreadsheets are utilised on a day-to-day basis to support business processes.
Despite their speed and flexibility there are many usage aspects within a business
environment that are inefficient, leading to user frustration and wasted time. This area of
negative user experience represents an attractive target for Enterprise Spreadsheet
Management to prove itself a definite good.

2.1 **Version Management**

Many users will have corrupted a spreadsheet and found that the easiest solution is to revert to
an earlier copy. In an operational business environment, the problem is much greater. Firstly,
users are often working on a spreadsheet that they didn’t write – so it is easier to break and
more difficult to fix and, secondly, the version to be recovered may not be preserved in hot
storage – it often requires a lengthy turnaround for IT to retrieve it from a tape back-up.

With the automatic file and version management contained in an Enterprise Spreadsheet
Management solution much of this lost time and frustration can be eliminated. Authorised
users can have easy access to any past version through an intuitive timeline interface, without needing to rely on others.

Our own experience in clients suggests that the user time saved simply from provision of a user-friendly automated version recovery utility can justify a material proportion (circa 10%) of the overall costs of Enterprise Spreadsheet Management.

An additional benefit to this approach is that the pattern of operational usage by time and by user becomes apparent, giving an explicit presentation of existing workflow practices surrounding the spreadsheet.

2.2. Data Integrity

If the output values from a packaged application change unexpectedly the owner will immediately alert management and investigate the business cause. However, if the output value from an operational spreadsheet changes unexpectedly the first response is to check that the spreadsheet is functioning properly before raising any management alerts. This results in two costs: the overhead of constantly checking of spreadsheets, even when they are correct, plus the delay in addressing any underlying business issue.

A typical business example is the ‘product control’ or ‘middle office’ department of investment banks. These departments are essentially the business guardians of trading desk activity. They have the responsibility of validating trader activity and providing aggregated reports of financial positions. When a reconciliation fails they are the ones responsible for uncovering the root cause, be it real or just a spreadsheet error. With spreadsheets containing many tens of sheets and millions of cell values, the process of error discovery, location and repair consumes large amounts of employee time – with the approach essentially a lengthy and frustrating process of “spot the difference”.

Investment banks have quoted to us that 30 to 50% of their product controllers’ time may be involved in such processes. Enterprise Spreadsheet Management can resolve these questions in seconds rather than hours, removing frustrating aspects of spreadsheet usage and allowing users to focus on added value activities. It can do this by filtering out all of the changes that are part of normal business practice, allowing user to rapidly identify the needles in the haystack.

Enterprise Spreadsheet Management can take the analysis of change events to an additional level, offering richer insight into the business practices that are leading to problems. It allows messaging alerts to be set on cell values to rapidly provide management awareness of a potential problem. For example, an alert might be set to fire if a cell reaches the value of ‘50’, up from a previous value of ‘40’. On drilling down on this alert the full history of the cell contents will provide additional business context. For example the full data sequence may be 40, 40, 40, 50 or 40, 49, 40, 50 or 20, 30, 40, 50 or 49, 49, 40, 50. Each of these sequences will suggest different business issues that may need to be addressed.

2.3 Data Integration

Operational spreadsheets are not an end in themselves. They are usually part of a longer information supply chain. This re-use of spreadsheet information is achieved either by manual re-keying or by creating spreadsheet links or by direct extraction. All of these methods regularly fail, leading to wasted user hours. Manual re-keying is clearly the least robust – and also invalidates the protection of any previous data validation. Spreadsheet links are notoriously error-prone but are still the preferred way of extracting historical data from spreadsheets e.g. a year-end spreadsheet will link to those of January, February, March etc.
Direct extraction is probably the most robust solution but still subject to failure e.g. where users have entered data outside the expected range.

Enterprise Spreadsheet Management can replace these ad-hoc and non-robust solutions, bringing major user benefits in terms of time saved and uncertainty removed. By creating a centralised, structured and secure database of all validated information this information can be re-used with confidence, using robust, refreshable calls to a secure database. Most importantly, spreadsheet information only needs to be validated once.

2.4 Analysis of Historical Data

Successive version saves of an operational spreadsheet provide snapshots of the evolution of the business process that is being supported. Each one is ‘final’ in the sense that it was a true representation of the state of that part of the business at that time.

The changes in business parameters contained in the spreadsheet therefore offer a highly granular dataset that may be analysed (e.g. for trends, variance and covariance) leading to business insight and potentially new opportunity. Using traditional spreadsheet processes the only ways to access these historical datasets is to open each spreadsheet individually and extract the relevant values (probably into a new spreadsheet), to configure multiple links or to write a specific data extraction component. All of these are highly laborious and prone to error. It certainly does not encourage ad hoc investigation where the inspiration for new insight may be derived.

Enterprise Spreadsheet Management offers a granular time series of the value (and formula) history behind every cell, allowing historical analyses to be performed with ease, thus removing hours of manual activity and leaving user time to focus on the opportunity for added value.

2.5 Spreadsheet Retirement

An additional source of user frustration is the timing of spreadsheet retirement. Commonly it is attempted too early, whilst functionality is still immature. As a result the user is called into frequent specification and scope change discussions, damaging the relationship between business and IT.

Enterprise Spreadsheet Management can assist with this process. If the formula history is static, with spreadsheet changes limited only to data values then the spreadsheet is a good candidate for migration. Similarly the spreadsheet can be used as a good specification template. However, if formula change continues to be widespread then any attempt at migration is likely to be beset with scope change issues.

It is perhaps fitting that true Enterprise Spreadsheet Management should assist with eliminating its very requirement – although by then there will be new spreadsheets to manage.

3. CONCLUSIONS

Having been responsible for establishing the initial use of the phrase Enterprise Spreadsheet Management we have worked with our clients to establish the quantitative and qualitative returns outlined in this paper. These clients are concentrated in the financial services and commodities trading business environments, where spreadsheet usage is amongst the most intense and complex in the business world. For them Enterprise Spreadsheet Management is not an optional extra, it has become a business necessity. For end-users, the potential ‘evil’ of an all-seeing big brother can be turned into ‘good’ by eliminating manual tasks and facilitating other repetitive processes based around spreadsheets.
4. REFERENCES


Spreadsheet Hell

Simon Murphy
Codematic Limited
Carlisle
Cumbria
UK
Simon.murphy@codematic.net

ABSTRACT

This management paper looks at the real world issues faced by practitioners managing spreadsheets through the production phase of their life cycle. It draws on the commercial experience of several developers working with large corporations, either as employees or consultants or contractors. It provides commercial examples of some of the practicalities involved with spreadsheet use around the enterprise.

1 ACKNOWLEDGEMENTS

The author would like to thank all the contributors from the smurfonspreadsheets blog for sharing their commercial experiences.

2 INTRODUCTION

60% of large companies feel 'Spreadsheet Hell' describes their reliance on spreadsheets either completely or fairly well. The same survey noted spreadsheet use at 100% of all respondents, the only universal technology. (Durfee, 2004).

It’s pretty hard to overstate the importance of spreadsheets to modern business life. (Croll, 2005) found the City of London to be heavily dependent, with most respondents suggesting that spreadsheets were critical to the ongoing viability of the markets and by extension of the City itself.

So we have a completely business critical resource, perhaps like the corporate network or email, and yet in general there appears to be no identifiable person or body responsible for managing it. In many organisations the responsibility falls through the gap between the IT department and the business users. Or it did, Sarbanes Oxley raised the profile of what was once every organisations dirty little secret.

3 BACKGROUND

Here are some approximate timings of recent representative spreadsheet based projects undertaken by the contributors:

1. Development = 3 months, live so far 14 months
2. Development = 6 months, live so far 4 years
3. Development = 12 months, live so far 9 years
4. Development = 5 months, live so far 7 years

In all cases the development phase was less than 25% of the total live to date figure, in some cases it is less than 10%. And yet much of the documentation seems to focus on...
development rather than maintenance. There are very limited published resources to assist in effectively managing up to 90% of the life cycle.

In other business areas the balance could be completely different of course, in particular there are a significant proportion of single use models in some fields.

'Spreadsheet hell' is at least as much about poor management as it is poor quality development. In fact many practitioners feel the mismanagement of production spreadsheets to be the single most significant risk factor in using spreadsheets. One contributor coined the phrase 'versionitis' to describe the uncontrolled proliferation of spreadsheets that seems to occur under poor management conditions.

During a recent spreadsheet risk awareness training session carried out by the author:

- 60% of the delegates felt their Excel skills were inadequate for their job.
- 60% had less than the equivalent of 2 days Excel training.

A list of possible training options and work changes was rated, the top two options were:

- 45% rated expert desk side support as the most useful training option
- 36% felt that a little extra time to deliver would be the most useful change, and in many cases more useful than additional training.

This small group session can not be considered statistically relevant, but the findings do support anecdotal evidence. Any serious program to manage the spreadsheet resource would need to address this perceived time and skill shortfall.

4 SPREADSHEET HELL

The spreadsheet issue can be considered on two levels, micro and macro. Spreadsheet hell at the micro level refers to 'frankensheets' (Bruce, 2006). These are big, ugly spreadsheet monsters that are hard to understand, hard to use and hard to test. At the macro level, regardless of the quality (or lack of) of individual spreadsheets the way those spreadsheets are used, shared and replicated creates a whole other level of spreadsheet hell.

This paper covers both, but with more emphasis on the latter. Where possible the points raised are illustrated with a real world example from the contributors’ commercial experience.

4.1 Micro Level Spreadsheet Hell

An individual spreadsheet can earn itself the frankensheet title fairly easily. Indeed some spreadsheet builders have created nothing else for years. It would be easy to suggest an element of designed in job security were it not for the clear pain the original author experiences when trying to understand their own previous work.

At one company the whole monthly management reporting for 120 business units was driven by an Excel macro one of the consolidation team had recorded/cobbled together. This was a closely guarded treasure and outside interference was not welcome, even though this imposed a significant burden on its owner each period end, especially as it failed most months. As this was an 8 hour process that ran overnight, any failure meant all financial reporting was delayed at least a day. Often the total delay, during which the business had no knowledge of its recent performance was 2-3 days. Although rooted in the individual spreadsheet, this key man dependency has implications at the macro level too.
Several contributors explicitly mentioned this secretiveness and the inevitable key man dependency as key problems (and a common problem), not least because it almost guarantees poor or non-existent documentation. “Don't touch my baby” syndrome was how one contributor described it.

Any spreadsheet that is difficult to demonstrate as fit for purpose automatically contributes to spreadsheet hell. And many spreadsheets would need re-writing as part of that assurance process. In reality building a spreadsheet twice still probably leaves it cheaper and quicker than most of the alternatives, but is still pretty rare. In one regulatory reporting project the spreadsheet version cost £30k versus an estimated £500k in a specialised product, it could have been re-written over 10 times, and still been cheaper.

More common than multiple builds is to release the first version to business users after a cursory review and hope the users will spot and report any glaring errors. E.g. one data gathering template was issued with several important balance sheet codes missing, soon fixed once highlighted.

There is a mixed level of agreement on what represents best practice at the individual spreadsheet cell level. E.g. some people think range names are extremely valuable, others don't. However most practitioners would agree on the basic aims of best practice, as being to make understanding and testing reasonably straightforward.

Many factors can affect understandability of a spreadsheet, and some of these have been covered in some depth in previous Eusprig papers. Most practitioners will have favourite techniques, and features they avoid, usually based on good or bad personal experiences.

This paper makes no attempt to recommend any particular approach over another, instead it focuses on the wider management story.

4.2 Macro Level Spreadsheet Hell

Many organisations are now producing corporate spreadsheet development guidelines, and that is very worthwhile. However, very few seem to have invested in technical infrastructure, either to support the development or the production phase. For example very few spreadsheet developers use the development edition of Office, with integrated source control, or any other form of source control.

More and more organisations are using Office admin policies to restrict access to certain features, for example a recent client blocks all access to the Visual Basic for Applications editor. It is freely available on request, but not by default, thus allowing the organisation to control how certain features are used and by whom. Contrast with the default installation of Office 2000 that many organisations deployed, that had full access to VBA, but no VBA help. No wonder so many people got in a mess.

Ideally a program of spreadsheet management would include:

- policies on when to use, and when not to use, spreadsheets
- procedures for safe and effective development of valuable spreadsheets
- features to use and those to avoid, with justifications
- adequate training and coaching appropriate to job role
- procedures and policies for managing the modification of live systems
- policies for safely archiving retired spreadsheets
- Full consideration of all aspects of the systems life cycle
Anecdotal evidence suggests the amount of effort spent keeping a spreadsheet working during its lifetime is inversely related to the quality of the product arriving in the live environment. One rather badly implemented spreadsheet took 3 days to correct, where a well built one would never have gone wrong. At one client, one particular model requires approximately 15 developer days of effort to implement a report headings change, which happens each quarter, another model requires just 1 day to make the same changes.

Keeping these complex spreadsheet systems working is only part of the problem. Another significant issue is extracting and using the business insight locked up in these rigid structures. In one example a stockbroker had 400 workbooks analysing individual stocks. They then needed to summarise which stocks had Price/Earnings ratio below a certain level. Poorly designed spreadsheet systems do not encourage this sort of slice and dice analysis.

4.3 Spreadsheet use

There are two broad categories of ongoing spreadsheet use:

1. **Normal use** – for many reporting applications this is the generation of regular reports, also included in here are any year end roll overs.

2. **Changes** – reasons to make non 'normal use' changes to live workbooks will be discussed below

**Normal use**

Normal use can be a significant contributory factor in spreadsheet hell. Many organisations have limited document management tools, and therefore much essential information must be stored in the file path and name explaining what version the file is and where it is to be used. Many of these files are extremely similar, perhaps 90% of the content is the same reference data or prior period results.

One common file structure is to have a folder for each month and then store the appropriate months spreadsheets (with the same file name) in there each month. In Excel it is not possible to open 2 workbooks with the same name at the same time, even if they are from different folders, so this approach instantly causes reconciliation challenges. The files must be temporarily renamed so they can be compared, failure to reset the names will almost certainly break a linked consolidation report somewhere. As time goes on multiple copies of history proliferate. For example by December there are 12 copies of Januarys results. This makes back posting (changing results of a previously checked and closed period) incredibly easy, and really every months results should be checked each month. Indeed at one clients the February numbers reported in February, March, and April were all (wildly) different, leading to significant consternation amongst the users.

It should be clear that this structure leads to huge duplication, and massive proliferation of very similar spreadsheets, a significant management problem. One very specific risk increased by having many similar spreadsheets floating around is the risk of using the wrong version. This can be exacerbated by a poor or non-existent naming conventions, or unstable file locations. Use of some form of document management system could be a quick win here.

Another common driver of file structure is the use of external links. This feature is widely considered extremely dangerous, but it is such a fast way to build complex systems, it is the norm. In one extreme case a monthly main board pack (approx 20 interlinked workbooks) had to be stored in its own folder each period and never recalculate as it contained an external link circular reference which meant it produced a different number
each calculation. The only way to 'lock' in on the published numbers was to archive the reported version somewhere it would not get recalculated before starting the following months reporting.

It is important to remember that spreadsheets are usually the presentation layer of a long chain of data manipulations. Unfortunately this leaves them susceptible to problems introduced by changes anywhere in the chain. One contributor had to hide from an angry manager who had to recall a widely distributed set of reports. The cause was a data supplier unilaterally changing the structure of the data they supplied with no warning. Its no coincidence this example is mentioned in the normal use section, it’s a common problem.

**Changes**

There are several basic reasons to make non normal changes to a live workbook.

1. To add a feature
2. To fix an error
3. To improve the design
4. To improve performance
5. To update embedded reference data

In all cases if multiple copies of the spreadsheet are scattered around the network or the world, then coordination and consistency will be a challenge. Very few inexperienced developers make version information obvious enough.

One very real problem with making changes to a live model is the high chance of unexpected side effects. Again, to an extent this is a factor of the spreadsheets underlying quality. Spreadsheets, especially where external links are used, are notorious for breaking changes. Fear of side effects can cause maintainers to contort the existing model rather than make simplifying changes. A simple example is converting a fairly simple formula to a complex array formula for fear of inserting a column for the intermediate calculations. The array approach is recommended regularly, and used regularly even though it is known to degrade performance and maintainability for the future.

VBA adds a whole new dimension to these problems, especially the generally poor quality code seen in many Excel applications. Some code will even overwrite corrections with incorrect values or formulas, often when least expected.

5 Conclusion

Commercial use of spreadsheets raises issues well beyond the quality of individual models. The overall process of managing the use of this critical resource can have a dramatic effect on the risks to which an organisation is exposed and the value it can leverage from its investments.

6 References


7 CONTRIBUTORS

Thanks to R. Bruce, H. Grove, R. McLean, M. Syben, D. Wallentin, and to those who gave their advice and feedback anonymously.
Managing Critical Spreadsheets in a Compliant Environment

Soheil Saadat
President and CEO
Prodiance Corporation
5000 Executive Parkway, Suite 270
San Ramon, CA 94583 – USA
soheil.saadat@prodiance.com

Spreadsheets – The Hidden Risks

The use of uncontrolled financial spreadsheets can expose organizations to unacceptable business and compliance risks, including errors in the financial reporting process, spreadsheet misuse and fraud, or even significant operational errors. These errors have been well documented and thoroughly researched by the European Spreadsheet Risks Interest Group [EuSpRIG, 2005]. With the advent of regulatory mandates such as SOX 404 and FDICIA in the U.S., and MiFID, Basel II and Combined Code in the UK, leading tax and audit firms are now recommending organizations automate internal controls over critical spreadsheets and other end-use computing applications, including Microsoft Access databases. At a minimum, auditors mandate version control, change control and access control for operational spreadsheets, and more advanced controls for critical financial spreadsheets [PwC, 2004]. While regulatory compliance has remained a key business driver, many organizations are implementing spreadsheet controls to manage operational risk, and to achieve sound corporate governance and process improvements.

The inherent complexities of operational and financial spreadsheets expose technological shortcomings of available spreadsheet management solutions. Specifically, financial spreadsheets often contain external links to other spreadsheets and databases. For example, a consolidated revenue spreadsheet may contain inbound links from individual product revenue reports, and outbound links providing results to executive dashboards or the overall balance sheet. Often these critical spreadsheets reside in employee desktops, in email attachments, or on corporate shared drives – an uncontrolled environment that is absent traditional IT controls. As such, security over these critical spreadsheets tends to be weak, access is often not controlled, file versioning is not implemented, there is no visibility into changes being made, nor validation that external links are correct.

The bottom line is critical business decisions are being made everyday based on data produced by critical spreadsheets, yet executives have little confidence or trust in the data being produced in uncontrolled environments.

Spreadsheet Links Create a Technology Challenge

Auditors and executives alike agree that the right solution to address these challenges is “to move to an automated, controlled, yet flexible technology-based environment” [Ernst & Young, 2007]. Centralizing spreadsheet control creates a new system of record for all critical spreadsheets, and enables organizations to apply auditor recommended IT controls such as versioning, security and access control, records retention, archival and backup, change control and workflow automation [PwC, 2004]. However, simply moving spreadsheets into a
document management system through traditional methods often breaks the links, and this requires many additional man hours to re-establish the links. Without the use of technology, this is a labor-intensive and manual process, and the lack of visibility into spreadsheet links and lack of documentation compounds the problem. Notwithstanding, today’s commercial document management systems are not designed to work seamlessly with Microsoft Office Excel to preserve and update spreadsheet links. If a spreadsheet is moved into a document management system, or even to another network file location, the links will break. Many companies have tried migrating critical spreadsheets into document management systems, only to have exasperated end users who cannot update their spreadsheet input data through the resulting broken links. These projects have failed miserably, leaving IT project managers, auditors and financial executives to look for alternatives.

**Solution for Managing Linked Spreadsheets**

Fortunately, for these technology and business challenges there is a solution. A proven approach to efficiently managing linked spreadsheets in a controlled and compliant environment:

- Automates spreadsheet discovery, documentation and risk analysis, including the creation dependency diagrams to provide visibility into existing spreadsheet links.
- Provides tools for the migration of critical spreadsheets into secure, document management repository while automatically re-establishing any and all links (to their new web folder location, e.g. http://sharepoint/).
- Generates a migration or inventory log of spreadsheets migrated and any changes to spreadsheet links.
- Incorporates a technology integration layer enabling leading document management systems (e.g. SharePoint) to automatically update real-time data feeds through spreadsheet links.
- Incorporates auditing of spreadsheet changes down to the cell level to satisfy change control requirements.
- Automates spreadsheet change request, testing, review and approval processes via workflow for both developers and end users.

**Automating Discovery, Documentation, and Risk Analysis**

To help organizations automate spreadsheet inventory efforts, discovery tools can be leveraged to search across a wide variety of data sources and report on spreadsheets and other end-user applications (including Access databases) that are being used within an organization [Protiviti, 2006]. Through a consolidated interface, users can search and generate an inventory report on spreadsheets meeting generic or custom search criteria (e.g. all spreadsheets where “Date Last Saved” equals “2006” or “2007” would represent spreadsheets last saved during the past year, or all spreadsheets where “Risk” equals “High” or “Medium”).

Spreadsheet analysis tools can perform a risk-based analysis (based on complexity and materiality) while automatically generating documentation about critical spreadsheets. For example a cell and formula diagnostic report can show formulas with errors conditions, uncover very hidden worksheets, invisible cells, inconsistent formulas, and a whole host of other key areas of risk. Inventory reports can also be generated listing all critical spreadsheets (and their dependents) along with a host of documentation, including date created, date last modified, owner, location, number of external links, number of worksheets, number of formulas, and many other criteria to show complexity. Spreadsheet experts often refer to this process as a model audit (or analysis of the correctness) of spreadsheets [Croll, 2007].
**Link Migration Tools**

Once relevant spreadsheets have been discovered, they should be migrated from uncontrolled desktops and shared drives into a secure, web repository. As mentioned above, this can be a challenging and time consuming task given the alternative of manual copy/paste operations and manually reestablishing links to dependant spreadsheets. An automated and proven approach requires a migration tool that automatically updates any links based on the new repository location of the spreadsheets. By moving spreadsheets into a secure, web based document management repository, a host of features and controls such as improved security and access control, versioning, check-in/check-out, records retention, workflow automation, and file level audit trails are available.

**Support for Leading Document Management Systems**

An additional requirement includes support for leading 3rd party document management systems, including Microsoft Office SharePoint Services and SharePoint Server. Most organizations already have these technologies in place, and will want to leverage their investments. As such, spreadsheet control solutions should support leading document management repositories via the WebDAV protocol, which allows the repository to expose itself as a network drive letter and provides a seamless end user experience. For example, a user opening a controlled spreadsheet from a SharePoint repository could simply use the standard File > Open dialog from within Excel to open a critical spreadsheet, check it out, makes changes, and then use the standard File > Save dialog to save the changes, check the file in, and then automatically submit it into a review and approval workflow process where electronic signatures are captured. It is through WebDAV that Excel can automatically and successfully update spreadsheet links with 3rd party document management repositories.

With this approach, the impact to end users is minimized, as is the impact to existing business processes. In addition, having no software requirements for client computers insulates end users from the complexities of the technology, minimizes training requirements, and reduces IT support.

**Change Management**

Following auditor guidance, there are two aspects of change management that are required for critical spreadsheets, a detailed audit history of changes down to the cell level, and an automated workflow process to enforce the requesting, incorporating, reviewing and approving of all changes. Spreadsheet control solutions can capture changes to data, formulas, macros, queries, and also report on row and column insertions/deletions and alert users via email on any these changes. This provides an extensive database for management reporting. Combined with automated workflows, spreadsheets can be routed for review, validation and approval to enforce corporate change management policies and to ensure that financial spreadsheets are appropriately reviewed during the quarter and year-end close process. [Panko and Ordway, 2008]

**Business Benefits**

By incorporating the technology capabilities described in this paper, organizations can take a proactive approach to automating the spreadsheet compliance lifecycle, from discovery and inventory, to risk analysis, management and automation or key financial workflows. Whether the need is being driven by the need to satisfy regulatory compliance mandates, improve existing business processes, or to better manage operational risk, controls over critical spreadsheets can be automated to help restore confidence and trust in key financial data.
analyzed and reported in spreadsheets. Business benefits realized to date by organizations successfully automating spreadsheet controls include:

1. Improved visibility into end user computing environment (e.g. spreadsheets and Access databases) via management and change reports
2. Improved compliance with regulatory mandates, including SOX 404, FDICIA, MiFID, Basel II and Combined Code that satisfies auditor scrutiny
3. Improved internal controls via technology automation
4. Improved financial processes via workflow automation
5. Improved spreadsheet development and use
6. Improved productivity for end users

References:


Risk Management for Complex Calculations: 
EuSpRIG Best Practices in Hybrid Applications

Deborah Cernauskas 
dcernauskas@i4mt.org

Andrew Kumiega 
kumiega@comcast.net

Ben Van Vliet 
bvanvliet@i4mt.org

Abstract

As the need for advanced, interactive mathematical models has increased, user/programmers are increasingly choosing the MatLab® scripting language over spreadsheets. However, applications developed in these tools have high error risk, and no best practices exist. We recommend that advanced, highly mathematical applications incorporate these tools with spreadsheets into hybrid applications, where developers can apply EuSpRIG best practices. Development of hybrid applications can reduce the potential for errors, shorten development time, and enable higher level operations. We believe that hybrid applications are the future and over the course of this paper, we apply and extend spreadsheet best practices to reduce or prevent risks in hybrid Excel/MatLab® applications.

I. Introduction

The spreadsheet is an enabling technology. Spreadsheets, based on Dan Bricklin’s VisiCalc, enable user/programmers to organize, collect, store, analyze, perform calculations and report data. With spreadsheets, users need not be computer scientists to develop sophisticated algorithms in code, freeing them to focus on their business strategies and quantifiable algorithms.

To keep up with the demand for more and more sophisticated analysis, spreadsheet capability has evolved greatly since the VisiCalc and Lotus 1-2-3 days. As the capability of the spreadsheet has grown, so too have the sizes of spreadsheet applications, the complexities of calculations and, therefore, the probability of errors.

Material errors in spreadsheets arise often because user/programmers are not well-versed in proper programming practices, despite Boehm’s findings that good architectural practices can reduce the cost of program implementation by avoiding defects (Boehm, 2001). Raftensperger (2001) estimates that 90% of spreadsheets have errors with consequences ranging from mild to severe. Studies by Davies and Ikin (1987) and Cragg and King (1993) found very informal spreadsheet development processes, most of which persist today.
There are several proven techniques that can reduce errors in and shorten development time of spreadsheets. Many of these are best practices documented within the EuSpRIG sphere. For operations beyond the capability of the spreadsheet, users can incorporate VBA code or third-party XLLs. Best programming and testing practices can apply to the coding of VBA macros and function libraries, as well as XLL libraries.

Creating new VBA functions promotes reusability, but VBA is no panacea. Here are two examples. Consider creating a Black-Scholes option pricing model or q-q plot (a quantile-quantile (q-q) plot is a graphical technique for evaluating whether two sets of data come from populations with the same distribution) in VBA code. A Black-Scholes function requires fifteen to twenty lines of code; a q-q plot substantially more. Although VBA has increased the functionality of the spreadsheet, it may not reduce development time, nor does it necessarily reduce the probability of errors. Today’s users demand more.

II. MathWorks’ MatLab

In more and more industries, the level of quantitative inquiry is passing by the technology that originally enabled it. For example, Excel can only handle matrices up to 52x52 in size. For larger matrix algorithms and greater capability, engineers are increasingly using MathWorks’ MatLab software (which incidentally also traces its origins back to VisiCalc) for complex calculations. MatLab is a high-level language and interactive development environment that enables users to perform computationally intensive tasks. These tasks can range from financial modeling, to computational biology, to simulation and optimization, to signal processing and control system design. According to the MathWorks website, MatLab is “used today by more than 500,000 engineers and scientists and by more than 2,000 financial companies worldwide.” Interestingly, the website goes on to explain that “professionals rely on MatLab to reduce development time, minimize costs and risks, and integrate new models.” The same could be said for spreadsheets. Conspicuous in their absence, however, are any documented best practices or proofs to support MathWorks’ claim of risk reduction.

Like the spreadsheet, MatLab is an enabling technology. Like spreadsheets, MatLab applications are also subject to risk of errors and fraud. Like enhancers of spreadsheet capability such as VBA or XLLs, though, MatLab is no panacea. MatLab has risks, including:

- Difficulty in testing
- Interim calculations and results due to a complex data interface.
- Lack of top to bottom, left to right structure for calculations.
- Ability to build applications interactively by manipulating arrays through a simple command-line structure. This produces code that can ramble as sample code gets copied and pasted into executable code.
- Lack of audited calculations and workflow due to a lack of data structures and documentation outside of the code.
- Absence of best practice development processes (this is very similar to Excel prior to EuSpRIG).
These risks do not (or no longer should) exist in spreadsheets. Which is to point out the following: if MatLab’s capabilities could be accessed through a spreadsheet, then EuSpRIG best practices could control these risks. This is now possible.

Through MatLab’s new Excel Link users can integrate MatLab’s mathematical and graphical capabilities into an Excel spreadsheet. Excel Link is a giant leap forward in facilitating the use of spreadsheets for complex calculations with efficiency, transparency and error reduction. Excel Link allows spreadsheet users to harness the power of MatLab’s extensive list of functions for modeling, statistics, and graphing in spreadsheet cells and VBA modules. Excel Link should be an equally large leap forward for MatLab users by providing their applications with a well-defined and easily auditable data interface. Excel Link allows MatLab users to use Excel for the data storage and presentation and MatLab for complex calculations. For example, implementing a Black-Scholes option pricing model or a q-q plot requires a single function call in MatLab, much simpler and more error-proof than creating VBA code. Excel Link allows spreadsheet capability to grow exponentially. At the same time, accessing MatLab through Excel also allows for control of MatLab risks through application of EuSpRIG best practices.

Once the prototype is complete, the MatLab code can be compiled into a .dll by using MatLab Builder for Excel. The .dll then ensures a high quality application with secure calculation code.

![Figure 1: Excel and MatLab](source: www.MathWorks.com)

To generalize, incorporating a commercial-off-the-shelf (COTS) component, such as MatLab, expands the complexity of analytical work that can be done in spreadsheets. Through COTS components, users can implement calculations once beyond the ability of the spreadsheet, while controlling risks inherent in the COTS development environment within a EuSpRIG best practices framework.

### III. Complex Calculations in Banking

CODA, a financial intelligence firm, reports that 95% of U.S. corporations use spreadsheets in financial reporting. For American financial institutions, the enactment of
Sarbanes-Oxley in 2002 in addition to the recommendations of Basel II (Basel II’s three pillars promote stability of the international financial system through minimum capital requirements, supervisory review and market discipline) has forced firms to look at how spreadsheets are used in financial reporting systems and to develop methods and procedures to reduce errors and increase security.

The same rules also apply to MatLab applications used in finance, since these applications regularly calculate profit and loss and risk reserves. Fraud or material errors in these spreadsheets can have significant consequences for such firms and their employees, including adverse audit opinions; reductions in stock price; and the possibility of fines and even incarceration.

IV. COTS Components: A Method of Defect Reduction & Prevention

The use of high level mathematics for derivatives pricing and forecasting of market, interest rate and credit risk, is forcing financial engineers to move beyond spreadsheets to more robust, rapid prototyping calculation engines available in COTS software. (COTS applications allow a user to quickly perform complex calculations faster than full development in C++. This is why MatLab is the tool of choice beyond Excel.) Implementing tested, COTS components reduces risk.

Software quality assurance should focus on preventive measures rather than testing. An alternative method to reduce and prevent defect is the use of re-usable software components, which accelerate development and are a big step towards achieving better, faster, cheaper software.

A recent study by Mohagheghi, et al. found that reusable software components have a lower rate of defects given the size of the code than non-reusable components. The same study found that non-reusable components have more errors which are correlated with the amount of the code.

COTS components, like MatLab have several advantages over non-reusable VBA components mainly due to the larger pre-built and tested functions (See Table 2). These include, but are not limited to, fewer defects, lower development costs, and shortened development time.

V. MatLab and Excel: Similarities & Differences

In Excel, shielding users from analytical complexities, is done through user controlled buttons, spinners and macros. Macros can also be coded in MatLab, which allows for more complex calculations often used in financial applications.

<table>
<thead>
<tr>
<th>Similarities</th>
<th>Excel</th>
<th>MatLab</th>
</tr>
</thead>
<tbody>
<tr>
<td>No database. Programs use flat files</td>
<td>Same</td>
<td>Interactive, but cumbersome</td>
</tr>
</tbody>
</table>
No versioning, i.e. most spreadsheets are single use, custom analytical tools.  

No versioning, i.e. most MatLab programs are single use, custom analytical tools.

Does not have a nice GUI, but one can be built.  

Same

<table>
<thead>
<tr>
<th><strong>Differences</strong></th>
<th><strong>Similarities between Excel and MatLab</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Can create a nice audit trail to show intermediate steps in a calculation.</td>
<td>The capability of the spreadsheet environment is expanded exponentially through Excel Link to MatLab. The MatLab software suite includes a variety of specialized functions in toolboxes such as financial time series, financial functions, and garch modeling.</td>
</tr>
<tr>
<td>Matrix computations are limited to a size of 52 x 52.</td>
<td>The amount of re-usable code, i.e. pre-built functions, for math and finance is substantially greater in MatLab than in Excel. This is shown by the number of pre-built functions as listed:</td>
</tr>
<tr>
<td>Regression analysis is easy to use but has limited statistics on equation fit.</td>
<td></td>
</tr>
<tr>
<td>Time series analysis is severely limited due to matrix limitations.</td>
<td></td>
</tr>
<tr>
<td>Ability to create visual tools and dashboards for analysis and monitoring.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Business Use</strong></th>
<th><strong>Excel</strong></th>
<th><strong>MatLab</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimazation</td>
<td>Solver</td>
<td>Optimization Tool Box</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>Solver Pro</td>
<td>Genetic Algorithm and Direct Search Tool Box</td>
</tr>
<tr>
<td>Time Series</td>
<td>No pre-built functions</td>
<td>Signal Processing Tool Box Neural Network Tool Box Model Predictive Tool Box Wavelet Tool Box Garch Tool Box for Forecasting</td>
</tr>
<tr>
<td>Control Analysis</td>
<td>No pre-built functions</td>
<td>Control System Tool Box Fuzzy Logic Tool Box Robust Control Tool Box</td>
</tr>
<tr>
<td>Financial Calculations</td>
<td>Basic financial functions</td>
<td>Financial Tool Box Financial Derivative Tool Box Fixed Income Tool Box</td>
</tr>
</tbody>
</table>
VI. Incorporating MatLab into a Spreadsheet

Employing MatLab commands in the Excel environment is straightforward since MatLab installs as a toolbar into Excel as shown in Figure 2.

All MatLab functions, commands and programs can be called using the MLEvalString command. Furthermore, once the MatLab code is completed it can be turned into a VBA module or a .dll file, via MatLab Builder for Excel. This ensures that the MatLab calculations can be locked and distributed so that end users cannot modify the algorithms. Figure 3 shows the Excel calls to access MatLab functions.
Figure 3: MatLab Link Management Functions and Data Management Functions

Excel Link provides four link management functions to initialize, start, and stop Excel Link and MatLab.

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>matlabinit</td>
<td>Initialize Excel Link and start MATLAB process.</td>
</tr>
<tr>
<td>matlabstart</td>
<td>Automatically start MATLAB process.</td>
</tr>
<tr>
<td>matlabstop</td>
<td>Terminate MATLAB process.</td>
</tr>
<tr>
<td>matlabopen</td>
<td>Start MATLAB process.</td>
</tr>
</tbody>
</table>

You can invoke any link management function except `matlabstart` as a worksheet cell formula or in a macro. You invoke `matlabinit` from the Excel `Tools Macro` menu or in a macro subroutine.

Use `matlabinit` to trigger automatic startup. If you install and configure Excel Link according to the default instructions, Excel Link and MATLAB automatically start every time you start Excel. If you choose manual startup, use `matlabinit` to initialize Excel Link and start MATLAB.

Use `matlabstop` to stop MATLAB without stopping Excel, and use `matlabopen` or `matlabinit` to restart MATLAB in the same Excel session.

Data Management Functions

Excel Link provides the following data management functions to copy data between Excel and MATLAB and to execute MATLAB commands from Excel.

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>matlabcell</td>
<td>Evaluate MATLAB command given Excel data.</td>
</tr>
<tr>
<td>matlabeval</td>
<td>Evaluate MATLAB command given Excel data and designate output location.</td>
</tr>
<tr>
<td>createMatlabMatrix</td>
<td>Create or append MATLAB matrix with data from Excel worksheet.</td>
</tr>
<tr>
<td>deleteMatlabMatrix</td>
<td>Delete MATLAB matrix.</td>
</tr>
<tr>
<td>evalMatlabCommand</td>
<td>Evaluate command in MATLAB.</td>
</tr>
<tr>
<td>getMatlabFig</td>
<td>Import current MATLAB figure into Excel spreadsheet.</td>
</tr>
<tr>
<td>getMatlabAttr</td>
<td>Write contents of MATLAB matrix in Excel worksheet.</td>
</tr>
<tr>
<td>getMatlabVar</td>
<td>Write contents of MATLAB matrix in Excel VBA variable.</td>
</tr>
<tr>
<td>getMatlabMatrix</td>
<td>Create or overwrite MATLAB matrix with data from Excel worksheet.</td>
</tr>
<tr>
<td>putMatlabVar</td>
<td>Create or overwrite MATLAB matrix with data from Excel VBA variable.</td>
</tr>
<tr>
<td>getMatlabErrors</td>
<td>Used by <code>matlabinit</code> to return standard Excel Link errors or full MATLAB errors.</td>
</tr>
<tr>
<td>setMatlabDir</td>
<td>Specify current working directory of MATLAB after startup.</td>
</tr>
<tr>
<td>fullMatlabWindow</td>
<td>Specify whether to use full MATLAB desktop or only Command window.</td>
</tr>
</tbody>
</table>

You can invoke any data management function except `putMatlabVar` as a worksheet cell formula or in a macro. You can invoke `matlabopen` only in a macro.

Figure 4: Applying MatLab functions to Excel data

Figure 3: MatLab Link Management Functions and Data Management Functions

Figure 4 shows two columns of data in an Excel sheet. There are several ways to put this data into MatLab, but the most straightforward way is to use the “matlabsub” function. In cell D6, the following MatLab command is entered:

```
matlabsub("mean","E6",A4:A1003")
```
The first argument to the function is the actual MatLab function that will be executed; the second argument is the starting cell location for the function output; and the third argument gives the cell locations of the input data).

Alternatively, the Excel data can be put into a MatLab matrix by using the MLPutMatrix function, which has two arguments: the MatLab variable name and the Excel cell address or range name. In Figure 5, the input data in cells A4:B1003 are assigned to a MatLab matrix called data through the named cell range DATA. The command in cell D15 gives the MatLab name “x” to the data in cells A4:A1003. The command in cell D17 calculates the mean of x and the command in D19 writes out the mean of x to cell E19.

Obviously, the power of MatLab commands within the Excel environment comes from the ability to execute the more complex MatLab commands.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>34</td>
<td>0</td>
<td>1.96</td>
<td>matlabsub(&quot;mean&quot;,&quot;E6&quot;,A4:A1003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>29</td>
<td>0</td>
<td>matlabsub(&quot;cov&quot;,&quot;x&quot;,A4:B1003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>23</td>
<td></td>
<td>0.032432</td>
<td>0.191071</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>20</td>
<td></td>
<td>0.191071</td>
<td>22.33575</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>26</td>
<td>0</td>
<td>22.33575</td>
<td>matlabsub(&quot;var&quot;,&quot;E11&quot;,B4:B1003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>16</td>
<td>0</td>
<td>MLPutMatrix(&quot;data&quot;,DATA)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>31</td>
<td>0</td>
<td>MLPevalString(&quot;x=data(:,1)&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>MLPevalString(&quot;y=mean(x')&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>28</td>
<td>0</td>
<td>1.96</td>
<td>MLPGetMatrix(&quot;m&quot;,E19)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: Alternative MatLab functions

Figure 6 illustrates an example provided by MatLab with the Excel Link add-in. In this example, the returns over time for three mutual funds is provided in cells B4:D9. In A15, the MLPutMatrix function assigns the column labels from cells F3 through G3 into the array named “Labels.”
Figure 6: MatLab Excel Link example

The next MLPutMatrix function call in cell A16 assigns the data in cells B4 through D9 into the matrix named "retseries". The command in cell A19 calculates the expected return and covariance matrix from the return series.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Portfolio Efficient Frontier</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Rates of return</td>
<td>Global</td>
<td>Corp Bond</td>
<td>Small Cap</td>
<td>Risk</td>
<td>ROR</td>
<td>Global</td>
<td>Corp Bond</td>
<td>Small Cap</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Nov-91</td>
<td>7.125%</td>
<td>4.125%</td>
<td>9.375%</td>
<td>0.769%</td>
<td>5.841%</td>
<td>0.56%</td>
<td>88.1%</td>
<td>3.6%</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Nov-92</td>
<td>5.125%</td>
<td>5.125%</td>
<td>3.875%</td>
<td>0.769%</td>
<td>5.725%</td>
<td>0.60%</td>
<td>85.7%</td>
<td>3.6%</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Nov-93</td>
<td>-1.875%</td>
<td>6.750%</td>
<td>10.562%</td>
<td>0.844%</td>
<td>5.803%</td>
<td>0.77%</td>
<td>83.0%</td>
<td>6.3%</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Nov-94</td>
<td>7.750%</td>
<td>6.062%</td>
<td>14.750%</td>
<td>0.834%</td>
<td>5.863%</td>
<td>1.13%</td>
<td>79.9%</td>
<td>11.0%</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Nov-95</td>
<td>5.250%</td>
<td>6.375%</td>
<td>3.625%</td>
<td>1.119%</td>
<td>6.961%</td>
<td>1.14%</td>
<td>71.9%</td>
<td>14.6%</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Nov-96</td>
<td>12.625%</td>
<td>6.125%</td>
<td>9.125%</td>
<td>1.267%</td>
<td>7.044%</td>
<td>1.87%</td>
<td>84.0%</td>
<td>17.5%</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Excel Link Functions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1. Transfer data to MATLAB.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0 = MLPutMatrix(&quot;Labels&quot;, F3:G3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0 = MLPutMatrix(&quot;retseries&quot;, B4:D9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2. Execute MATLAB Financial Toolbox functions.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0 = MATRIX([ret, cov = covstats(retseries)])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0 = MLPutMatrix(&quot;risk&quot;, [risk, ror, weights] = portopt(ret, cov, 20))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>3. Transfer output data to Excel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0 = MLPutMatrix(&quot;matrix&quot;,&quot;h4&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>4. Plot efficient frontier data and label the figure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>0 = MILineGraph(&quot;source.xlsx&quot;,&quot;portfolio&quot;,&quot;labels&quot;,&quot;labels1&quot;,&quot;labels2&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7: Portfolio Efficient Frontier Output

The command in cell A20 calculates the mean variance efficient frontier using the vector of expected returns and a specified variance-covariance matrix. The third argument is:

[risk,ror,weights] = portopt(ret,cov,20)
The portopt function is the number of portfolios generated along the efficient frontier. The function generates three outputs: portfolio risk (“risk”); portfolio return (“ror”); and the portfolio weights (“weights”). This should be compared to the standard Excel method as described in *Financial Modeling* by Simon Benninga. Benninga’s Excel-based method requires eight matrix multiplications, three matrix inversions, and five mathematical calculations that normally fill an entire Excel sheet. These calculations are complex so errors would be easy to make. Add to this, the limitation of only having 52 stocks, which makes Excel not able to perform this calculation in the real world.

The MLGetMatrix commands in cells A23 through A25 write out the portfolio optimization output starting in cells “F4”, “G4”, and “H4”. The final MatLab command in cell A28 plots the output of the portfolio optimization. The graph appears as Figure 8.

![Portfolio Optimization Graphical Output](image)

**VII. Conclusion**

Spreadsheets are known to be a great productivity tool for data analysis, model building and reporting for non-programmers. When Excel is used to consolidate financial data, price derivatives or develop financial pro-formas, the use of a proper software design methodology should be employed. There are advantages and disadvantages to using the spreadsheet environment to perform complex calculations. While a spreadsheet is slower than a dedicated software package, a spreadsheet provides the benefit of transparency, allowing for easier verification and validation. But, spreadsheet capability is limited relative to the higher level analytics demanded by users.

For more complex calculations, the robust libraries offered by COTS, such as MatLab, increases spreadsheet capability. Software re-use through COTS components greatly extends the functionality of spreadsheets and at the same time helps achieve better,
faster, cheaper spreadsheet development. But, these components lack the intuitive user
development environment, transparency, and furthermore, best practices to control risks.
By creating hybrid applications, EuSpRIG controls flow through to the MatLab code.

While the focus of this paper has been on incorporating MatLab components into
a spreadsheet, we believe that EuSpRIG best practices can be applied to a variety of
hybrid spreadsheet/COTS applications to control risks, including those identified by
Sarbanes-Oxley and Basel II, arising from complex calculations that were previously
beyond the capability of spreadsheets alone.

Furthermore, we believe that EuSpRIG best practices can and should be extended
to stand alone MatLab applications as well, since the cause of errors and the error
prevention techniques are similar. The same group of user/programmers are creating
complex calculations using higher level tools.

Bibliography


Management*. 135-137.

Bewig, Philip, 2005.  *How do you know your spreadsheet is right?*

Markowitz, H.M.  1952.  “Portfolio Selection.”  *Journal of Finance* (March):  77-91

Mohagheghi, P., Conradi, R., Killi, Ole, and Schwarz, H.  *An Empirical Study of Software
Reuse vs. Defect Density and Stability*.


European Symposium on Spreadsheet Risks*, Amsterdam, 61-76.

Conditions of Risk”  *Journal of Finance 19*. (September) 425-442.
Categorisation of Spreadsheet Use within Organisations: A Progress Report

Mukul Madahar, Pat Cleary, David Ball
Cardiff School of Management
University of Wales Institute Cardiff
Colchester Avenue
Cardiff, CF23 9XR
mmadahar@uwic.ac.uk, pmcleary@uwic.ac.uk, dball@uwic.ac.uk

ABSTRACT:

There has been a significant amount of research into spreadsheets over the last two decades. Errors in spreadsheets are well documented. Once used mainly for simple functions such as logging, tracking and totalling information, spreadsheets with enhanced formulas are being used for complex calculative models. There are many software packages and tools which assist in detecting errors within spreadsheets. There has been very little evidence of investigation into the spreadsheet risks associated with the main stream operations within an organisation. This study is a part of the investigation into the means of mitigating risks associated with spreadsheet use within organisations. In this paper the authors present and analyse three proposed models for categorisation of spreadsheet use and the level of risks involved. The models are analysed in the light of current knowledge and the general risks associated with organisations.

1. INTRODUCTION:

Many organisations rely on spreadsheets as a key tool in their financial reporting and operational processes (Price Waterhouse Coopers (PWC), 2004). Spreadsheets are becoming a significant part of the organisational decision-making framework and there is ample evidence that spreadsheets are erroneous. Panko and Nicholas (2005) highlighted the main studies undertaken to analyse the extent of spreadsheet error, which included studies by Hicks (1995), Coopers and Lybrand (1997), KPMG (1998), Butler (2000), and others. There are many software packages and tools such as SpACE, ComplyXL, ClusterSeven and Actuate which tend to assist in locating and rectifying errors within complex spreadsheet models. Majority of the investigations concerning spreadsheet problems have been from a technical perspective. This paper is a part of the investigation within University of Wales Institute Cardiff (UWIC) that is examining the spreadsheet risks from a management/organisational point of view rather than from a technical stance. The purpose is to relate the risks associated with spreadsheet use to organisational risk management. The aim is to develop a framework for categorising spreadsheet use, incorporating risk as one of the criteria. Once this framework is developed, it could prove very valuable for managing risks associated with spreadsheet use.

Many researchers have categorised errors in spreadsheets (PWC, 2004; Panko, 1996; Galletta, 1993; Rajalingham, Chadwick and Knight, 2000; Rajalingham, 2005). PWC (2004) categorised spreadsheets by use and complexity. This research attempts to incorporate the risk factor into the categorisation of use of spreadsheets. The paper proposes a number of models of categorisation which are based on a pilot study conducted within the accommodation office of the Facilities
Department at UWIC. Subsequently a comparative analysis of the various models proposed is presented.

This paper is organised into three further sections. Section 2 sets out the context of the pilot study. Section 3 highlights the three models that were proposed as a result of the pilot study. Section 4 then presents a comparison of the models proposed. Then finally the conclusions are presented.

2. THE STUDY:

2.1 Organisation:

For conducting the pilot study the Accommodation Office, was selected within the Facilities Department of UWIC. The Office is responsible for arranging accommodation for students and therefore includes all the halls of residence as well as assisting in finding private accommodation. During the summer, the same department works with the Conference Services Department to organise accommodation for conferences. The spreadsheet use within the department is diverse ranging from basic record keeping (trivial) to complex budgeting and forecasting (strategic).

2.2 Methodology:

The research methodology behind the pilot study reflects an interpretive approach due to the nature of this investigation. As Saunders et al. (2006) highlights, business situations are complex and unique. This study was conducted in order to understand how this department use spreadsheets and perception of their importance and the associated risks.

As this study was a pilot, it was decided that Un-structured interviews were appropriate, in order to extract maximum information without any bias or influence on the interviewee. As highlighted by Bell (1993), ‘unstructured interviews centred round the topic area may produce a wealth of valuable data.’ The rationale behind the pilot investigation was to identify and analyse which aspects to explore further and areas that need to be discarded.

The data gathered was analysed to identify the spectrum of use of spreadsheets within the department. The models proposed are based on conducting in-depth one-to-one interviews of all the key personnel involved with development and use of spreadsheets within the department. All the interviews were recorded and then transcribed and analysed.

The authors initially presented the models INFORMS, 2006 and the feedback from that has been incorporated into the discussion.

3. FINDINGS:

The spreadsheets were an integral part of the case study department. On one extreme they were used for designing basic paper forms or data stores, whereas on the other extreme they were used for budgeting and forecasting, which formed the more strategic use of spreadsheets.

3.1 Model 1:

Dimensions for Model 1 (Ref. Figure 1):

- Use: The categories for this dimension of the model were based on the type of use of the spreadsheet. The use of spreadsheets within the department was significant. They were
mainly used as a data source. Some of the employees used complex models too, for budgeting and forecasting. Therefore the spreadsheets could be classified as being ‘Trivial’ data sources or ‘Strategic’ spreadsheets. Further analysis of the interviews revealed that within the Strategic spreadsheets, there were calculative (The spreadsheets used for budgeting and forecasting) and non-calculative spreadsheets (For example, The UWIC Rider Bible which recorded the details of the student passes for UWIC Bus service). Therefore the first model had three categories within use.

1. Trivial Informative
2. Strategic (Non Calculative/ Informative)
3. Strategic Calculative.

• Importance: This dimension was based purely on reliance on the information contained within the Spreadsheet. Therefore the three categories produced:

   1. Not important/ Little Important
   2. Important
   3. Critical

• Risk: For this dimension the conventional categorisation was used. Any definition of risk is likely to carry an element of subjectivity, depending upon the nature of the risk and to what it is applied. (Riabokon, 2004) As such there is no all encompassing definition of risk. Chicken & Posner (1998) acknowledge this, and instead provide their interpretation of what a risk constituents:

   Risk = Hazard x Exposure

They define hazard as “… the way in which a thing or situation can cause harm,” (ibid) and exposure as “…The extent to which the likely recipient of the harm can be influenced by the hazard” (ibid). Therefore risk in this case can be assessed on probability of information being incorrect or erroneous and the extent of problems that can be caused by it being erroneous. The problems or the loss can be financial, operative or administrative. Risk, therefore, is measured in terms of impact and likelihood. Therefore the three categories proposed for this model were:

1. Low
2. Medium
3. High
Discussion for Model 1 (Refer: Figure 1):

This model highlights 27 different categories (3 X 3 X 3 Matrix) of spreadsheet usage. The research group at UWIC discussed the above model and it was highlighted that there is some overlap within the categories of ‘Use’. Thus it was proposed that it might be worth reducing the use categories to Calculative and Non-Calculative and the spreadsheets that are used for strategic purposes can be categorised on the level of ‘Importance’. Thus a new framework for categorisation was proposed (Refer: Figure 2), with just two categories of ‘Use’ i.e. Calculative and Non-calcuative.
3.2 Model 2

This model was considered to be acceptable specific to the case study department for the pilot but the classification was considered more complex. As can be seen there seems to be some overlap between ‘Risk’ and ‘Importance’. For example, something that represents a risk can be assumed to be important to the organisation. Considering the complexity of the above model, another classification has been proposed (Refer: Figure 3). This model emphasise Importance vs. Urgency.

Dimensions for Model 2 (Refer: Figure 3):

- The Importance means that dealing with that spreadsheet results in a high pay-off.
- Urgent means tight deadlines are associated with it. (The urgency dimension was inspired by Pawel J. Kalczynski (2006) paper presented at UKAIS Conference 2006.) One of the reasons why spreadsheets are commonly used is that they are quick and easy to use tools. Most of the organisations can easily differentiate what is urgent and what is not.

Discussion:

This section now describes the individual sections of the model along with the proposed approach that can be followed for these sections.

- **Section 1**: Low Importance and Low Urgency: Do nothing
- **Section 2**: Low Importance and High Urgency: This is the situation in which spreadsheets, being the quick and easy-to-use tool, are commonly used.
- **Section 3**: High Importance and High Urgency: This is the category of spreadsheet is critical and the organisation requires to put management controls in place.
Section 4: High Importance and Low Urgency: In this category, as the urgency is low, it might be better to use other methods, like database etc.

This model is very simple to implement and apply on individual basis, but there are further two dimensions, which need to be addressed, i.e. ‘Complexity’ and ‘Risk’, thereby making it complex and hard to implement in big organisations.

Risk in this model is weighted by assessing management structures in place for three variables: (Refer: Figure 4)

1. Spreadsheet Error
2. Control
3. Compliance

![Figure 4: Three variables for Risk measurement for Model 2](image-url)
3.3 Model 3

The overlap between the ‘Use’ and ‘Importance’ in Model 1, the fuzziness between the ‘Risk’ and ‘Importance’ in Model 1 (modified), and problems with implementing Model 2 in big and complex organisations, lead the researcher to propose another model (Model 3: Refer Figure 5). This model works on more conventional approaches towards Risk.

**Dimensions for Model 3 (Refer: Figure 5):**

- The X-Axis is to classify the ‘Magnitude’ of risk i.e. in other words the classification for this dimension can be based on severity of the consequences of errors within spreadsheets, which can be financial or business risk (which can also include reputation and compliance).
- The Y axis highlights the ‘Dependency’, which can be operational, tactical or strategic.
- The ‘Urgency’ dimension from Model 2 can also be incorporated into this model.

![Figure 5: Model 3](image)

**Discussion:**

When defining the ‘Dependency’ dimension, as mentioned earlier, the classification can be strategic, operational or tactical. Another possibility is the PWC (2004) classification as defined below, which is mainly for the financial domain.

1. Operational: Spreadsheets used to facilitate tracking and monitoring of workflow to support operational processes.
2. Analytical/Management Information: Spreadsheets used to support analytical review and management decision making. These may be used to evaluate the reasonableness of financial amounts.
3. Financial: Spreadsheets used to directly determine financial statement transaction amounts or balances that are populated into the general ledger and/or financial statements.
4. COMPARISON OF THE THREE MODELS:

The categorisation of the first model derived from the pilot study can be considered to be specific to this case study department. Considering the spectrum of use of spreadsheets within large organisations Model 1 seems the most appropriate.

The authors suggest that it might be complex to segregate into one of the 27 categories, therefore considering the practical application Model 2 was proposed. This model is simple to apply practically but it is weaker as there are two further possible dimensions (Complexity and Risk). This might need to be considered in organisations larger than the pilot organisation. Further this model is mainly for designing/development of new spreadsheets, and it does not take into account the thousands of existing spreadsheets which might be in use within big organisations. Even as far as the case study organisation is concerned, most of the spreadsheets tend to be in Section 1 of Model 2 (Figure 3). Within this department the level of urgency is low, because it concerns maintaining regular records and generating feedback at the end of year.

Model 3 uses a more understandable terminology within the majority of organisations. This aspect makes it easier to understand and implement in different types of organisations. Furthermore the authors believe that for the dimensions in this model it is easier to define criteria and define categories. Means of measuring is yet to be investigated.

5. CONCLUSION:

The spectrum of spreadsheet use within the pilot study organisation was very diverse. The research suggests that the first model proposed by the authors was simple but the key problem encountered within this model was the difficulty to classify a specific spreadsheet within one of the 27 categories of spreadsheets. But the supportive argument is that it does cover use of spreadsheets within organisation well.

The Model 2 on the other hand is also simple but easier to understand. Findings suggest that this model is also easy to apply generally, but one might need to consider two further dimensions (Risk and Complexity) when considering bigger organisations. Emphasis is placed on new spreadsheet development, and there is some fuzziness in dealing with thousands of spreadsheets already existing within the organisation.

According to the analysis the Model 3 seems to be the most acceptable for categorising spreadsheets. The primary reason is that it uses more conventional and universally understandable terminology and could be applied to a wide range of organisations. More specifically the dimensions are easier to understand, therefore the model is easier to implement. The authors perceive that it would be easier categorise using the Model 3 dimensions, but the means of measurement and categorisation are yet to be investigated. The next stage of the research is to develop clear definitions of each of the dimensions i.e. Magnitude and Dependence and then investigate appropriate criteria to measure and develop specific categories.

6. REFERENCES:
• Kalczynski, P. J. (2006), Representing Fuzziness in the Ontology of Time for Semantic Web, UKAIS Conference Proceedings, held at University of Gloucestershire, April 9-11.
• Panko, R. (1996), Spreadsheets on Trial, 29th Annual Hawaii International Conference on System Sciences.
Awareness and Market Changes in the Control of End User Computing Risk Management

EuSpRIG 2007

Visibility is a key component in the Effective Management of End User Application Risk

Elements of the IT infrastructure are visible and controlled under a lifecycle management process.

Not centrally managed and less likely to be documented applications include:
- Desktop Reports, Forms
- Spreadsheets
- Desktop Databases
32% of Corporate Data is Stored within End User Computing (EUC) Applications

"Today's regulatory environment demands more transparent and controlled processes, and spreadsheets are a weak link in the chain of accountability."
Neil Raden

Source: Baseline Consulting annual survey of 250 senior IT managers

30-40% of Corporate Data is Not Being Controlled Creating Unacceptable Risk

- Leading National mortgage lender writes down $2.4B worth of mortgages
  - Stock drops 13%
  - $8.5 billion knocked off the bank’s market value
  - Change control error in key Spreadsheet

- European Bank loses 30% of market cap overnight when $691MM trading fraud revealed. Trader had manipulated trade results using a programmatically hidden sheet and an alternate data feed

- Online retailer reports material understatement for cost of goods sold
  - Share price drops 25%
  - CEO loses position
  - Spreadsheet calculation error

Source: EuSpRIG Documented
Control Requirements Are Rapidly Changing
Accounting Standard getting tougher

Significant Trends:
- Audit Warnings regarding improperly controlled EUCs
- Audit Deficiency Statements around EUC applications that remain uncontrolled
- Major Mortgage Lender Suing previous Auditor for $2B for failing to identify problems
- 150-300 Key Spreadsheets under Control unlikely to remain the standard/requirement
- Many Corporations and Banks in particular often have millions of undocumented and uncontrolled financial spreadsheets

... and not just spreadsheets, but PC databases, reports and other end user applications that like spreadsheets - require control mechanisms

Perspective Evolves
- 1+ year ago – spreadsheets = glorified calculators
- Today – spreadsheets = applications
- Contain formulas and logic
  - Informs management on most operations decisions
  - Demand signals – ‘what do we build?’
  - Sales forecast – ‘what is our revenue this quarter?’
- Feed data to almost all ERP systems
- Contribute critical answers for shareholder reporting
  - 99% of all companies’ shares outstanding calculation done in a spreadsheet – this is the denominator of EPS

Trend – increasing awareness that spreadsheets need control
Production Level Testing and Control Technology Implications

<table>
<thead>
<tr>
<th>Need</th>
<th>Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUC includes</td>
<td>Flexibility – document, monitor and control any critical client resource/data</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>• Errors</td>
</tr>
<tr>
<td>Databases</td>
<td>• Fraud</td>
</tr>
<tr>
<td>Reports</td>
<td>• Data Leakage</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>EUC feeds external reporting process,</td>
<td>Scale and filter - 100,000's to millions of EUC resources</td>
</tr>
<tr>
<td>most ERPs and virtually all operational</td>
<td></td>
</tr>
<tr>
<td>decision making</td>
<td></td>
</tr>
<tr>
<td>Complex references, linking and inter-</td>
<td>Rich functionality - Accommodates many-to-many interdependencies</td>
</tr>
<tr>
<td>connections</td>
<td></td>
</tr>
</tbody>
</table>

Quality assurance for spreadsheets is as important as it is for the ERP systems

"As some companies have discovered, errors in relatively simple spreadsheets can result in potential material misstatements in their financial results. Recently, several large companies have either publicly disclose control deficiencies or been publicly censured by regulators related to insufficient spreadsheet controls."

PricewaterhouseCoopers. The Use of Spreadsheets: Considerations for section 404 of the Sarbanes-Oxley Act.
End User Computing Risk Management

Control & Compliance

1. Discover
2. Validate
3. Control

Agentless Discovery – Find EUC Applications & Files

- Find hidden sources of financial information distributed across organization
- Across Highly Distributed Servers, Desktops and Notebooks
- Create a secure centralized snapshot repository for analysis and audit
Discover – Relationships
Many Significant Spreadsheets feed or are fed by other applications

Map relationships between key data sources

Answers:
- What connects to what?
- Who "owns" the application?
- Who has copies of the application?
- Where does the data originate?
- What data feeds other applications?

Discover – Change Management

Examine changes – EUC changes as well changes in related applications and systems:

- Structural changes (New columns, fields, etc.)
- Reference changes (Where information is sourced)
- Properties changes (Author, Subject, Host location, etc.)
Validate - EUC Applications & Files

- Rules, template and policy enforcement
- Automated error discovery
- Spreadsheet complexity analysis and sorting
- Integrated audit documentation
- Graphical data and formula flow visibility
- Graphical precedent and dependent representation
- Alerts, cell analysis, dropped links, numbers stored as text, etc.
- View, print, or export
  - Where information resides
  - How it has been changed and by whom
  - Application interdependencies
  - How changes affect "downstream" applications

Benefits of EUC Risk Management

- Decreases financial and non-compliance Risk
  - Increased transparency eliminates errors and omissions of content and processes
  - Provides peace of mind to everyone responsible for control and compliance (SOX, FDA 21 CFR part 11 etc...)
- Documents currently undocumented information from distributed systems
- Improves the understanding of accounting and finance application dependencies and information flow
- Enhances visibility and control
  - Critical applications stored, monitored and managed in a secure repository
- Automates manual audit processes
- Allows Scarce finance and IT resources to be freed up
Please visit us at www.compasssoft.com, or contact me directly at pbach@compasssoft.com

Risk, Spreadsheets and Compasssoft

"Compasssoft is the market leader for enterprise spreadsheet management."
  - Bloor Research

"The first one-stop-shop for spreadsheet controls."
  - Gartner Research
Influences of The Compasssoft Perspective

- Over 160 public companies use Compasssoft EUC Risk Management software

[Logos of various companies]
Impact of Errors in Operational Spreadsheets
Stephen G. Powell, Barry Lawson, and Kenneth R. Baker
Tuck School of Business
Dartmouth College
Hanover NH 03755 USA
May 22, 2007

Abstract
All users of spreadsheets struggle with the problem of errors. Errors are thought to be prevalent in spreadsheets, and in some instances they have cost organizations millions of dollars. In a previous study of 50 operational spreadsheets we found errors in 0.8% to 1.8% of all formula cells, depending on how errors are defined. In the current study we estimate the quantitative impacts of errors in 25 operational spreadsheets from five different organizations. We find that many errors have no quantitative impact on the spreadsheet. Those that have an impact often affect unimportant portions of the spreadsheet. The remaining errors do sometimes have substantial impacts on key aspects of the spreadsheet. This paper provides the first fully-documented evidence on the quantitative impact of errors in operational spreadsheets.

1. Introduction
Although previous research has suggested that errors are prevalent in spreadsheets (Panko and Halverson, 1996; Panko 2005), we have only limited information on the types of errors that occur, why they occur, and how to avoid them. We recently summarized the research literature on spreadsheet errors (Powell et al., 2006a) and came to the following conclusions:

- There is no generally agreed upon taxonomy of errors in spreadsheets.
- The distinction between actual errors and poor practices is not well-defined.
- All studies that have examined spreadsheets in the field have found errors, although there is no standardization in these studies of
  - the categories of errors
  - the methods used to detect errors
  - the sample of spreadsheets studied.
- It is not possible at present to estimate accurately the prevalence or the impact of errors in spreadsheets.

The research we report here was designed to test for errors in a large sample of spreadsheets in actual use by organizations. We developed an explicit auditing protocol and trained a group of researchers to apply it consistently. Using an explicit protocol is important for two reasons: one, it allows other researchers to replicate and improve on our work; two, it contributes to the development of improved auditing procedures, which is important in its own right. The auditing procedure is described in detail in Powell et al. (2006b).

We begin this paper with a brief review of previous work on spreadsheet errors. Then we describe the design of our study and the sample of spreadsheets we audited. In a previous study we audited 50 spreadsheets from a variety of organizations and summarized the results in terms of error instances (the occurrence of a single type of error) and error cells (the number of cells affected by a single error instance). The current study focuses on the quantitative impact of errors. Each error identified in this study was verified by the developer of the spreadsheet, and the quantitative impact of fixing the error was determined.
2. Research design
Our research into spreadsheet errors is focused on completed, operational spreadsheets, not errors made in a laboratory setting or errors made during the development of a spreadsheet. In a previous study we used an auditing procedure on a large number of spreadsheets, but we did not have access to the developers. This allowed us to audit a large number of spreadsheets, but it also meant that we could not check our understanding of a model with the developer. In practice, this meant that at times we accepted a suspicious formula as correct because we could not be sure that it was incorrect. In the current study we worked with 25 volunteer spreadsheet developers within five different organizations. The developers each completed a survey describing their spreadsheet’s purpose and design, as well as any unusual or special formulas or assumptions. Two researchers then independently audited each spreadsheet and pooled their results. Finally, we debriefed the developer on each issue our audit raised, and categorized each issue as an error, a poor practice, or not an error. Actual errors were then corrected and the change in the relevant output cell was recorded as the quantitative measure of the impact of the error.

2.1 Sample spreadsheets
We first identified five organizations that were willing to provide volunteers and to have their spreadsheets audited. This included two consulting companies, a large financial services firm, a manufacturing company, and a college. Each organization identified five volunteers, each of whom provided one spreadsheet for auditing. We provided the following specifications to the volunteers for help in choosing a spreadsheet to audit:

- contains 3-10 worksheets
- contains 250-10,000 formulas
- occupies 200-1,000 kb of memory
- developed by the volunteer in the last twelve months
- contains no complex Visual Basic code
- is well understood by the developer
- has no broken external links

Not all the spreadsheets we audited conform to these specifications. In fact, the average number of sheets in our sample was 15.2 (the range was 2 to 135 sheets) and the average size in kilobytes was 1,463 (the range was 45 to 7,450kb). Many of the spreadsheets in our sample were larger on one or more of the dimensions than specified above.

While our sample is not strictly random, it is certainly representative of the general population of spreadsheets (with the caveats cited above). The sample includes spreadsheets from different types of organizations, spreadsheets created by developers with different degrees of expertise, and spreadsheets that span a broad range from small and simple to large and complex.

2.2 Auditing protocol
The auditing protocol we developed for a previous study is a highly-detailed document that specifies the steps to take in auditing a spreadsheet for size, complexity, several types of qualitative features, and errors. (A complete description of the protocol is available at our research project’s website, http://mba.tuck.dartmouth.edu/spreadsheet/index.html.)

This protocol evolved over several months of testing. During this time we trained auditors and tested the protocol ourselves and through our assistants on dozens of operational spreadsheets. Our current study used a very similar approach but focused less on gathering data about the spreadsheets and more on finding potential errors. The typical approach was to review the survey
Impact of Errors in Operational Spreadsheets
Powell, Lawson & Baker

provided by the developer, especially the portion describing the various worksheets and their interrelationships. The next step was to run the auditing software Spreadsheet Professional (http://www.spreadsheetinnovations.com), which provides summary maps for each sheet and reports on the location of potential errors. We then examined each sheet in turn, first looking at the map for suspicious cells or ranges and scanning the error tests for unusual conditions. Then we inspected the sheet itself, first determining the location of data and formulas and subsequently auditing every unique formula and most copied formulas.

In this study we recorded data only on cells that were potentially errors. For each problematic cell or range we recorded the following information:

- location: cell address or range
- type of error (see below)
- how it was discovered (whether by map analysis, error tests, code inspection, or sensitivity testing)
- description of the possible error

After we had discussed the potential error with the developer, we then recorded how the issue was resolved (Error, Poor Practice, No Error). For Errors, we also recorded the cell used to measure the impact and the absolute and percentage changes in that cell when the error was corrected.

2.3 Measuring impacts
Measuring the impact of an error on a spreadsheet is necessarily somewhat subjective. First, some errors occur in a single cell while others occur in many cells. Do we consider each cell separately and measure the impact of correcting it alone, or do we correct all cells with a similar error and measure the overall impact? Second, some error cells are used to calculate many other cells while others impact no other cells. When a cell impacts many other cells it is not always obvious which of the impacted cells to use to measure the effect. (And, of course, different errors can impact different cells.) Third, it is not always clear how to correct an error. For example, if erroneous inputs were used do we replace them with average inputs or extreme inputs? Finally, it is necessary to decide whether to measure errors in absolute or relative terms, and whether to combine all the errors in a given workbook into one overall error.

In this study we chose to measure the effect of each error separately. In most cases we corrected all cells with a given type of error, considering this one error with a single impact. When such an error impacted only the erroneous cells themselves we computed the maximum change from the base case and took that as our error estimate. When such an error impacted a single cell we measured the impact of correcting all the error cells on that one cell. We did not attempt to determine a single error estimate for each workbook but measured each error separately. In many cases the only cell impacted by an error was the cell itself. When an error cell had dependencies, we traced these to what we judged to be the most important dependent cell and measured the impact of correcting the error on that cell.

When we began this research, we had expected that most spreadsheets would be set up to calculate a few key outputs. If this were the case, it would be straightforward to determine the impact on these key outputs of an error anywhere in the spreadsheet. We were surprised to discover during our research that many spreadsheets do not have a small number of key output cells. Rather, we found many examples in which hundreds or thousands of results were calculated that themselves had no dependents. This made it more difficult to determine the impact of errors.
In the end we had to use quite a bit of judgment to decide which cell or cells to use to measure the impact of errors.

3. Error types
One of the challenges of spreadsheet error research is how to categorize errors. As we pointed out earlier, many different error classifications have been offered (Rajalingham, et al., 2000; Purser and Chadwick, 2006). Most of these suffer from the same flaw: errors that arise from different causes cannot be distinguished by an auditor. For example, when we encounter an error in a formula we can rarely determine whether the error was due to sloppy typing, lack of domain knowledge, lack of Excel knowledge, a subsequent user changing the formula, or an unknown cause. We can, however, easily detect some formulas that give the wrong result. We can also identify many practices that are likely to cause errors as the spreadsheet is used or that simply will make it harder than necessary to use the spreadsheet productively. Other poor practices include limited or nonexistent documentation, duplication of inputs, illogical physical layout, and so on.

After considerable testing we settled on the following six error types that our experience with auditing suggested were well-defined in theory and could be identified with high reliability in practice:

- Logic error - a formula is used incorrectly, leading to an incorrect result
- Reference error - a formula contains one or more incorrect references to other cells
- Hard-coding numbers in a formula - one or more numbers appear in formulas and the practice is sufficiently dangerous
- Copy/Paste error - a formula is wrong due to inaccurate use of copy/paste
- Data input error - an incorrect data input is used
- Omission error - a formula is wrong because one or more of its input cells is blank.

More information on this error taxonomy is available in Powell et al. (2006c).

We should point out that hard coding was identified in our previous study as the most common poor practice. In this study we ignored hard coding in most instances, on the grounds that it was unlikely to represent an outright error. However, there were a few instances in which contradictory inputs were hard-coded and we did cite those as potential errors.

4. Impact of errors
Table 1 summarizes our results. In column 1 we have used a two-digit code to label each spreadsheet. For example, spreadsheet 3.4 is the fourth spreadsheet from organization 3. The table gives the following information for each spreadsheet:

- number of issues raised in our audits
- number of errors confirmed in interviews
- number of errors with non-zero quantitative impact
- maximum percentage impact
- maximum absolute impact

Within this sample of 25 spreadsheets we identified a total of 381 issues. After we discussed these issues with the developers we found that nine spreadsheets had no errors; among the remaining 16 spreadsheets we found a total of 117 errors. Of these 177, 47 had zero quantitative impact, leaving 70 errors with non-zero impact.
As we pointed out above, there are two ways to measure the impact of errors: absolute and relative. Absolute impacts are important because they tell us how large errors are in the units of the spreadsheet. However, they cannot be compared easily across workbooks, since a million dollar error may be trivial in one spreadsheet and catastrophic in another. Relative (or percentage) errors more accurately reflect the significance of an error, but they have their shortcomings as well. One problem with relative errors is that percentage changes cannot be determined when the initial value is zero; another is that percentage changes in percentages are not generally as meaningful as percentage changes in dollar amounts. We present our results here in both absolute and relative terms.

Figure 1 shows the distribution of absolute error impacts. The most salient point to draw from this figure is that 47 of the errors we found had zero impact on the spreadsheet. This often came about when a formula had an erroneous reference, but both the erroneous and the correct input cells had the same value. Thus when the error was fixed the results did not change.

Returning to Figure 1, we see that 12 of the errors involved percentages; among these the average absolute change was 22%. Twenty-four of the remaining 58 errors involved absolute errors less than $10,000. However, some errors were huge: the largest single absolute error we found was over $100 million!

Figure 2 shows the distribution of percentage error impacts. (The NA category includes four errors in cells with an initial value of zero, for which a percentage change is not defined.) Of course 47 of the 117 errors had zero impact, regardless of how measured. Forty-seven of the 66 remaining errors were less than 10% of the initial value. As with absolute errors, there are some very large errors: four, in fact, were over 100%.

Our evidence suggests that spreadsheet practice is very different among the five organizations we studied. In the five spreadsheets from Organization 5 we could identify only five issues to discuss with the developers and no errors were identified among those five issues. Organization 5 is a small consulting company with highly educated employees and a culture that demands excellence. The spreadsheets we audited from this firm were works of art: thoughtfully designed, well documented, easy to understand, and error free.

Organization 4 had two spreadsheets with no errors and two with 22 and 44 errors, respectively. The quality of the spreadsheet practice in this organization clearly depends on just where one looks. In this case we found both the best of practice and the worst of practice in offices just a few miles apart.

In Organization 3, which is another consulting company, all the spreadsheets we audited had errors but in three cases no error had a measurable impact on the results. Even in the remaining two spreadsheets the errors were few in number and fairly small in terms of impact.

Organizations 1 and 2 are both very large. One is a financial firm and the other is a manufacturing firm. Some of the spreadsheets we audited from these companies were astonishingly large and complex. Perhaps for this reason, only two of the ten we audited were error-free (four had no errors with impact). The quality of spreadsheet practice in both of these companies was inconsistent, with inadequate attention paid to spreadsheet design, simplicity, ease of use, documentation, and consistency.
We can summarize our findings as follows:

- Some organizations have spreadsheets that are essentially error-free.
- Within a single organization, spreadsheet practice can range from excellent to poor.
- Some organizations use spreadsheets that are rife with errors and some of these errors are of substantial magnitude.
- Many errors have zero impact, or impact unimportant calculations.
- There is little correlation between the importance of the application or the risk involved and the quality of the spreadsheet.
- Few spreadsheets contain errors that, in the eyes of their developers, would destroy their usefulness.

5. Qualitative observations

Many writers have observed that the spreadsheet, for all its attractiveness to end-users, is in some ways a dangerous modeling platform. Not only are the logic of the model and the numbers commingled, but the physical layout permits unstructured designs. It is not surprising that amateur programmers, who lack structured design methods, make errors when using such free-form software.

Our research has shown that errors come in more varieties than perhaps even the most extensive taxonomy can encompass. Because the spreadsheet platform is so unstructured, and because end-users generally follow unique designs, errors and poor practices can arise in thousands of different guises. Error researchers inevitably must use their judgment in deciding what is an error and what is not. Thus we should be skeptical of claims about the frequency and impact of errors based on rough averages and casual research.

Another general observation that our research supports is that spreadsheet auditing is more difficult and limited than might have been anticipated. We knew in advance that we would not be able to identify errors in problem formulation or in the use of the model by auditing the spreadsheet itself, although these types of errors may be the most consequential. But even within the narrower domain of our audits we encountered limitations. First, the data used in most spreadsheets is undocumented and there is no practical way to check it. Even the original developer would have difficulty checking the data. Second, many spreadsheets are so chaotically designed that auditing (especially of a few formulas) is extremely difficult or impossible. Finally, we have found that many spreadsheets do not have just a few key outputs but are used to calculate hundreds or thousands of results. This makes it difficult to unambiguously measure the impact of a particular error.

One important generalization our work supports is that many errors are benign: they either have no impact on the results, the quantitative impact is very small, or the effect is on a vestigial portion of the spreadsheet that is no longer of interest. One can conjecture that this is the result of a sensible attitude toward errors on the part of spreadsheet developers. Perhaps developers look out for errors that impact the key outputs, and in general are good at correcting them. However, they pay less attention to inconsequential errors and therefore more of these survive to be observed. And, as we know from our interviews, many developers do not clean up their spreadsheets before they move on to other tasks.

One factor that might explain the substantial differences within and among companies in the quality of their spreadsheets is the degree of risk involved. We might hypothesize that companies
devote their best resources to high-risk spreadsheets, and fewer resources to low-risk ones. (There is some evidence in a user survey we conducted that would support this conjecture. See Lawson et al. (2006).) We did not measure this feature of the spreadsheets we audited (this would certainly be difficult to do), but our impression is that no such correlation existed within our sample. For example, one of the best spreadsheets we audited was designed to help with daily staffing of nurses and doctors to a medical practice. The spreadsheet was elegantly engineered, error-free, and easy to use. But little was at risk: an error in staffing at worst would assign the same person to adjacent shifts or to too many consecutive days. Errors of this type would almost certainly be caught, and their impact would be negligible. None-the-less, the spreadsheet was nearly perfect. By contrast, we also audited spreadsheets in use in a major financial firm for calculating tax liabilities (measured in the billions) to various state and national entities. These spreadsheets were astonishingly complex, difficult to understand, difficult to work with, and error-prone. So factors other than risk appear to explain spreadsheet quality.

Another observation that helps to understand our results is that many of the developers we worked with were not especially surprised or devastated when we pointed out potential errors. Sometimes the reaction was that they knew the formula “wasn’t quite right” but they saw that it gave the right answer and thus was acceptable. Sometimes the reaction was that the result was “close enough,” or that the result in question was no longer used, or not important. So developers seem to have a sense of what level of accuracy is appropriate for a given spreadsheet. (It is another question entirely as to whether their perceptions are correct, and the spreadsheets are actually as accurate as they need to be.)

An experienced auditor can rather quickly detect a spreadsheet that is likely to have errors. The major symptoms we observed of poor spreadsheet practice are the following:

- Chaotic design
- Embedded numbers
- Special cases
- Non-repeating structures
- Complex formulas.

Chaotic design refers to a poorly structured physical layout of the formulas and data. Numbers embedded in formulas, while not necessarily direct causes of errors, are strongly correlated with the presence of other problems. Special cases refers to designs in which similar results are calculated in slightly different ways, which requires great care in building and checking formulas. Non-repeating structures includes designs in which the formulas in a row or column change structure repeatedly, precluding the use of copying and pasting. In the hands of experts, complex formulas can be used to great effect. But in the hands of novices the same formulas can be error-prone.

Why is spreadsheet practice sometimes so poor? We cannot know for sure, but we did gather some anecdotal evidence during our interviews. When asked what kept them from building better spreadsheets, our developers typically cited one or more of the following reasons:

- Time pressure
- Organic design
- Changing specs
- Lack of testing
- Lack of relevant knowledge and skills
Time pressure was the most often cited reason. Many spreadsheets are built under great time pressure, which precludes use of some of the most effective methods for avoiding errors. Managers of spreadsheet developers should be aware of the effects of putting their employees under excessive time pressure. Another commonly cited factor was organic design: either the spreadsheet design was inherited from a predecessor spreadsheet, or it grew organically during the project without ever consciously being designed. Another complaint was changing specifications: if the designer had only known from the start what the spreadsheet was going to be used for, he or she could have designed it more appropriately. We also observed that very few of our developers used any formal approach to testing their spreadsheets; in fact, most of them did no testing as such. Finally, in some cases we could observe directly that the cause of an error was lack of relevant knowledge, either of the problem domain or of spreadsheet tools. It was remarkable, however, how rare this cause appeared to be. Most developers could see quickly that a particular formula was an error, once we had pointed it out to them. Only very rarely did we have to explain to them why it was an error, or how to fix it.

Finally, we offer a comment on the importance to auditing, and to good spreadsheet usage in general, of good design. Our work makes us extremely conscious that a well-designed spreadsheet is simple, consistent, and general. Simplicity means a logical use of worksheets and a logical and intuitive layout of each individual sheet. Simplicity makes building and auditing easier. Consistency means, for example, that a single formula can be written and then copied down an entire row or across an entire column. Such a formula can easily be checked. Rows or columns in which the formulas change structure constantly often hide errors. Generality means that the spreadsheet is built to handle all of the likely combinations of inputs that users will want to use. The opposite is a workbook in which individual cases are calculated separately, which makes it difficult to keep inputs consistent across cases.

6. Summary and Future Research
We have audited 25 spreadsheets from five organizations. We identified cells or ranges that appeared to be problematic and discussed each one with the developer of the spreadsheet. For each issue that was classified as an actual error, we then identified the cell or cells affected and measured the absolute and percentage impact of correcting the error. Several conclusions emerge from this research:

- The quality of spreadsheet practices differs substantially among and within organizations.
- Some individuals and organizations are capable of developing essentially error-free spreadsheets.
- Many spreadsheets are built in ways that violate good design practices.
- Operational spreadsheets are highly complex and often poorly structured.
- Poor practices (such as hard-coding numbers in formulas) abound, but quantitative errors are relatively rare.
- The quantitative impact of many errors is negligible or zero, or occurs in unimportant cells.
- Devastating errors are rare.

None of these conclusions should be taken as proven by the current research. Rather, they are suggestive hypotheses that should be refined through further research.
References


### Table 1
Summary of Audit Results

<table>
<thead>
<tr>
<th>Organization-Workbook</th>
<th># Issues</th>
<th># Errors</th>
<th>Errors with Non-zero Impact</th>
<th>Maximum Percentage Impact</th>
<th>Maximum Absolute Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>1.2</td>
<td>50</td>
<td>6</td>
<td>5</td>
<td>28.8%</td>
<td>$32,105,400</td>
</tr>
<tr>
<td>1.3</td>
<td>18</td>
<td>7</td>
<td>3</td>
<td>137.5%</td>
<td>$110,543,305</td>
</tr>
<tr>
<td>1.4</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>1.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>2.1</td>
<td>19</td>
<td>6</td>
<td>5</td>
<td>3.6%</td>
<td>$13,909,000</td>
</tr>
<tr>
<td>2.2</td>
<td>27</td>
<td>11</td>
<td>7</td>
<td>16.0%</td>
<td>$74,000,000</td>
</tr>
<tr>
<td>2.3</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>2.4</td>
<td>30</td>
<td>4</td>
<td>3</td>
<td>416.5%</td>
<td>$10,650,000</td>
</tr>
<tr>
<td>2.5</td>
<td>40</td>
<td>2</td>
<td>2</td>
<td>NA</td>
<td>8.90%</td>
</tr>
<tr>
<td>3.1</td>
<td>19</td>
<td>2</td>
<td>2</td>
<td>5.3%</td>
<td>$238,720</td>
</tr>
<tr>
<td>3.2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>3.3</td>
<td>11</td>
<td>2</td>
<td>2</td>
<td>15.6%</td>
<td>$4,930,000</td>
</tr>
<tr>
<td>3.4</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>3.5</td>
<td>23</td>
<td>1</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>4.1</td>
<td>27</td>
<td>22</td>
<td>12</td>
<td>116.7%</td>
<td>$13,355,445</td>
</tr>
<tr>
<td>4.2</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>141.8%</td>
<td>$272,000</td>
</tr>
<tr>
<td>4.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>4.4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>4.5</td>
<td>79</td>
<td>44</td>
<td>27</td>
<td>39.1%</td>
<td>$216,806</td>
</tr>
<tr>
<td>5.1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>5.2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>5.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>5.4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>5.5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Totals</td>
<td>381</td>
<td>117</td>
<td>70</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 1. Distribution of Errors, by Absolute Impact
(n = 117)

Figure 2. Distribution of Errors, by Percentage Impact
(n = 117)
Thinking is Bad: 
Implications of Human Error Research for 
Spreadsheet Research and Practice

Raymond R. Panko, 
College of Business Administration 
University of Hawai`i 
2404 Maile Way 
Honolulu, HI, USA 
Ray@Panko.com

ABSTRACT

In the spreadsheet error community, both academics and practitioners generally have ignored the rich findings produced by a century of human error research. These findings can suggest ways to reduce errors; we can then test these suggestions empirically. In addition, research on human error seems to suggest that several common prescriptions and expectations for reducing errors are likely to be incorrect. Among the key conclusions from human error research are that thinking is bad, that spreadsheets are not the cause of spreadsheet errors, and that reducing errors is extremely difficult.

1 INTRODUCTION

In past EuSpRIG conferences, many papers have shown that most spreadsheets contain errors, even after careful development. Most spreadsheets, in fact, have material errors that are unacceptable in the growing realm of compliance laws. Given harsh penalties for non-compliance, we are under considerable pressure to develop good practice recommendations for spreadsheet developers and testers.

If we are to reduce errors, we need to understand errors. Fortunately, human error has been studied for over a century across a number of human cognitive domains, including linguistics, writing, software development and testing, industrial processes, automobile accidents, aircraft accidents, nuclear accidents, and algebra, to name just a few.

The research that does exist is disturbing because it shows that humans are unaware of most of their errors. This “error blindness” leads people to many incorrect beliefs about error rates and about the difficulty of detecting errors. In general, they are overconfident, substantially underestimating their own error rates and overestimating their ability to reduce and detect errors. This “illusion of control” also leads them to hold incorrect beliefs about spreadsheet errors, such as a belief that most errors are due to spreadsheet technology or to sloppiness rather than being due primarily to inherent human error.
1.2 Which Error Research?

Most people who are involved with spreadsheets have a good grasp of spreadsheet error research. However, spreadsheet error research is only an insignificant dot in the total picture of human error research.

Given this wealth of research in many fields, focusing exclusively or even primarily on spreadsheet error research makes little sense. Such a narrow focus would bring the not invented here syndrome to a new level of silliness. Unfortunately, this has been done to an uncomfortable degree in past spreadsheet research and writing.

To jump ahead slightly, we should focus especially on research on programming errors and software testing/inspection because experiments and field inspections of operational spreadsheets have shown that software errors and spreadsheet errors are very similar in error rates, detection rates, and types of errors.

1.3 Convergent Validation

Every research methodology has both strengths and weaknesses. Consequently, scientists try to provide convergent validation, in which they measure the same construct using different methodologies. If most or all methods give comparable outcomes, then it is possible to have confidence in the results [Campbell & Stanley, 1963].

Figure 2 shows how convergent validity can work in spreadsheet error research. First, in the spreadsheet realm, there are surveys, experiments, and inspections (“audits”) of operational spreadsheets. However, there also are general human error studies, including software error studies.

Figure 1: Convergent Validity

Although it is true that all spreadsheet research studies have high convergent validity [Panko, 2007b], these studies must have high convergent validity with human error studies in other cognitive domains. As importantly, human error studies in other cognitive domains are likely to suggest practices that we can study to reduce errors in spreadsheets and are also likely to call into question some common spreadsheet good practice prescriptions that run counter to research in other areas.
2 ERROR FREQUENCY DATA

At the most basic level, human error research in many domains has measured the frequency of errors and error detection rates.

2.1 Error Rates by Level

Some of the best human error research has been done in writing. Research on human writing, for instance, has shown that writing is extremely mentally demanding. When a person is typing or writing, they are also concerned about the grammar of the entire sentence as well as what he or she has already said and what he or she still plans to say to complete the story or argument [Flower & Hayes, 1980; Hayes & Flower, 1980].

Figure 3 shows error rates by level. For mechanical actions, such as typing characters or words, human accuracy is 99.5% to 99.8%. At the level of complex thoughts, such as sentences, accuracy falls to 95% to 98%. At the level of a document, accuracy is 0%, in the sense that any document of nontrivial length will contain grammatical and spelling errors [Panko, 2007a].

Figure 2: Error Rates by Level

<table>
<thead>
<tr>
<th>Mechanical Action</th>
<th>Writing</th>
<th>Programming</th>
<th>Spreadsheet</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex Thought (C)</td>
<td>Typing</td>
<td>Typing</td>
<td>Typing</td>
<td>99.5%-99.8%</td>
</tr>
<tr>
<td>(C)</td>
<td>Spelling</td>
<td>Parentheses</td>
<td>Pointing</td>
<td></td>
</tr>
<tr>
<td>Product (C^N)</td>
<td>Document</td>
<td>Program</td>
<td>Spreadsheet</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Module</td>
<td>Module</td>
<td></td>
</tr>
</tbody>
</table>

In programming, these three levels correspond to typing actions, statements (lines of code), and programs (or preferably program modules). The accuracy percentages shown in the last column also hold for programming [Panko, 2007a].

Spreadsheet research has also shown the error rates shown in the last column [Panko, 2007b]. This is good because if spreadsheet research had gotten different results, the spreadsheet research would certainly be wrong.

2.2 Probability of a Spreadsheet Error

Most people are comfortable with the error rates in Figure 3 for typing errors and pointing errors and even for complex thoughts (formulas); they have a more difficult time accepting that nearly all spreadsheets contain errors. One reason for doubt is that people are not good at understanding how errors multiply. Given an error rate of 5% at the formula level, many people believe that there is only about a 15% or 20% probability of an error in the entire spreadsheet.

Actually, if the error rate per formula is C, then the probability of an error in the spreadsheet as a whole is approximately C^N, where N is the number of root (noncopied) formulas in the entire spreadsheet. Note in Figure 4 that even for small spreadsheets with only 10 root formulas, an accuracy rate of 95% to 98% at the formula level translates into a substantial probability of an error (18% to 40%). As soon as the number of root formulas rises to 100 (which actually is quite small for corporate spreadsheets), errors are almost 100% certain. Even if formula error rates are an order of magnitude smaller, large spreadsheets will almost certainly have errors.
Thinking is Bad: Implications of Human Error Research for Spreadsheet Research and Practice

Panko

Figure 3: The Probability of a Spreadsheet Error

<table>
<thead>
<tr>
<th>Root Formulas / Formula Accuracy</th>
<th>95%</th>
<th>98%</th>
<th>99%</th>
<th>99.50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>40.1%</td>
<td>18.3%</td>
<td>9.6%</td>
<td>4.9%</td>
</tr>
<tr>
<td>50</td>
<td>92.3%</td>
<td>63.6%</td>
<td>39.5%</td>
<td>22.2%</td>
</tr>
<tr>
<td>100</td>
<td>99.4%</td>
<td>86.7%</td>
<td>63.4%</td>
<td>39.4%</td>
</tr>
<tr>
<td>500</td>
<td>100.0%</td>
<td>100.0%</td>
<td>99.3%</td>
<td>91.8%</td>
</tr>
<tr>
<td>1,000</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>99.3%</td>
</tr>
<tr>
<td>10,000</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

2.3 Error Detection Rates Are Lower

So far, we have seen that people are 95% to 98% accurate when they create formulas. However, when people try to detect errors, their accuracy rates are much lower.

- In proofreading, for instance, spelling errors are the easiest to detect. However, even for nonword errors, in which the mistake does not create a new word (for example, ther for there), the detection rate is only about 85% to 95%. For word errors, in which the mistake gives another dictionary word (love for live), detection rates are only about 70%. For grammatical errors, detection rates are far lower [Panko, 2007a].

- In software inspections, in turn, individual inspectors only find 40% to 60% of all errors when they work alone, and this falls off if inspection is rushed or too lengthy. Group inspection can increase this yield to 70% to 80% of all errors [Panko, 2007a].

- Several experiments have been done on spreadsheet inspection [Galletta, et al., 1993, 1997; Panko, 1999]. They have shown that individual inspectors find 50% to 60% of seeded errors. This is consistent with proofreading success rates, software inspection rates and other detection rates in other human cognitive activities [Panko, 2007a].

All of these studies used 100% inspection, in which the person looked at each and every word, statement, or formula.

3. ERROR TAXonomies

It seems reasonable to say that to control errors, you must understand them. A number of error taxonomies have been put forward to illuminate the differences between different types of errors so that different countermeasures can be put into place for different errors if appropriate.

3.1 Criteria for an Error Taxonomy

Which error taxonomy is the best? There is no answer to that question because different taxonomies have different strengths and weaknesses. However, we should ask several questions to assess any taxonomy.

- First, we should ask whether the taxonomy can be applied reliably in inspections of finished spreadsheets or whether they can only be applied during observation of the actual development. If a method is developed for observation, its application to inspections of finished spreadsheets must be carefully scrutinized and validated.

- Second, do, different categories should have different error occurrence rates, different error detection rates, require different avoidance or detection methods, or have some
other operational importance. Distinctions that do not lead to differential actions need justification.

- Third, from a research viewpoint, awe must validate any measurement methodology for construct validity before it is used in a publication. This is a broad topic, but one aspect of construct validity is straightforward to measure and is a litmus test for publications. This is inter-rater reliability. Unless different people can get the same results when they apply a methodology to an artifact, the methodology lacks reliability. The most common way to measure reliability is to have multiple raters apply the methodology to a set of artifacts and then assess their consistency with Cronbach’s [1971] alpha or some newer reliability measure.

3.2 Norman and Reason

One of the most widely-used taxonomies is the mistakes/slips/lapses taxonomy created by Norman [1984] and Reason [Reason & Mycielska, 1982]. The basic distinction is between mistakes and slips or lapses. Mistakes are errors in intention. If we intend to do something that is inherently incorrect, this is a mistake. In contrast, lapses and slips are errors in executing an intention. If we have the right intention but the wrong execution, this is a slip or lapse, depending upon whether the mistake is do to a memory or sensory-motor failure.

The attractive thing about the Norman and Reason taxonomy is that it can be used in root cause analysis. However, it is designed to explain thinking and execution, so it is used most naturally in observational research. Construct validity analysis would have to be done to see if the taxonomy can be applied reliably to finished spreadsheet artifacts.

3.3 Rasmussen and Jenson

Another error taxonomy that was created for observational studies was the Rasmussen and Jenson [1974] taxonomy, which was developed for studies of experienced professionals with strong sets of rules for what to do in various circumstances. Rasmussen found three types of errors.

- Skill-based errors are basically slips or lapses in highly skilled behaviour.
- Mistakes are divided into rule-based or knowledge-based errors depending on whether the person has a well-established rule (first see if it is plugged in) or whether the person must use general knowledge to solve the problem. The distinction between rule-based and knowledge-based errors is important because when the subject does not have a rule and must resort to general knowledge, execution is likely to be slow with a high potential for errors.

However, the Rasmussen and Jenson taxonomy was created for detailed observational studies. Applying it to finished artifacts is questionable because you normally must be able to hear someone’s protocol analysis verbalizations to know whether the person is applying a rule or general knowledge. Also, unless the person has considerable expertise, they will not have the well-developed rule base needed to apply the taxonomy.

3.4 Panko

Panko and Halverson [1996, 1997] created a taxonomy with several distinctions:

- First, it distinguishes between quantitative errors (which give the wrong number immediately) and qualitative errors (which are likely to lead to wrong numbers later.)
• Second, based on Allwood’s [1984] work in mathematics, the taxonomy distinguishes between mechanical errors, logic errors in creating formulas, and omission errors (leaving something out of a model). This trichotomy of quantitative errors was found to correspond different error occurrence rates [Panko and Halverson, 1997] and, in a limited test, different error detection rates [Panko, 1999].

• Third, for logic errors, it distinguishes between Eureka errors [Lorge and Solomon, 1955], which are easily proven, and Cassandra errors, which are difficult to prove to be errors even if they are detected. Cassandra errors cause particular problems in error-seeking.

Figure 5 shows the author’s revised error taxonomy. Based on research in writing [Flower & Hayes, 1980; Hayes & Flower, 1980], which suggests three simultaneous phases—mechanical typing or writing, sentence construction, and overall plan—this paper replaces the mechanical / logic / omission distinction with a different trio. Mechanical errors are called slips or lapses to conform with the Norman and Reason distinctions. Logic errors are simply called formula errors to distinguish between logic errors in individual formulas and logic errors in the overall model design. Most significantly, omission errors are replaced with the more general concept of modeling errors, which are errors in the overall model (omission are only one type of modeling errors).

The justification for the slip/lapse-formula-model distinction is show in Figure 6. Here, detection rates are shown to be different, based on various spreadsheet development and testing studies. Although the numbers are rough, they show that the triad appears to be useful in distinguishing error conditions.
The model also adds input and output errors to the traditional slip/lapse-formula-modelling triad. This was left out of the original Panko/Halverson taxonomy because their research did not look at problems with dirty data for input or output problems such as misinterpretations. Obviously, these need to be added.

Most errors are blameless. Reason [1990] has noted that most human errors are due to the quirky ways in which our memory and cognition work, not because we are sloppy or because we use a spreadsheet. Humans evolved to have reasonable accuracy at reasonable cost and speed. Unfortunately, when bottom-line values in spreadsheets require hundreds or thousands of calculations, human cognition’s inherent limitations are insufficient.

However, some inaccuracies truly are blameful. Figure 5 lists three examples of blameful activities.

- Violations are willful departures from required practices. Violations would be departures from the company’s standards in spreadsheet development and testing.

- Puffery is making the results look better than they are by making questionable assumptions or choosing modeling decisions that strengthen your case. A certain amount of puffery is likely to be inevitable, but there is a grey zone beyond which puffery is culpable. More extremely, fraud consists of misrepresenting information or withholding information in order to convince another person to take actions against their economic interest or the interests of the firm. Fraud is a criminal offense.

- A final blameful action shown in the figure (although there undoubtedly are many more), is building spreadsheets despite a lack of expertise. Everyone stretches their spreadsheet ability or domain ability to some extent, but to build spreadsheets when the developer lacks sufficient expertise is culpable behavior.

3.5 Perspective

Overall, errors are like multiple poisons, each of which is fatal. Different errors may require different countermeasures for reducing their number and for doing testing.

In particular, one cannot simply examine the “hard parts,” such as long formulas. Certainly, long formulas are more likely to be wrong than short formulas. However, there are likely to be far more short formulas than long formulas, so it is likely that most errors occur in the “easy parts.” “Cherry picking” in testing or inspection is a poor idea.

4 ERROR RESEARCH: SUGGESTIONS

A good theory should suggest useful practices. We will look first at some possible suggestions for reducing errors based on error research. We will then look at several areas where current research and recommended practice may be questionable based on error research.

4.1 Software Errors and Spreadsheet Errors are Particularly Similar

One general suggestion is to look at the software literature on development errors, testing, and inspection. As noted earlier, the research to date indicates that spreadsheet error rates and detection rates are very similar to those found in software research. So are types of errors. This may surprise many people because software statements can be very complex. However, most software lines of code are very simple, and compilers or interpreters catch most errors in the placement of parentheses and other weirdness in programming.
However, if there really are parallel error and detection rates in spreadsheets and programming, this should be a sobering idea. Software researchers have shown that error is extremely persistent. No “silver bullet” has been found to reduce errors to negligible levels. In particular, experience in testing and inspection has found that roughly a third of all time must be spent in these to activities to have reasonable error detection rates [Panko, 2006b].

4.2 Reduce the Memory Load in Modeling

Linguistics has shown that a good way to cause errors is to increase someone’s memory load. As discussed earlier, writers must simultaneously consider typing, their current sentence, and the overall flow. The same helps in spreadsheeting. This suggests that such actions as modularization and outlining the overall module before building individual parts could reduce memory loads and so could reduce errors.

4.3 Do Not Rush

A third way to induce errors in linguistics is to ask someone to work very fast. As speed increases, error rates increase. This suggests, consistent with software research, that when we seek errors through inspection we must work slowly.

Probably, we should slow way down when we reach complex parts of the model during development and inspection. When drivers approach traffic hazards where they need to slow down, they usually slow down only slightly—not to a degree consistent with the danger in the hazard. We probably need to guard against this tendency in development and inspection.

4.4 Error Correction during Development

Allwood’s [1984] study of students solving mathematical problems used protocol analysis to look in detail at their development process. He discovered that they corrected many errors as they went along. Sometimes, they stopped and did error hunting when they were merely suspicious that “something was wrong.” In addition to this general error suspicious, they sometimes had error hypotheses in which they thought that a particular problem had occurred.

In many cases, the subject went back to check part of his or her work without having any sense of a problem. It was simply a precaution. These “standard check” episodes often found problems that error hypothesis episodes and error suspicion episodes failed to find.

One way to apply standard checks in spreadsheet development this is to check every formula after entering it. One way is to double-click on a formula cell or hit F2. Both show the formula and in color show the precedent cells. (This provides more information than the arrows functions on the reviewing toolbar.) It is important to hit escape after checking a formula to avoid changing it after using double-clicking or F2.

4.5 Use Short Formulas

A common good practice recommendation is to keep formulas short. However, Raffensperger [2003] has suggested the opposite—making formulas long instead of having a string of shorter formulas in order to reduce the length of links to precedent cells. This should increase understandability and should reduce link error in referring to precedent cells because the cells should be nearby.
The evidence to date seems to support keeping formulas short. In proofreading for words of different lengths, detection fell off extremely rapidly with word length [Panko, 1999]. In a spreadsheet inspection study, Panko [1999] found lower detection in longer formulas. However, the frequency of pointing errors to precedent cells was not studied. Direct research is still needed.

5 ERROR RESEARCH: CHALLENGES

In addition to suggesting new areas for spreadsheet research and practices, error research can cause us to question some generally accepted prescriptions for reducing spreadsheet errors.

5.1 Safe and Effective Prescriptions

In medicine, drugs must be proven both safe and effective before they are sold. If it is not effective, why bother? If it is not reasonably safe, the drug may do more harm than good. We must do the same thing for spreadsheet error research prescriptions. We need to conduct controlled experiments to test prescriptions for safety and effectiveness. Simple consensus is not enough. You cannot vote on truth. Fundamentally, our general lack of awareness of errors makes human perceptions beliefs to be suspicious.

5.2 Inspection

A number of studies have been done on spreadsheet inspection to look for errors. Here, the relevant human error research is software inspection research, although proofreading research should also be useful. Fortunately, the developer of software code inspection, Fagan [1976] added the tenet that results should be recorded and analyzed. This has led to a number of controlled studies in the real world. Based on those experiences, software inspections have evolved to include the following main points.

- Understand the requirements and the model logic before you inspect. This is critical for identifying a failure to meet requirements, omissions, and other model errors. Usually, code inspection begins with a meeting in which the author explains the requirements, flow of the model, and specific sections of code.
- Do not rush. Limit inspections to about two hours, and limit inspections to about 200 lines of code. When the rate of inspection goes up, error detection falls.
- Use team inspection. As noted earlier, inspection research in software and in spreadsheets has consistently shown that individuals catch only 50% to 60% of errors. Team inspection in spreadsheets can raise the detection rate to 80%. The gain comes almost entirely from the types of errors that individuals have the most difficult time detecting [Panko, 1999].

5.3 Use of an Assumptions Section: IPO versus Flow

The area that probably requires the biggest “reality check” is the desirability of or lack of desirability of using an assumptions section that contains all of the input data. The processing section of the model then will contain formulas with links back to the assumptions section.

One reason for suggesting the use of an assumptions section is that it allows the quick and accurate detection of all input cells. Second, software and statistical programs have traditionally separated programs files from data files (although object-oriented programming no longer separates all code from data).
Counting against assumptions sections, many formulas in the processing section will have to refer to distance cells in the assumption section. In an exploratory pilot study that was not published, the author found that when subjects had to point to precedent cells that were off the current screen, their error rates went up substantially. In addition, if the assumptions section is put on a different worksheet in an Excel workbook than the processing sections, neither F2/double-click nor precedent errors will give useful information to help check for errors.

Comprehension may also be reduced. When a data cell is looked at out of its processing context, it may be difficult to understand. Note that in all common forms, such as the infamous U.S. 1040 income tax form, the information is presented as a flow with data and formulas interspersed in a logical way.

Overall, we probably need to conduct a good deal of research on the suitability of input-processing-output (IPO) worksheet designs and flow worksheet designs.

### 5.4 Static Checking Programs

One way to reduce errors in word processing is to use the spelling checker function. In software development, this is called static testing. However, spell checkers are only good for finding nonword errors. They are likely to miss all world errors. People can do much better than this.

Grammar checkers are even more problematic. Galletta et al. [2005] found that students who used grammar checkers had *poorer* grammar in essays than those who did not. Analyzing the data, they found that students who used grammar checkers often took incorrect suggestions. Static testing programs do not really identify errors. They really say, “Look here” and perhaps given a correct or incorrect reason why.

Static inspection programs for spreadsheets generally have two functions. The first is to show the logic flow in the program or program module at a very high level, so that the inspector can divine the flow. This can be extremely helpful, but is it as effective as doing a preparation session in code inspection where this is done more formally?

Second, the static inspection program should identify errors. Unfortunately, there are certain spreadsheet errors for which static testing programs are likely to be ineffective. For instance, if the spreadsheet developer omits a variable, expecting a static inspection program to find it seems unreasonable. In addition, while some pointing errors will give indications of errors, others do not. Finally, if the developer has the wrong algorithm for a formula or expresses it incorrectly in terms of order of precedents, parentheses, or other matters, can the static checker find most or all of these errors?

Doing a prescan with a static inspection program is almost certainly a good idea. However, it is important not to expect too much from this. Given the number of static testing vendors and the enthusiasm of their claims, the goal of proving prescanning safe and effective is especially important.

We must understand what errors static testing programs identify and do not identify in order to have an idea of how much work the individual inspector must do after prescanning. Is the human part just a “mop up” operation, or is the end of prescanning simply where the real work begins?
5. CONCLUSION

The basic argument in this paper is a request that academics and professionals in spreadsheet error research draw on the vast human error literature to inform their research, work, and general thinking. Almost all of the results that we have seen in spreadsheet experiments or field inspections have been predictable in light of prior findings in human error research. In addition, this paper has shown a few samples of how human error research can suggest new research and practice directions and of how at least some aspects of research and recommended practice that we may need to challenge.

It is fundamentally important to understand that the problem is not spreadsheets. Asking what is wrong with spreadsheets has some value, but it is fundamentally the wrong issue. The real issue is that thinking is bad. Of course, thinking has some benefits, and we are accurate 96% to 98% of the time when we do think. But when we build spreadsheets with thousands (or even dozens) of root formulas, the issue is not whether there is an error but how many errors there are and how serious they are.

The second fundamental thing to understand from human error research is that making large spreadsheets error free is theoretically and practically impossible and that even reducing errors by 80% to 90% is extremely difficult and will require spending about 30% of project time on testing and inspection. Trusting “eyeballing” and “checking for reasonableness” to reduce errors to a reasonable level flies in the face of a century of human error research as is a testament to the profound human tendency to overestimate their ability to control their environment even when they cannot.

The third fundamental thing to understand is that replacing spreadsheets with packages does not eliminate errors and many not even reduce them. The problem, again, is that thinking is bad. Unless there are no human decisions or design actions in using a package, there will be thinking, and therefore there will be errors. In addition, packages often do only part of what must be done, requiring people to use spreadsheets for many risky calculations. A good package should reduce errors, but this is speculation that can and should be tested.

REFERENCES


Source Code Protection for Applications Written in Microsoft Excel and Google Spreadsheet

Source Code Protection for Applications written in Microsoft Excel and Google Spreadsheet

Thomas A. Grossman
University of San Francisco, School of Business & Management, San Francisco CA 94117
tagrossman@usfca.edu

ABSTRACT

Spreadsheets are used to develop application software that is distributed to users. Unfortunately, the users often have the ability to change the programming statements (“source code”) of the spreadsheet application. This causes a host of problems. By critically examining the suitability of spreadsheet computer programming languages for application development, six “application development features” are identified, with source code protection being the most important. We investigate the status of these features and discuss how they might be implemented in the dominant Microsoft Excel spreadsheet and in the new Google Spreadsheet. Although Google Spreadsheet currently provides no source code control, its web-centric delivery model offers technical advantages for future provision of a rich set of features. Excel has a number of tools that can be combined to provide “pretty good protection” of source code, but weak passwords reduce its robustness. User access to Excel source code must be considered a programmer choice rather than an attribute of the spreadsheet.

1. INTRODUCTION

Spreadsheets can be a powerful language to write application software for distribution to users. Unfortunately, the users often have access to the programming statements (programmers call these the “source code”) of the spreadsheet application. This makes it easy for users to change the application they are using, while they are using it. Although this feature is desirable for exploratory modeling, collaborative work, and rapid development, it plays havoc with attempts to insure accuracy, prevent rampant user customizations, provide version control, have a workable update and maintenance process, and institute effective information system controls. Someone ought to do something.

This paper discusses what might be done and who can do it. Spreadsheets (there are over a dozen of them now, in two distinct classes) should be thought of as a class of computer programming languages, and critically evaluated for their suitability for application development. This paper proposes six features that the designers of spreadsheet computer programming languages should include in their products, and explores how they are handled—and might be implemented—in the dominant Microsoft Excel spreadsheet and in the “software-as-a-service” Google Spreadsheet. Software-as-a-Service spreadsheets seem to have a technical advantage in delivering these features. The paper also explains how to provide “pretty good protection” of source code in the dominant Excel spreadsheet.

If the six application development features were implemented in spreadsheet computer programming languages, it could revolutionize the development of application software in spreadsheets. This has potential to substantially reduce the risks and problems associated with spreadsheets, reduce lifecycle costs, allow effective information systems controls, and make spreadsheets suitable for high-control uses such as Sarbanes-Oxley compliance.

1.1. Literature Survey

I am unaware of any literature that discusses creating spreadsheet applications, or explores in any detail the points raised in this paper. [Bewig 2005] mentions users as distinct from developers, but focuses on what one might call ‘power developer’ skills. He touches on many

1.2. Structure of Paper

The paper begins (section 2) with a discussion of spreadsheet applications, and a proposed categorization into “intentional spreadsheet applications”, “de facto spreadsheet applications” and “accidental spreadsheet legacy applications”. This leads into section 3 which argues that spreadsheets should be thought of as a class of computer programming languages for writing application software. Computer scientists routinely evaluate computer programming languages for their suitability for various tasks, and section 4 of the paper concludes that spreadsheet languages would be much better suited for application programming if they prevented users from accessing the source code. Section 5 presents six application development features that are desirable in a spreadsheet computer programming language.

Section 6 discusses a new class of spreadsheet languages delivered over the web, and uses Google Spreadsheet as an example. Section 7 discusses the excellent prospects for these “software-as-a-service” spreadsheets to provide application development features, and sketches how these might be implemented for the case of Google Spreadsheet by adapting the existing sharing feature. Section 8 explores “local” spreadsheets, focusing on Microsoft Excel, and discusses Excel’s tolerably effective source code protection features, which are hindered by poor password encryption. Section 9 summarizes Excel protections and concludes that user access to source code is not an attribute of Excel but rather is a choice by the programmer. Section 10 discusses how to provide application development features in local spreadsheets. Conclusions and further research are discussed in section 11.

2. MOTIVATION: SPREADSHEET APPLICATION SOFTWARE

Application software is “programs written for a specific application to perform functions specified by end users” [Laudon and Laudon 2006]. Spreadsheet application software is a computer program written in a spreadsheet and provided to end users. I propose three categories for classifying spreadsheet application software:

**Intentional spreadsheet applications** are software developed in a spreadsheet and purposefully deployed to users. [Grossman, Mehrotra, and Özlük 2006] give three examples built by professional developers. These are a spreadsheet application for sale (dozens of worksheets in three workbooks totaling 70 MB); a spreadsheet application distributed to thousands of users in a multi-national company; and a spreadsheet decision support system provided by a consulting firm to a client organization. These spreadsheets are very much the product of traditional application development, where the developers selected the spreadsheet as the appropriate programming language. Intentional spreadsheet applications are undoubtedly constructed by casual programmers as well as professional developers.

**De facto spreadsheet applications** are spreadsheets constructed for personal use that are distributed to other people. There is widespread awareness of spreadsheets that are built for individual use that find their way onto other people’s computers. These spreadsheets can take on a life of their own, and present many well-known challenges such as risk of error, improper usage, inefficient updating and maintenance, inappropriate and uncontrolled modification, and so forth. These spreadsheets were not constructed with the intention of use by other people, but become applications by dint of distribution to users.

**Spreadsheet accidental legacy applications** are spreadsheets inherited by a person when they start a new job. [Grossman, Mehrotra, and Özlük 2006] present three examples, including one that led to a multi-million dollar financial loss. Industry is recognizing
spreadsheet legacy applications; [Gardner 2006] tells us that “a new business [of spreadsheet management] is being built around the millions of old spreadsheet applications that were developed months and years ago, but are still present in companies’ financial records”.

3. THE SPREADSHEET IS A COMPUTER PROGRAMMING LANGUAGE FOR WRITING APPLICATIONS

Spreadsheets are being used for application development, whether by professional or amateur developers, acting with intention or by accident. However, spreadsheets receive little attention and (sometimes little respect) for their use in this role. I believe that spreadsheets merit careful study as a powerful application development language.

3.1. Not an Application, but a Language for Developing Applications

The spreadsheet is commonly referred to as an “application”, akin to a word processor or an email client. I think it is a mistake to refer to the spreadsheet as an “application”. The spreadsheet should be considered a class of computer programming languages. Like most computer programming languages, a spreadsheet can be used to write applications that are delivered to users, as well as for personal productivity and other uses.

To be considered a class of computer programming language, there must be substantial choice, and there is. Wikipedia tells us there are over a dozen spreadsheet computer programming languages available. The dominant spreadsheet language is Microsoft Excel, which uses the traditional “local” software delivery model, where the spreadsheet product is installed on the user’s hard drive. Other local spreadsheet languages include OpenOffice Calc, Lotus 1-2-3, Quattro Pro, KOffice KSpread, and Gnumeric.

A new delivery model called “Software as a Service” (SaaS) runs a spreadsheet computer programming language on a remote server. Programmers and users interact with the spreadsheet language using a web browser. Google Spreadsheet gets the most press, but others include EditGrid, iRows, Simple Spreadsheet, ThinkFree Calc, wikiCalc and Zoho Sheet.

Because SaaS spreadsheets reside on a remote server rather than on the user’s hard drive, in principle the programmer has a high degree of control over who can access the spreadsheet and what they can see. A SaaS spreadsheet language may have advantages for restricting what user activities with the spreadsheet.

3.2. Strengths and Weaknesses of Spreadsheet Computer Programming Languages

Spreadsheet computer programming languages have many attractive programming features. Most prominent is a beguiling accessibility that allows people who are not professional programmers to quickly write software with little or no training. Spreadsheets are an excellent rapid development environment, and support collaborative interactions. Spreadsheet languages are attractive for application development because users are familiar with the interface and deployment is easy due to the ubiquity of Excel. Spreadsheet languages are attractive to users because they are easily extended and connected to user-built spreadsheets, although these features are a mixed blessing.

As with any class of programming languages, there are drawbacks. Errors are a concern, as they are in all software, and some question the wisdom of letting non-professionals program. The most significant drawback of a spreadsheet from the perspective of application programming is that in general the user can modify the source code. What this means is that the user can access and change the computer programming statements of the application that he is using, while he is using it!
4. **THE PRIZE: A SPREADSHEET COMPUTER PROGRAMMING LANGUAGE THAT IS GENUINELY SUITABLE FOR APPLICATION PROGRAMMING**

Traditional programming languages prohibit or at least make it difficult for the user to access or modify the source code. Compiled languages such as C++ provide an executable but not the source code, and the source code cannot easily be reverse engineered from the executable.

In contrast, spreadsheets make it easy for the user to mess about with the source code. The user receives not an executable, but a copy of the source code, along with an integrated user-friendly source code editor, non-cryptic programming statements, and an instantaneous code interpreter. It’s virtually an invitation to the user to modify the source code. Because other interpreted languages don’t provide this bundle of features, they are less vulnerable.

The spreadsheet application programmer faces unique challenges. It is easy for someone who receives a spreadsheet application to modify the underlying programming statements, by intention or by accident. This can lead to errors. It confuses any versioning structure the programmer might have, and greatly hinders maintenance and adoption of new versions. It makes a mockery of any information system controls and is a root cause of “the subversive spreadsheet”. For financial organizations it is inconsistent with Sarbanes-Oxley compliance.

There would be great benefit if spreadsheet application programmers could enjoy the same level of control that, say, C++ application programmers have in the distribution and control of their source code. This would allow developers to use spreadsheet languages with much greater confidence, and permit their use in high-security environments such as Sarbanes-Oxley. This would be a great prize. The question now become, what features are needed to obtain this prize, and which are available in existing spreadsheets?

5. **THE CHALLENGE: SPREADSHEET LANGUAGE FEATURES TO SUPPORT APPLICATION DEVELOPMENT**

To support the use of the spreadsheet computer programming language for writing applications, the publishers of spreadsheet languages need to provide certain features. The ideal approach would be to consider “application development features” during the design phase of the spreadsheet language. Failing this, the task becomes adding features to an existing language. There are six features, with only the first being absolutely necessary.

The essential feature is to (1) **protect source code from user changes**. This prevents users from accessing and modifying the source code. One way to think about this is to provide the ability to designate each cell as “full-access” or “display-access”. The user has full privileges over full-access cells, including the ability to view and edit cell contents. The user has minimal privileges over display-access cells, and can see displayed values but not cell contents, including when the spreadsheet is saved to a local hard drive. An enhanced feature would allow cells to be designated “no-access” where the user can’t see them. One can easily imagine other possibilities and extensions, but these are the key elements.

It is obviously desirable to (2) **prevent user customization** by prohibiting adding programming statements to the spreadsheet. This enhances application integrity and is essential for version control. More subtly, it is important to consider the possibility of linking a user spreadsheet into the application spreadsheet to create a local system; this might be a beneficial integration or a problematic rogue system. Therefore, the programmer should have the ability to (3) **restrict linking** user spreadsheets into the application. Finally, it would be helpful to (4) **prevent users from sharing the application** without the developer’s permission. This would solve the problem of proliferation.

It is desirable to (5) **provide easy control of protection features** using a well-designed user interface where the programmer can select the features he wants to use. Finally, it is important to (6) **prevent user tampering with protection features** by employing effective password...
protection, and avoiding simple circumventions such as the Excel copy-and-paste evasion discussed below.

6. **SaaS SPREADSHEETS: THE CASE OF GOOGLE SPREADSHEET**

A SaaS spreadsheet is resident on a server, and information is sent to the user’s computer only as appropriate, whereas with a local spreadsheet all information is present on the user’s computer. For this reason, SaaS spreadsheets seem advantageous for source code control, and it is argued that SaaS systems can provide excellent security [Wikipedia 2007]. SaaS security can be based on user accounts authenticated with log-in passwords over SSL connections, which extensive e-commerce experience shows to be very effective.

Since SaaS spreadsheets are new, I will use Google Spreadsheet (docs.google.com) as an example. The capabilities of the Google Spreadsheet beta release pale in comparison to Excel. However, Google spreadsheet is not intended to be a copy of Excel on the internet; instead it provides innovative “web-centric” features, including simultaneous editing by two users and API “hooks” to connect the spreadsheet to other applications.

Google Spreadsheets controls access through log-in passwords. It has a sharing feature that a programmer can use to grant access to other people. A “collaborator” is granted full control of the spreadsheet. A “viewer” is granted the ability to see spreadsheet but not edit it. A viewer cannot change nor copy cell contents. If the a viewer copies a range of cells into Excel, only the values are copied, not the cell formulas. A viewer can observe the complete cell contents (including formulas) as shown in the screenshot below, where cell A3 is partially displayed but its contents are observable by hovering the cursor over a field in the bottom bar:

At the time of writing, information in the spreadsheet can not be hidden, and the viewer can always see the source code.

7. **FEATURES REQUIRED IN SaaS SPREADSHEETS TO SUPPORT APPLICATION DEVELOPMENT: THE CASE OF GOOGLE SPREADSHEET**

If the source code were protected, a SaaS spreadsheet could provide significant advantages for writing and distributing applications. The programmer could maintain on a server a master copy of the application that large numbers of users access remotely. A new version could be rolled out at the click of a button, with the old version safely locked away.

In Google Spreadsheet, a “sharing” feature controls the information that is sent to the user. In principle, Google Spreadsheet could provide an excellent facility for protecting source code (feature 1 in section 5) by creating a new sharing class, which I’ll call “limited-user”. In its simplest form, a limited-user would have the following privileges:

1. See all values displayed in the spreadsheet.
2. For cells containing data (text or numbers): the access control state is *full-access*. The user can observe complete cell contents and edit cell contents; the bottom bar detail is enabled.

3. For cells containing cell formulas: the access control state is *display-access*. The user can not observe cell contents nor edit the cell; double-clicking the cell and the bottom bar detail are disabled.

4. A copy of the spreadsheet saved locally contains values not cell contents.

A slightly more sophisticated approach would allow the programmer to *specify* data cells as full-access. This would allow the programmer to safeguard certain text and numeric values (such as labels and model parameters) while permitting the user to change input values.

Google’s invitation-only sharing system prevents users from sharing the application (feature 4). Google’s log-in password system seems suitable for preventing user tampering (feature 6) but ideally should be implemented using SSL. A *limited-user* could be prevented from writing cell formulas (feature 2) and linking to the spreadsheet (feature 3). It seems likely that a skilled user interface designer could allow easy control (feature 5).

It appears that a SaaS spreadsheet programming language could deploy source code protection that users could not evade, and in addition provide all six of the desired application programming features.

8. **LOCAL SPREADSHEETS: THE COMPLEX CASE OF MICROSOFT EXCEL**

Because of its ubiquity, Excel merits special attention. Excel has several features that can be used together to safeguard source code. Because the safeguards are unduly complex and some combinations of settings are insecure, it is worth reviewing them. Menu sequences in Excel 2003 are shown in normal font. Menu sequences in Excel 2007 are shown in *italics*.

8.1. **Cell Protection Formats: Lock and Hide Cells**

The Cell Protection Formats control whether a user can see or edit the source code in a cell. Cell Protection Formats are controlled in the Protection Tab of the Format Cells dialog box, accessed via Format\Cells…\Protection\ or Home\Cells\Format\Cells\Protection\. The Cell Protection formats are activated only when Sheet Protection (section 8.2) is enabled.

If a cell is formatted Locked, the user can not edit its source code; if unLocked the user can edit its source code. If the cell is formatted Hidden, the source code in a cell can not be seen (even in Formula View); if unHidden the source code can be seen.

When Sheet Protection is disabled, the Cell Protection Formats are turned off and all cells are unLocked and unHidden. This is the usual Excel behavior, where the user has the keys to the kingdom and can see and edit all source code.

The default cell format is Locked and unHidden, so when Sheet Protection is enabled, the standard behavior is that cell contents can be seen but not changed. Formatting a cell as unLocked and Hidden is not recommended, because it allows the user to edit a cell whose contents he can not see. Strongest control is obtained by formatting a cell as Locked and Hidden, which prevents the user from seeing or editing the source code.
8.2. Sheet Protection: Control Cell Selection and Activate Cell Protection Formats

Sheet Protection has two types of effects. First, it controls whether the Cell Protection Formats (section 8.1) are enabled. Second, it controls the ability of the user to select cells; format cells (including conditional formatting); insert and delete rows, columns and hyperlinks; use analytical tools; and edit objects and scenarios (plus it disables the precedence/dependence arrows on the formula auditing toolbar, which is not programmer-selectable). Sheet Protection is controlled in the Protect Sheet dialog box, accessed via Tools\Protect Sheet\ or Review\Changes\Protect Sheet\.

When Sheet Protection is enabled, the default settings are “Select locked cells” and “Select unlocked cells”. Disabling “Select locked cells” renders it impossible for the user to select a cell that he can not change (due to its being Locked), which reduces user frustration. There is a more subtle benefit as well. When the user navigates in the spreadsheet using the Enter key, Tab key, or Arrow keys, Excel cycles only through the unLocked cells. This is equivalent to a traditional application where the user can move the cursor only to data entry fields. This keeps the user’s attention focused on the cells he is permitted to edit, and facilitates navigation among these cells.

When “Select locked cells” is disabled, the user can not see the source code contained in locked cells while viewing the spreadsheet Normal View. However, in Formula View, the user can see (but not edit) cell formulas. (Note that Formula View is toggled with the keyboard shortcut ctrl + ` where “`” is the grave accent.) To insure that the user can not see the source code, disable “Select locked cells” and insure that Locked cells are also Hidden (section 8.1).

Note that if the worksheet is Protected with “Format cells” enabled, the user can access the Format Cells dialog box, which is the place where cells are Locked and Hidden. A devious user might think he could change the format of a cell to alter its Cell Protection Format to unLock or unHide it. Fortunately, when Sheet Protection is enabled, the \Protection\ tab within the Cell Formatting dialog box is not available. (Drat, foiled again!)

The programmer can secure Sheet Protection using a password (section 8.5).

8.3. Hiding Worksheets: Make Source Code Unavailable to Users

An effective technique for securing source code is to place it on worksheets that are Hidden and hence unavailable to the user. To Hide or unHide a worksheet, use Format\Sheet\Hide\ and Format\Sheet\Unhide\ or right-click worksheet tab\Hide\. The user can only learn about a hidden worksheet by using Format\Sheet\Unhide\. The programmer can use Workbook Protection (section 8.4) to prevent the user from unHiding any Hidden worksheets.

Although Visual Basic for Applications (VBA) which is built into Excel, and its big brother Visual Basic (a programming language often used for writing traditional applications) are outside the scope of this paper, it is worth mentioning that the programmer can use VBA to hide a worksheet. VBA even has the capability to make a worksheet “very hidden”. A “very hidden” worksheet cannot be discovered or unhidden using Format\Sheet\Unhide\ in Excel and can only be unHidden from within VBA.

8.4. Workbook Protection: Control Worksheets

Workbook Protection controls whether a user can modify the worksheets or resize windows in Excel. Workbook Protection is controlled in the Protect Workbook dialog box, accessed via Tools\Protection\Protect Workbook\ or Review\Changes\Protect Workbook\.
Enabling Structure means that the user can not move, delete, hide, unhide, rename, or insert new worksheets. Enabling Windows means that the user can not resize or move Windows. The ability to prevent users from unHiding worksheets is particularly important if the programmer has chosen to Hide worksheets.

The programmer can secure Workbook Protection settings using a password (section 8.5).

8.5. Excel Passwords: Smart Practice, But Not Robust Protection

The above-mentioned features to limit access to source code can be secured using a “workbook element” password. I recommend their use. Passwords will stymie the casual user, and likely provide adequate protection for most situations. However, a determined user can evade the password controls.

Excel has two types of passwords. The password to open a file uses high-quality encryption and is difficult to crack. Unfortunately, the passwords to protect workbook elements use very poor encryption. [McGimpsey 2004] has a clear explanation of the inherent weaknesses of these workbook element passwords prior to Excel 2007. Google on “crack Excel password” and you will see a mini-industry of nefarious tools. It is easy to obtain software that will quickly evade a workbook element password, and sometimes crack an open-file password.

Excel workbook element passwords should be viewed as a smart practice, providing sufficient source code protection for common situations. However, a moderately energetic user will be able to circumvent Excel workbook element passwords and gain access to the source code. This should be a concern for spreadsheets containing sensitive data, or requiring high source code integrity, such as those subject to Sarbanes-Oxley compliance.

9. PROTECTING SOURCE CODE IN MICROSOFT EXCEL

Excel has a set of source code protection features that work reasonably well provided they are carefully applied, and users do not take the effort to crack the workbook element passwords. Excel protections may be under-utilized, perhaps because they are so cumbersome to use. Excel source code protection recommendations are summarized below.

9.1. Recommendations for Source Code Control in Microsoft Excel

Spreadsheet application programmers can safeguard Excel source code by taking all of the following steps before deployment of a spreadsheet application to users.

Cell Protection Formatting
The default cell protection is Locked and unHidden. To obtain the strongest protection, the programmer should change the Protection Format of all cells in the spreadsheet:

- Empty cells: Format as Locked and Hidden.
- Cells containing source code: Format as Locked and Hidden.
- Cells containing data entry fields: Format as unLocked and unHidden.

Sheet Protection
The programmer should protect the worksheet. This is required to activate the Cell Protection Formats. When protecting the worksheet, enable the minimum necessary access. Under “Allow all users of this worksheet to:” the programmer should:

- enable (check) “Select unlocked cells”
- disable (uncheck) all other options
Hide Worksheets
Hide any worksheets that do not contain inputs or outputs intended for the user.

Workbook Protection
The programmer should protect the workbook. Enable “Structure”.

Passwords
Use password protection for the elements that allow it, which are Sheet Protection and Workbook Protection. These passwords can be circumvented using readily available software.

9.2. Make Copying of Hidden Cells Difficult

There are many choices and options when protecting Excel source code. In some circumstances, it is possible for a user to evade protections. It works like this. The user selects two input cells while holding down the shift key, which selects a range of cells. This range includes the two input cells, plus all cells in the resulting rectangular range including any protected cells. With certain sets of protection settings, a copy-and-paste will reveal the source code. Section 9.1 is so detailed and specific in part to prevent copy-and-paste evasions.

It is smart practice to design the application user interface to prevent the possibility of this evasion. Place all inputs are in a common location, with no source code in the “rectangle” between any selectable input cells. One way to do this is to have a distinct input module. Better yet, place all cell formulas on a separate worksheet and Hide it.

9.3. Excel Provides “Pretty Good Protection”, or Lock the Door When Leaving Home

Although cumbersome to use, Excel provides what might be called “pretty good protection” of spreadsheet source code.

Using the Excel protections with workbook element passwords is similar to locking the door to your home while hiding a key under the flowerpot. The door is locked, but an enterprising person can find the key and gain entry. This is fine if you live in a good neighborhood or have little that is worth stealing, but is unwise if you live in a rough part of town or have an overflowing treasure chest in the front hall.

Failure to use the workbook element passwords is like leaving your front door closed but unlocked. Failure to use the protections at all is like leaving your front door swinging in the wind: it’s an invitation to improper entry.

The implications are clear: User access to the source code is not an attribute of the Excel spreadsheet, but is a choice by the programmer. Accepting the default of full access is a choice, not a given. The programmer could choose to protect the code.

10. FEATURES REQUIRED IN LOCAL SPREADSHEETS TO SUPPORT APPLICATION DEVELOPMENT: THE CASE OF MICROSOFT EXCEL

In aggregate, Excel can be considered a “pretty good” language for application development in situations where users are generally compliant. However, it could be much better.

Section 5 discusses six spreadsheet language features to support application programming. The essential element of source code control (feature 1), including locking all cells to prevent user customization (feature 2) is discussed in sections 8 and 9. Unfortunately, because a user can easily link his own spreadsheet into an Excel application, it seems inherently difficult to prevent linking (feature 3). Preventing users sharing the application (feature 4) also seems difficult to do through technical means, although strong managerial policies might be helpful.
The Excel protections are cumbersome and confusing to use. To obtain full protection requires the programmer to use four different tools, each with multiple combinations of settings and questionable defaults, plus two different passwords. Excel needs a purposefully designed source code protection tool (feature 5). This would include a new user interface that makes it simple to enable all protections necessary for complete protection, safeguarded by a single password. Any such facility should be carefully tested for evasions.

Passwords for source code control (feature 6) in local spreadsheets are, at least in theory, problematic. Securing the source code in a local spreadsheet is in some sense more difficult than securing SaaS source code because the password hash or some other “secret” must be resident in the spreadsheet file. Even an excellent encryption scheme is (at least “in theory”) vulnerable to the depredations of the crafty people who have cracked other encryption schemes, including expensively-developed protections used for DVDs.

Despite this theoretical difficulty, it is likely that the shoddy encryption system currently used for Excel workbook element passwords [McGimpsey 2004] could be greatly improved. Increasing the key space from its current 194,560 to something much larger (e.g., 128 bits) would slow password cracking software, perhaps to the point of impracticability.

11. CONCLUSIONS AND FURTHER RESEARCH

This paper explores a number of ideas around spreadsheets. The main points are 1) spreadsheets are a computer programming language used for application development, 2) that this is a good thing, and 3) it would be a better thing if spreadsheet languages were designed to provide six application development features. The paper explores the status and prospects of application development features in a local spreadsheet (Microsoft Excel) and a software-as-a-service spreadsheet (Google Spreadsheet).

It is desirable that designers of spreadsheet computer programming languages incorporate application development features into their products. This is different from “bolting-on” poorly integrated protection features, as seems to be the case with Excel. I propose design changes to a SaaS and a local spreadsheet. Although the proposed design changes might appear to be simple, making changes to complex software products is never easy and might not be feasible. It is better to incorporate design-for-application-development in the early stage of creating a spreadsheet computer programming language.

Excel provides adequate protection of source code, provided the programmer chooses to use it, although the protections can be defeated by a determined user who applies password-cracking software. Version control will be improved by source code protection, but because users will be able to write their own spreadsheets that hook into the application, it will still be a challenge. Because Excel is a local spreadsheet, the problem of proliferation will persist.

Google Spreadsheet provides no protection of source code. However, the Software-as-a-Service spreadsheet delivery model has excellent potential for providing a strong application development platform, because it is possible (at least in principle) to provide the complete package of application development features. It is to be hoped that researchers or spreadsheet publishers will implement these features so that we can experiment with them and learn more.

Spreadsheet “lock box” approaches merit attention. For example, a service provided by Risk Integrated (www.riskintegrated.com) accepts an Excel file for deposit in a secure location, and runs it for clients who access inputs and outputs over the internet [Jafry et al 2006, 2007]. It provides all six spreadsheet application development features (section 5), plus change logging, audit trails, Monte Carlo simulation, and more.

A number of research opportunities regarding spreadsheet applications are emerging. What programming and software engineering practices make sense for building spreadsheet applications? For example, is there an alternative to the messy and risky nested IF constructs?
used for contingent logic? Can spreadsheet files be treated like subroutines, and sensibly linked into a full application, as is done with compiled software?

Handling data for a spreadsheet application has a host of issues. How can a programmer insure that users enter meaningful data? How can a manager insure that multiple people use the same data? How can an analyst efficiently run multiple large data sets through a spreadsheet application? The possibility of data sets with different numbers of rows and columns is but one complicating factor. These problems might soluble through a combination spreadsheet language features, spreadsheet programming practice, and user practice.

There are over a dozen spreadsheet languages, and it would be useful to explore and compare the application development features in each of them.

This paper focuses on purposeful applications. What can be done regarding de facto applications? Is there a migration path to something more robust? Can we enhance the quality (or slow the decay over time) of accidental spreadsheet legacy applications?

We would greatly benefit from empirical research on all aspects of spreadsheet application development, usage, and evolution. The vast world of spreadsheet users and developers will provide many gratifying surprises and opportunities to those who seek.

ACKNOWLEDGEMENTS

I am grateful to JP Allen, Steve Alter, Greg Benson, Mike Middleton, and Terence Parr at the University of San Francisco for their insights. Two referees provided valuable inspiration. Any errors of omission or commission are the sole responsibility of the author.

REFERENCES


Patrick Kemmis
Giles Thomas
Resolver Systems Ltd
17a Clerkenwell Road, London, EC1M 5RD, UK
info@resolversystems.com

ABSTRACT

We intend to demonstrate the innate problems with existing spreadsheet products and to show how to tackle these issues using a new type of spreadsheet program – Resolver. It addresses the issues head-on and thereby moves the 1980’s “VisiCalc paradigm” on to match the advances in computer languages and user requirements. Continuous display of the spreadsheet grid and the equivalent computer program, together with the ability to interact and add code through either interface, provides a number of new methodologies for spreadsheet development.

1. INTRODUCTION

Spreadsheets have many intrinsic problems stemming from the history of their development and the difficulties in changing the core user interface and coding language. The method of interaction has not significantly changed in over twenty years. The language used for their program code has not advanced with the developments in new dynamic computer languages.

Resolver Systems has developed a new type of spreadsheet product, which addresses many of the issues identified in this paper and offers a new method of interaction and development.

Developing robust spreadsheets is very hard using existing products. The rigorous testing of the computer software industry is absent from almost all business user-developed spreadsheets. The need to change as data requirements and calculations evolve invariably leads to their deterioration over time. The ever increasing size of data manipulated in spreadsheets only exacerbates the problems. Further, as spreadsheets are used more and more to manipulate data loaded from external databases or feeds, the inherent limitations of existing approaches are becoming clearer, and the need for alternative approaches more obvious.

One possible solution is for the problem spreadsheets to be passed to the IT departments for replacement or enhancement. However, the disconnect between business user-developed spreadsheets and IT developed programs has made it hard for solutions to cross this barrier. This has lead to spreadsheets continuing to be used well beyond their being “fit for purpose”, and reluctance by IT departments to take on the conversion of spreadsheet solutions where they were not involved from the beginning.

It is proposed that these problems are reduced when the underlying program code of a spreadsheet is made visible for business or IT users to interact with, and enhance.
2. RESOLVER

Resolver is a new generation spreadsheet product which aims to tackle the problems identified in this paper and allow for robust and reliable spreadsheets to be developed by non IT people, while always providing a transparent bridge into an IT environment. This is achieved by changing the standard spreadsheet user interface and the underlying coding structure of the spreadsheets.

As advanced users are well aware, all spreadsheets are in reality computer programs; Resolver simply exposes to the user that underlying program alongside the traditional spreadsheet “grid”. Resolver converts the spreadsheet's structure and formulae into a readable, easily-understood sequential computer program which is executed every time the spreadsheet is recalculated.

The user can input data and formulae through the traditional grid view (using normal spreadsheet data and formulae) or as user code in the “coding pane” by entering additional user defined functions and code in sections interleaved around the code generated by Resolver from the formulae and data entered into the grid. Code entered in this way can also be used by formulae entered into cells on the grid. The section where the user code is added determines its interaction with the formulae entered into the grid.

The displayed program, updated in real-time as the spreadsheet is modified, allows the user to modify and extend the program in a structured manner. It contains everything required to define the worksheets, the formatting of the cells, the data and formulae in the cells and the user code directly entered into the program code.

Resolver further displays an output window, which allows the user to follow the execution of their code, for example by printing to trace execution and to display intermediate calculation results, or by examining stack traces showing the details of any error's location.
2.1 The Coding Sections

There are six main sections of program code which are executed by Resolver in the following order for every recalculation of a workbook. Three are system defined and are not editable and the other three (also thought of as "set-up", "tweak" & "polish") are user input and maintained.

Import statements & Worksheet creation - The code here loads appropriate background data, libraries and databases (where created through the spreadsheet user interface) and creates a workbook and worksheets for results to go into. This code is not editable by the user.

Pre-constants user code – The first section in the code where the user can enter their own code. It allows users to "set-up" their workbook as entering Python code in this section enables data from external sources to be uploaded and unique, personalised functions to be incorporated. External libraries can be loaded here from central repositories or internet sites. Bespoke links can also be established to external databases instead of using the drop-down menus to create more standard database connections. Code executed here cannot reference data or formulae entered into the grid as they have not yet been defined in the program code.

Constants and formatting - This section of code is generated by Resolver from cells containing just data constants (such as text or numbers but not formulae) and from all the formatting defined at cell, row or column level in each worksheet of the workbook. As data is held completely separately from the formulae, this aids locking the formulae and not the data making it easy to produce a secure data input file. This code is not editable by the user.

Pre-formulae user code - Entering Python code in this section enables users to "tweak" their calculations as it allows access to constants, but values to be defined by formulae have not yet been calculated. User code in this section can reference the constants entered into the worksheet grid, but values to be defined by formulae in the grid have not yet been calculated in each recalculation and therefore cannot be used. However functions or calculations defined in the pre-constants user code can be used.

Formula code - This section of code is generated by Resolver from the formulae entered into the worksheet grid. The formulae in the grid are converted where necessary to a Python compatible syntax but there is always a one-to-one relationship between formulae typed into grid cells and formulae in this section of the program code. This code is not editable by the user.

Post-formula user code - Entering Python code in this window can "polish" the result by highlighting, isolating and/or manipulating outputs for re-use elsewhere. User code in this section can use any data or results on the grid or use any functions already defined. However it cannot be used by any formulae on the grid and therefore is best used for final changes to the presentation of data on the grid or for exporting data to external files or databases etc.
2.2 Interaction between formulae typed in the grid and user code

Code and functions added as user code can be used by formulae entered in the grid. Similarly formulae or constants in the grid can be referenced by user code. Additional libraries and functions can be added through the code view which can be used both within the code view itself and from the cell formulae. This tightly binds the cell formulae and the code view together into a single coherent program. Defining a function in the user code allows that function to be used in a grid cell or in other user code.

An example function to add on Value Added Tax (17.5% in the UK) is shown below. This simple function - withVAT() – takes an amount passed to it (which could be a cell reference) and returns the amount after adding VAT.

```python
def withVAT(amount):
    return amount*1.175
```

This function can either be used in a cell:

If Cell A1 = 100 then typing “=withVAT(A1)” into Cell A2 would show 117.5

Or directly in user code:

```python
vatTotal = withVAT(workbook[“Sheet1”].A2.Value)
```

Resolver executes the generated program (with the user's modifications and extensions) each time the user changes it, either via the spreadsheet view or directly in the code view, and then takes the results of this execution and displays them back as values in the spreadsheet or through interactions with external databases and services.

It allows the user to export the customised program - that is, both what was generated from their work in the grid, along with any code they have written - as code that can be executed as part of a traditional computer program. There is always a one-to-one relationship between the code displayed and the code executed to populate the spreadsheet.

2.3 Program Code Language

Resolver uses the Python programming language [van Rossum, 1995] for the program code, which is an easily learnable and highly expressive language with a rich standard library; it has been successfully used in a wide variety of applications, in both businesses and universities. Its clean syntax allows newcomers to become productive rapidly, while its more sophisticated features and built-in test framework allow experienced developers to build powerful tools which can quickly adapt to changing requirements. It has been enhanced and extended to meet the needs of a spreadsheet environment. The clear layout and syntax is in stark contrast to the complexities required to code in standard spreadsheet scripting languages.

The formula code retains compatibility with standard spreadsheet syntax but still permits the full range of Python expressions to be used offering a rich coding environment.

The formulae in the grid are converted to standard Python syntax in the program code.
Kemmis & Thomas
2.4 The interaction of formulae & data in the grid with user & other program code

In the very simple example below, numbers have been typed into two cells (B2, B3) and a formula into cell B4. The notes describe how the data and formula input on the grid are used in the different sections of the program code and the result is displayed back on the grid.

**THE GRID**

<table>
<thead>
<tr>
<th>THE GRID</th>
<th>Notes</th>
<th>PROGRAM CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Grid input</strong></td>
<td></td>
<td><strong>Code view and order of execution</strong></td>
</tr>
<tr>
<td>Constants</td>
<td></td>
<td>Import statements and worksheet creation</td>
</tr>
<tr>
<td>Cell A1 = “Title”</td>
<td></td>
<td>System defined code (no user code).</td>
</tr>
<tr>
<td>Cell B2 = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cell B3 = 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formulae</td>
<td></td>
<td>User code: Pre-constants</td>
</tr>
<tr>
<td>Cell B4 = B2 + B3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Grid display**

<table>
<thead>
<tr>
<th>Grid display</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**Notes**

1. User defined functions or constants in the Pre-constants user code section are available for use by formulae entered on the grid.
2. Text and numbers entered as constants on the grid are converted automatically to program code.
3. User defined functions or constants in the Pre-formulae user code section are available for use by entered on the grid.
4. Formulae entered on the grid are converted automatically to program code.
5. The results of all stages of the program code are held in a separate copy of the grid and displayed as required.
3. DEVELOPMENT METHODOLOGIES

3.1 Loading and manipulating data from external databases

Where traditional spreadsheets are used with data supplied from external databases or other sources, the standard approach of many users is to paste the required data once into the spreadsheet and trust that it will be copied over again, should the data in the database change.

The alternative is to write a considerable amount of code to handle the data loading and later updates. However, as the shape and size of the data change over time, the code to load the data and any formulae on the worksheets all have to work reliably with the data as it changes - a not inconsiderable challenge. As data shape, volume and calculation requirements change, there is more and more opportunity for the three to get out of step - leading to errors.

Entire database tables, or the results of arbitrary SQL queries, can be imported directly into Resolver documents as new worksheets, which can immediately be used in calculations. If necessary, Resolver can update the worksheet in real time as the underlying dataset changes. Because there is no need for the user to write code themselves, this allows for sophisticated data analysis without any assistance from IT departments, raising the bar for end-user computing. Calculations can be performed based on database field references and across all rows of data ensuring that the code continues to work reliably.

3.2 Combining data from multiple sources

Data might logically belong in the system or model being created in the spreadsheet, but needs to be edited by more than one user simultaneously. For example, a sales director might want up-to-date information on the sales made by a number of salespeople. With a traditional spreadsheet, they might create a template for each salesperson, email a copy to each of them, and ask them to email back a completed one on a regular basis. The director would then create a summary spreadsheet with appropriate formulae to aggregate the data across all of the salespeople, and as the completed templates came back, they would copy the data over to their summary. This is clearly a very manual and error-prone system.

Resolver features Shared Worksheets, which can be simultaneously used by multiple users. Because different documents can use the same Shared Worksheet, the same data can be used for different purposes in different contexts; as the data is updated in one location, all other users can see the changes cascade through their own documents. Similarly, shared worksheets can allow a single spreadsheet to collate information from multiple sources.

3.3 Worksheet level formulae

Frequently worksheets are used to combine data from other worksheets; for example, the numbers in a worksheet containing account balances might be multiplied by those in equivalent positions in a worksheet containing percentage interest rates, to provide an identically-structured worksheet containing interest amounts. In a traditional spreadsheet program, specifying a worksheet like this requires a formula to be near-replicated across the entire resulting sheet, and although spreadsheets provide functionality to make this replication easier, errors can be introduced into the resulting grids of formulae easily - for example, a new row added to the end of the first worksheet and to the second worksheet will not cause a new row to be added to the end of the third.
Resolver features formulae for worksheets as well as for cells; this allows a single formula to fill a worksheet with values calculated from one or more other worksheets. Worksheets calculated in this way will be updated correctly as the input data grows or shrinks; this method also diminishes the incidence of single cells with incorrect formulae, since there is no need to mechanically specify the formula for every cell in the worksheet.

3.4 Working with multi-dimensional data

Standard spreadsheet products support pivot tables and other simple database methods of interacting with data where it is held against a number of dimensions. However pivot tables are notorious for being difficult to maintain and to use the resulting data.

Lotus developed a successor to Lotus 123 in the 1980s, called Improv [Zisman, 1993]; a clone called Quantrix [Rubash, A.R., Rubash M.A., 2005] is still available today. It is based on multi-dimensional data structures defined by the user, starting with a single cell, and requires the user to define all the data structures first and then populate the data afterwards. Where the requirement is to analyse pre-defined data structures such as financial Budgets and Forecasts, then this provides an easy way to “slice and dice” data across several dimensions. However, we would assert this is not how most spreadsheet users think and it has not proved to be a mainstream solution to developing general purpose robust spreadsheets. Users in general don't like having to define everything up-front, and prefer a more iterative approach to developing their spreadsheet.

Resolver allows data to be kept on separate sheets as though in a database and referenced via header rows permitting a multi-dimensional analysis to be performed as though using a database without requiring the overhead of setting up and maintaining an actual database.

3.5 Transfer of spreadsheet models to full programming languages

Not everything that is possible in a programming language is possible with a traditional spreadsheet; existing spreadsheet products provide separate scripting facilities that allow limited interaction with the spreadsheet model, but there is invariably a conceptual gap between the two parts of the system.

Frequently a spreadsheet used to perform a calculation must be rewritten from scratch in a traditional programming language, if it is to be reused. Business users are poor at documenting their spreadsheets and IT departments are generally unwilling to undertake the detailed analysis required to fully disassemble the code from a spreadsheet that has often evolved over time in an unstructured manner. The time required for the business user to prepare a specification, and the inevitable misunderstandings when communicating that specification from business to IT, ensures that spreadsheets continue providing strategic solutions far beyond their useful life.

Alternatively, for example, Savvysoft, a software company based in New York, has produced a product, now called Calc4Web, [Savvysoft, 2007] that converts existing spreadsheets into C++ computer code. This is great for one-off conversions of complex spreadsheet models into computer programs, but there is no seamless integration of the two, and it is not really appropriate for business user interaction with the code produced. While developing the spreadsheet model it is impossible to see the evolving computer code and the C++ code would be impenetrable to any business users anyway.
Since a Resolver document is simultaneously a spreadsheet and a computer program, the numerical data remains separate from the logic which operates on it. Resolver spreadsheets make it possible for business users to develop a spreadsheet as usual, and for IT to instantly extract their algorithms and use them in more complex systems. Equally, it allows IT departments to make live data from existing systems available to end-user spreadsheets on demand.

3.6 Creation of standard calculation libraries

Spreadsheet models are rarely re-used elsewhere in other spreadsheets due to the intrinsic difficulties in changing links and in documenting a spreadsheet. Often models are re-built from scratch as this is more reliable. This leads to tremendous waste and potential unintended changes. This should not be underestimated as a problem. There are many examples of users recreating the same calculations over and over again because there is no practical method of issuing standard libraries based directly on other users’ calculation models except by going through an IT development exercise [Howard, 2005].

Resolver program code derived from a spreadsheet model can be saved as a library for use in a central repository.

Resolver spreadsheets can instantly import and use any external components written for Microsoft's popular .NET platform, and many of those written for Python. This gives Resolver spreadsheets unparalleled flexibility; they can be integrated into other systems with great ease, and allow organisations to continue to make use of their existing investment in spreadsheet models.

3.7 Publishing to the intranet / internet

Many departments using spreadsheets need to publish the results on the internet or intranet for internal or external consumption. There is no straightforward mechanism to publish a traditional spreadsheet to a webpage.

Online spreadsheets such as Google Docs allow for easier sharing and some publishing of data, and this is great for small businesses on multiple sites or for remote access. But the user is dependent on an external online service being always available, and it does not integrate well with local spreadsheets and databases containing potentially sensitive data. This type of solution is not really appropriate for organisations requiring secure and reliable sophisticated spreadsheets where performance and control are important.

Resolver offers a web server module where any spreadsheet can be converted to a form that is immediately publishable and further provides an interaction allowing the data to be changed and re-displayed.

3.8 Locking down spreadsheets for data input only

There are many cases such as Budget or Forecast collection where it is imperative that the user does not change the formulae or structure of a spreadsheet so that the calculations remain correct and the central consolidation of data remains complete. This is possible with traditional spreadsheets, although the process to selectively lock cells is very cumbersome and there is no mechanism to easily extract the data and not the formulae from a worksheet.
The separation of data and formulae in Resolver permits the easy lockdown of the formulae allowing reliable use of the spreadsheet for data gathering while ensuring the ongoing reliability of the formulae. This also allows for the easy extraction of the data only for consolidation elsewhere as the constants are immediately available for consolidation.

3.9 Data Typing

A source of many errors in traditional spreadsheets has been through entering the wrong type of data in a cell or from a database load.

Text, dates, numbers or indeed integers all have particular needs for display and processing correctly and can have peculiar effects on formulae and therefore the results of a calculation.

There is no straightforward method to determine visually the type of data held in a set of cells. Indeed in traditional spreadsheets, all that can be determined is the display format and some properties of the data in each cell. This generally has to be determined by writing code or using the Goto function to select text or numbers.

Resolver can indicate the type of data held in every cell on a worksheet and further it can force a cell to hold and process only one type of data independently of how the data is actually displayed.

For example, numbers loaded from a database may represent codes that should be treated and sorted like text, or text that looks similar to dates needs to be processed in a single consistent manner. The ability to separate the internal processing of data from the display is key to robust and consistent treatment of data from external sources.

3.10 Ad-hoc changes for what-if analyses

Ad-hoc data analysis by changing values in the spreadsheet can be very useful, but sometimes changing the spreadsheet can be dangerous, as the ad-hoc changes can be accidentally saved. There is no mechanism to change the values used in a calculation while retaining the original data.

Resolver permits code created values to overwrite a user input value in a grid cell. This allows for the selective changing of data inputs for what-if analyses while still preserving the original user input data.

3.11 Data bounds

Because only part of a worksheet is normally visible at any one time, and there is no normal indication of whether or not there is any data "off-screen" within a worksheet, errors can be introduced by spreadsheets not taking account of data that happens to have been invisible when the user was writing part of the model. For example, if a user was summing up figures from a number of rows, a row that happened to be scrolled down below the edge of the viewable area might be missed out because there was no obvious indication that it was there. Where scripting languages are used then the problems are even worse as there is no obvious link between the size of the data and the range allowed for in the scripting code.
Resolver includes an indicator of the rectangular area bounded by the data and formulae and the coding language naturally operates over all rows or columns within the bounded area.

4. RESOLVER CURRENT STATUS

4.1 Development status

Resolver is designed to emulate standard spreadsheets as far as possible, within the new programming interface, through syntax compatibility for formulae typed in to the grid directly. It has been in development for some eighteen months and the list of supported functions is growing all the time.

Resolver retains compatibility with traditional spreadsheets as much as possible; for example, formulae entered into the grid uses a superset of the syntax users will already be familiar with.

At the same time, the formula syntax also supports features that reflect its Python heritage - cells can contain not just numbers or text, but also lists, dictionaries, arbitrary .NET objects, and - for the truly adventurous - functions, including lambda expressions.

The code in the code view is Python pure and simple; formulae are rewritten appropriately when they are compiled down into the formula code section.

Most of the functions with which users will be familiar - such as SUM, IF, and COUNTIF - are currently supported, and the list is growing. These have been augmented by additional functions providing a rich set of additional features such as full database access with real-time updates, access to financial data feeds such as Bloomberg, and the ability to interact with these data sets through database functionality within Resolver.

Further, the use of external publicly available libraries within the Python and the .NET environments provides further functionality.

5. CONCLUSION

No previous product has been really successful at combining existing spreadsheet practices with a more robust solution which can make best use of good computer software methodologies and real-time integration with the outside data world.

What is required is a product for the ordinary spreadsheet user that allows them to manage their own development of reliable spreadsheets using best practice developed in-house or externally. It needs to allow the power user to interact with the spreadsheet using either in-cell formulae or modern simple computer code fully integrated into a whole. The IT developer should be able to work with the same code developed by the business users without any further translation or conversion so that errors are minimised and reliability is maximised.
No previous product has combined these enhancements into one integrated solution.

Resolver as an end-user tool sits midway between traditional spreadsheets and full business intelligence tools. It still provides the familiar spreadsheet environment which business users are so familiar with, while also offering a program code view of the same data and formulae with the robust database interfaces offered by sophisticated analysis tools.

Using Resolver raises the bar of what can be developed without IT involvement, while at the same time providing an easy route for moving user-developed solutions into a full IT managed environment. It makes it possible to introduce new methodologies to spreadsheet development, where the data grid and computer language paradigms coalesce into one coherent approach.

6. REFERENCES


Fun Boy Three Were Wrong: it is what you do, not the way that you do it

Jocelyn Paine

www.j-paine.org/ and www.spreadsheet-factory.com/
popx@j-paine.org

ABSTRACT

I revisit some classic publications on modularity, to show what problems its pioneers wanted to solve. These problems occur with spreadsheets too; to recognise them may help us avoid them.

1. INTRODUCTION

Bananarama, and Fun Boy Three before them, and Sy Oliver before them, sang a song with the refrain “‘T ain’t what you do it's the way that you do it; That's what gets results”. They were wrong; but they weren't programmers. In programming, the maxim should be “It is what you do, not the way that you do it”. That's the essence of modularity.

I decided to talk about modularity because many people — including myself — assert that spreadsheets (and other programs) should be modular. Fewer can state with precision what this means, or why it's good. It's like Dilbert's boss, as described by AI researcher Michael Covington's excellent presentation How to Write More Clearly, Think More Clearly, and Learn Complex Material More Easily [Covington, 2002]. On slide 56, Covington says:

Dilbert’s boss wants an “object-oriented database”, but he doesn’t know what makes a thing a database, or what makes it object-oriented. He doesn’t know what he’s talking about!

By showing what problems the pioneers of modularity were trying to solve, I hope to explain why it's useful. They worked on modularity because they needed to manage the increasing complexity of software, and the growing number of software blunders, mistakes, and catastrophes. Perhaps we need it to manage the growing number of spreadsheet blunders, mistakes, and catastrophes.

More specifically, I reason as follows:

Excel has no support at all for modularity. Therefore, if we can see what problems led the pioneers to work on it, and we find such problems in our own spreadsheeting, we can conclude that Excel is inappropriate.

But if, nevertheless, we are forced to use a spreadsheet, we may look for other spreadsheets that aren't Excel and that use the ideas behind modularity to overcome such problems.

But if, nevertheless, we are forced to use Excel, we may look for tools that help us use these ideas to overcome such problems.
While if we are designing such tools, we ought definitely to know about these ideas.

And if we are forced to use Excel and we dont have such tools, knowing when modularity is needed may help us predict that problems will occur.

1.1 Content Of This Paper

This paper is organised as follows. Section 2 summarises the main ideas. If you're a computer scientist, you'll probably already know them, and can skip the rest of the paper. Section 3 is a small example, showing how they apply to spreadsheets. Section 4 looks at history, examining problems that inspired the researchers. Section 5 glances at structured programming and top-down design, a closely-related problem-solving technique. Section 6 is my conclusion.

2. KEY IDEAS

When designing a program, it's important to hide information about data structures, especially if they may need frequent redesign.

Such structures should be defined within a single module. This module should provide special operations with which other modules can access the data. These other modules should, indeed, not access it in any other way, even if the language provides such access and the programmer knows about it.

Indeed, to improve program correctness, well-designed programming languages will forbid access except through these operations.

When a programmer uses such a module, he or she needs to know only about these operations, but not about how the data is actually represented. This walls off the effect of changes: the module's author can now safely change the data representation — to make it smaller, or faster to index, or to log its use to a diagnostics file, or whatever — as long as the interface to the access operations is left unchanged.

In other words, it's what you do, not the way that you do it.

2.1 Terminology

A data type, viewed through its access operations, is called an abstract data type.

A programmer who uses the module need know only the interface to, or the specification or definition of, the abstract data type: i.e. the names of the access operations, the types of data they take and return, and what they do.

But the programmer does not need to know how the data is actually represented and how the access operations work. Thus, the data's implementation is hidden. This is the principle of information hiding.
A compiler for a language with abstract data types will be able to compile a module that uses abstract data types from other modules, as long as it knows their definitions. This is separate compilation.

Such a language supports independent development by members of programming teams. The programmer who is designing one module need only tell the other programmers what its interface is. As long as this remains unchanged, he or she can safely change the module's implementation at any time.

Here's an example interface definition, from Section 2.1, *Modules*, of Niklaus Wirth’s historical paper *Modula-2 and Oberon* [Wirth, 2006]:

```plaintext
definition Stacks;
  type Stack;
  procedure push( var s: Stack; x: real );
  procedure pop( var s: Stack ): real;
  procedure init( var s: Stack );
end Stacks
```

I should explain that “stacks” are frequently used as examples in papers about abstract data types. A stack is a data structure that represents a first-in first-out queue, like the stack of trays in a cafe. If you push item X onto a stack, and then item Y, and then Z, the top of the stack will be Z. Pop Z off, and the top will be Y; and so on. Stacks are popular in these discussions because they occur so widely: for example, in almost every implementation of functions and function calls.

At any rate, the essence of a stack is that you can push things onto it, and pop them off it. Together with an operation to create a new stack, this is, as Wirth says following that interface definition:

> ... exactly the information a user (client) of module Stacks needs to know. He must not be aware of the actual representation of stacks, which implementers may change even without notifying clients. 25 years ago, this water tight and efficient way of type and version consistency checking put Mesa and Modula-2 way ahead of their successors, including the popular Java and C++.

### 3. EXAMPLE

Imagine that we are planning a program to read details of a set of loans, calculate the monthly interest due on each, and report it in a nicely formatted table with one loan’s charges per column, each column headed by the original loan details. This program can be in C++, VBA, Java or whichever: it doesn't matter. Each loan is characterised by, amongst other information, an interest rate and a flag saying whether the interest is simple or compound.

Before starting the design, we decide to divide the program into three modules and allocate responsibility for each to a different programmer. The first module reads and stores the loan details. The second module calculates and stores the monthly interest charges. And the third module tabulates and prints these charges, headed by a description of the loan.
Alice, the programmer responsible for the first module decides to store the interest-type flag just as it arrives from the input menus: as one of the strings Simple or Compound. She tells the other programmers of her decision.

You can see where I'm going with this. The design decision to store this flag as it came in is fine; just until the programmers' boss tells them the company is going multinational, and now needs a version in every language from Afrikaans to Welsh. Bob, who has relied on this flag in numerous places while calculating interest; and Cath, who refers to it when printing the loan details, will both have to update their code. And how will they agree between themselves and with Alice about what happens with all the new languages?

To avoid this problem, we could implement “loan” as an abstract data type, exported from a “Loans” module. The interface would look like this:

```pascal
definition Loans;
type Loan;
procedure set_interest_rate( var l: Loan; rate: real );
procedure set_interest_type( var l: Loan; simple: boolean );
procedure interest_rate( var l: Loan ): real;
procedure interest_type( var l: Loan ): boolean;
end Loans
```

One member of the programming team would be asked to write the Loans module. He or she would provide a standard interface to the information about whether the interest is simple or compound. This would be defined by the procedure set_interest_rate which sets a “simple” flag to true or false, and the procedure interest_rate, which returns this flag.

It would be the responsibility of a separate programmer to write the input module. They would call set_interest_type, passing it true or false depending on whether the input string was Simple or Compound; or its Afrikaans, Armenian, ...Welsh equivalent.

And a third programmer would write the output module. They would call interest_type to get the flag, and print the corresponding string Simple or Compound in each loan description.

### 3.1 Application to spreadsheets

The problem with spreadsheets is that it is almost impossible to hide such details. We end up with a situation reminiscent of that which David Every [Every, 1999] describes of the first high-level languages:

Part of the problem with the first high-level languages is they could deal with only a few types of data. Programmers used those very primitive data types to try to construct everything. So often programmers used arrays of primitive types to describe more complex types -- since there was no "natural" representation of what they wanted, and they couldn't create them (easily). Programmers had to decode and recode these arrays in many different places in the code, and any errors would be catastrophic.
Note that in spreadsheets, there are two aspects of structuring data: within the cell and between cells. Imagine that we have an input sheet where the user enters details of each loan. Most Excel developers would probably allocate another sheet — perhaps hidden — for the monthly interest calculations, and a third sheet for the output tables. That’s fine; just until one needs to change the number of loans.

4. HISTORY

4.1 The 1960’s Software Crisis

Let me begin with a quote from Niklaus Wirth’s *Pascal and its Successors* [Wirth, 2002]:

> The other fact about the 1960s that is difficult to imagine today is the scarcity of computing resources. Computers with more than 8K of memory words and less than 10us for the execution of an instruction were called super-computers. No wonder it was mandatory for the compiler of a new language to generate at least equally dense and efficient code as its Fortran competitor. Every instruction counted, and, for example, generating sophisticated subroutine calls catering to hardly ever used recursion was considered an academic pastime. Index checking at run-time was judged to be a superfluous luxury. In this context, it was hard if not hopeless to compete against highly optimized Fortran compilers.

Yet, computing power grew with each year, and with it the demands on software and on programmers. Repeated failures and blunders of industrial products revealed the inherent difficulties of intellectually mastering the ever increasing complexity of the new artefacts.

(In this and other quotes, I’ve replaced the authors’ citation numbers by my own, which refer to the References section at the end of my paper.)

4.2 The 1970’s Software Crisis

Now I’ll turn to Wirth’s *Modula-2 and Oberon* [Wirth, 2006], from which I took the example module definition. The paper begins as follows:

> In the middle of the 1970s, the computing scene evolved around large computers. Programmers predominantly used time-shared “main frames” remotely via low-bandwidth (1200 b/s) lines and simple (“dumb”) terminals displaying 24 lines of up to 80 characters. Accordingly, interactivity was severely limited, and program development and testing was a time-consuming process. Yet, the power of computers — albeit tiny in comparison with modern devices — had grown considerably over the decade. Therefore the complexity of tasks, and thus that of programs had grown likewise. The notion of parallel processes had become a concern and made programming even more difficult. The limit of our intellectual capability seemed reached, and a noteworthy conference in 1968 gave birth to the term software crisis [Naur and Randell, 1968].

109
Small wonder, then, that hopes rested on the advent of better tools. They were seen in new programming languages, symbolic debuggers, and team management.

Little, it seems, had improved in the ten years. So what were the programming languages we had to work with?

Wirth continues by naming some languages of the era. Fortran still dominated scientific programming, and Cobol business data processing. PL/1 was IBM’s mammoth attempt to unite the two. Lisp was popular in academic AI. And Pascal was Wirth’s own creation, reflecting ideas on structured programming, which I’ll return to in Section 5. But, he says, none of the available languages were truly suitable for handling the ever growing complexity of computing tasks. What was missing? My next section title suggests one answer.

4.3 Support For Team Programming

In *Pascal and its Successors*, Wirth explains how he wanted to create a language “adequate for describing entire systems, from storage allocator to document editor, from process manager to compiler, and from display driver to graphics editor*. The language would be used to program the Lilith workstation, a successor to Xerox PARC’s Alto. Modules were introduced as a key feature that:

… catered for the urgent demands for programming in teams. Now it became possible to determine jointly a modular decomposition of the task and to agree on the interfaces of the planned system. Thereafter, the team members could proceed independently in implementing the parts assigned to them. This style is called modular programming.

As a skilled language designer and implementor, Wirth probably had little trouble writing a compiler to handle modules. But why would he want to?

4.4 Information Hiding

In Section 2.1, *Modules*, of *Modula-2 and Oberon*, Wirth says:

Modula’s solution was found in a second scoping structure, the module. … The module merely constitutes a wall around the local objects, through which only those objects are visible that are explicitly specified in an “export” or an “import” list. In other words, the wall makes every identifier declared within a module invisible outside, unless it occurs in the export list, and it makes every identifier declared in a surrounding module invisible inside, unless it occurs in the module’s import list. This definition makes sense, if modules are considered as nestable, and it represents the concept of information hiding as first postulated by D.L. Parnas in 1972 [Parnas, 1972].

Wirth is saying that Modula modules affect visibility. This is similar to how private procedures behave in VBA code modules. In a VBA project, you cannot have two functions with the same name in two different modules, if they are declared as public. But you can if they are both private. In this case, the module acts like a wall, through which other modules cannot see its private parts.
It might seem perverse to restrict oneself in this way. But with restriction comes freedom; to write whatever auxiliary functions you want in a module without worrying that their names will clash with those in other modules.

It also means that if programmers Alice and Box are collaborating, Alice can write whatever private functions she wants and not let Bob find out about them and start using them. That’s good, because if Bob started using them, he might come to rely on them. And if he relied on them, his code might suffer very badly when Alice decided she needed to change these private functions to do something else.

But why would she? Because she wants to change her design. This brings me to David Parnas’s concept of information hiding which Wirth says the module represents. Let’s turn to that.

4.5 What Should Be Hidden?

Parnas introduces his paper, *On the Criteria To Be Used in Decomposing Systems into Modules* [Parnas, 1972], by quoting a 1970 textbook on the design of system programs. The book recommends segmenting the project into clearly defined tasks, each corresponding to a distinct module. Each module has well-defined inputs and outputs. Each module can be tested independently, and it is easy to timetable programming tasks so that modules are coded and tested in the right order. And because the program is modular, when a bug occurs, it is easy to discover which module is responsible.

But, continues Parnas,

> Usually nothing is said about the criteria to be used in dividing the system into modules. This paper will discuss that issue and, by means of examples, suggest some criteria which can be used in decomposing a system into modules.

The meat of Parnas’s paper is a thought experiment in which he asks us to imagine that we’re planning a new program. We have to divide it into subtasks, and give each to a different programmer. What is the best way to do so, and why?

He considers a “Key Word In Context” index generator. This generates one particular kind of concordance index. Concordances are lists of the words in a text, usually alphabetical, showing where each word occurs. Literary scholars use them (or did before computers became so powerful) to characterise the literary style of a text, and discover, for example, that Shakespeare was really written by Bacon.

A Key Word In Context index indicates a word’s context by displaying the words on either side. Here, from Wikipedia, is an example formed by KWICing the two sentences “Wikipedia, the Free Encyclopedia” and “KWIC is an acronym for Key Word In Context, the most common format for concordance lines”. Note that the index contains as many copies of each sentence as are needed to align each of its words in the key position:

```
KWIC is an acronym for Key ...
... Word In Context, the most common format for
```
Fun Boy Three Were Wrong: it is what you do, not the way that you do it

Jocelyn Paine

... the most common format for concordance lines.
... is an acronym for Key Word In Context, the ...

Wikipedia, the Free Encyclopedia

What asks Parnas, is the best way to divide up this programming task? His argument has the same structure as my Loans example. One possible modularisation is by processing stage. Have one module that reads the text to be indexed, a next that does the circular shifting necessary to make the copies of each line, a third to build the index, and a fourth to output it nicely formatted. A master control module will also be needed to coordinate them. This, he says, is:

... a modularization in the sense meant by all proponents of modular programming. The system is divided into a number of modules with well-defined interfaces; each one is small enough and simple enough to be thoroughly understood and well programmed. Experiments on a small scale indicate that this is approximately the decomposition which would be proposed by most programmers for the task specified.

4.6 Modularise To Hide Difficult Design Decisions, Not By Processing Stage

Parnas then contrasts that “conventional” way of cutting up the project with a second, “unconventional” way. In this, there is one module that handles storage of the lines. Other modules can get at them only through functions that this module exports: they are not allowed to access the low-level representation’s bits and bytes.

This is a big difference. In the first modularisation, the input module, circular-shift module, index-building module, and output module all know about these bits and bytes, and hence will all have to be changed, perhaps drastically, if the representation is changed. As it might be, if for example the lines are to be stored on disc rather than in memory, or if the way characters are packed into words is changed to speed up processing.

Parnas concludes:

We have tried to demonstrate by these examples that it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others. Since, in most cases, design decisions transcend time of execution, modules will not correspond to steps in the processing.

4.7 Abstract Data Types

The key, therefore, is to hide information about data. This leads me back to Modula-2 and Oberon, where Wirth says:

Modula-2’s module can also be regarded as a representation of the concept of abstract data type postulated by B. Liskov in 1974 [Liskov and Zilles, 1974]. A module representing an abstract type exports the type, typically a record structured type, and the set of procedures and functions applicable to it. The type’s structure
remains invisible and inaccessible from the outside of the module. Such a type is called *opaque*.

I recommend Liskov’s historical account in *A History of CLU* [Liskov, 1992]. In Section 2, *Data Abstraction*, she explains how she became disenchanted with the papers on programming methodology because they were so nebulous about what modules were, and about how to modularise. She noticed that in many of these papers, the modules were defining data types. This led her to link modules to data types, and then to abstract data types.

Programmers would, she thought, find it easy to design in terms of abstract data types: both because they are used to deciding about data, but also because the notion of abstract data type could be defined precisely.

Liskov implemented these ideas in a language named CLU. The name, short for “cluster”, referred to a cluster of abstract data type access operations like that in my examples. These were its interface to the rest of the program; and that brings me back to team programming.

### 4.8 Interface Versus Implementation

Here is another quote from *Modula-2 and Oberon*, in which Wirth discusses Mesa, the language used to program Xerox PARC’s Alto workstation and one inspiration for Modula-2:

… global modules appear as the parts of a large system that are typically implemented by different people or teams. The key idea is that such teams design the interfaces of their parts together, and then can proceed with the implementations of the individual modules in relative isolation. To support this paradigm, Mesa’s module texts were split in two parts: The implementation part corresponds to the conventional “program”. The definition part is the summary of information about the exported objects, the module’s interface, and hence replaces the export list.

Please note again the importance given to coordinating programming by teams.

### 5. STRUCTURED PROGRAMMING, LEVELS OF ABSTRACTION, AND TOP-DOWN DESIGN

In my section on the 1970’s software crisis, I mentioned Pascal, Wirth’s own language created to reflect his ideas on structured programming. I want to say a little about that topic, because it is such a fundamental problem-solving tool. Wirth says as much in *Pascal and its Successors*:

The only solution lay in structuring programs, to let the programmer ignore the internal details of the pieces when assembling them into a larger whole. This school of thought was called Structured Programming [Dijkstra, 1972], and Pascal was designed explicitly to support this discipline. Its foundations reached far deeper than simply “programming without go to statements” as some people believed. It is more closely related to the top-down approach to problem solving.
Liskov says much the same thing in Section 2, *Data Abstraction*, of *A History of CLU*, worth reading for a nice account of the early work on programming methodology and of how different researchers’ ideas were interrelated:

> Not using gotos [Dijkstra, 1968] was a part of structured programming because the resulting program structures were easier to reason about, but the idea of reasoning about the program at one level using specifications of lower level components was much more fundamental. The notion of stepwise refinement as an approach to constructing programs was also a part of this movement [Wirth, 1971].

### 5.1 The Top-down Approach To Problem Solving

So what is the top-down approach? It’s a method of solving design problems by splitting them into different *levels of abstraction* or, one can equally say, *levels of description*.

You start by specifying your problem at the highest level of abstraction, in terms of concepts that, in this level’s universe of discourse, you will agree to regard as primitives. That is, you will postpone worrying about how they are to be implemented. Instead, you devote your efforts to specifying, as precisely as you can, how the thing you are designing is to be built from these primitives.

Once you have done so, you descend one level of abstraction to a new level, L2. The concepts that were primitives at the highest level L1 now have to be implemented in terms of new primitives at level L2. A very important part of the design task is to work out what these new primitives are. Generally speaking, you always want them to be “parsimonious”, both by themselves and in relation to each other. Think of words and phrases such as “clean”, “elegant”, “uncluttered”, “standing in clear relationship to”, and so on. Not only does this make the program easier to write, it makes the solution easier to document.

Once you have worked out the L2 primitives, you need to specify the L1 primitives in terms of them. Again, this specification must be as precise as it can; for it will become part of a program.

And then, you descend once more, to a new level. And you keep on going … until you have reached primitive operations which really are primitives, because they are provided either by your hardware (for example, addition and subtraction) or by your program libraries (for example built-in functions such as MIN, MAX, and CUMPRINC).

This is a powerful and profound problem-solving technique. I urge you to become familiar with it. A good non-technical account is given by Douglas Hofstadter in Chapter 10 of *Gödel, Escher, Bach: an Eternal Golden Braid* [Hofstadter, 1979].

Stepwise refinement is closely related to abstract data types. The types’ access operations — push, pop, set_interest_type — become primitives at one level, to be implemented in the level below.
6. CONCLUSION

As I wrote this paper, I found that I had two objectives. One was the objective I started with: to explain the programming problems that inspired the pioneers’ work on modularity.

The other arose as I reread their writings. Spreadsheets are programming languages. They are visual; but they are languages. And they are used for systems that are huge, elaborated, and involve collaboration within teams. Their authors need intellectual tools — ideas — to manage such complexity. These ideas ought to be supported by their programming language, namely the spreadsheet. And we ought to be teaching these ideas. One way to do so is to encourage spreadsheeters to read the early papers on programming. Many of them are close enough to the hardware and software to be fairly easily understood, particularly if the reader has programmed in VBA.

It seems to be assumed that “end users” must be enabled to program without being taught programming’s fundamental problem-solving techniques. This is odd. You would not expect to design and construct a hi-fi, or a car, or a banquet, or a garden, or a medical diagnosis, or a detective story, or a life drawing, without sound knowledge of your discipline’s fundamental problem-solving techniques. Why should spreadsheets be different?

I want to quote from Niklaus Wirth’s *Conclusions and Reflections*, from *Modula-2 and Oberon*:

… The incredible advances in hardware technology have exerted a profound influence on software development. Whereas they allowed systems to reach phenomenal performance, their influence on the discipline of programming have been rather detrimental as a whole. They have permitted software quality and performance to drop disastrously, because poor performance could easily be hidden behind faster hardware. In teaching, the notions of economizing memory space and processor cycles have become a thing apart. In fact, programming is hardly considered as a serious topic; it can be learnt by osmosis or, worse, by relying on extant program “libraries”.

This stands in stark contrast to the times of ALGOL and FORTRAN. Languages were to be precisely defined, the unambiguity to be proven; they were to be the foundation of a logical consistent framework for proving programs correct, and not only syntactically well-formed. Such an ambitious goal can be reached, only if the framework is sufficiently small and simple. By contrast, modern languages are constantly growing. Their size and complexity is simply beyond anything that might serve as a logical foundation. In fact, they elude human grasp. Manuals have reached dimensions that effectively discourage any search for enlightenment. As a consequence, programming is not learnt from rules and logical derivations, but rather by trial and error. The glory of interactivity helps.

The world at large seems to tolerate this development. Given the staggering hardware power, one can usually afford to be wasteful in time and space. The boundaries will be hit in other dimensions: usability and reliability.

Spreadsheets are also languages.
7. REFERENCES


Douglas Hofstadter (1979). Chapter 10, Levels of Description, and Computer Systems, of Gödel, Escher, Bach: an Eternal Golden Braid. The book was first published in 1979. There are later editions, including a “20th Anniversary Edition” with a new 23-page preface, but as far as I know, they don’t add anything to this chapter.


Peter Naur and Brian Randell (1968), Software Engineering, Nato Science Committee, Conference Report, (October 1968). The Report is online at homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF 3:56pm 13/6/07.

David Parnas (1972). On the Criteria to be used in Decomposing Systems into Modules, Communications of the ACM, Volume 15, (December 1972). www.acm.org/classics/may96/ 8:34pm 8/6/06.


Concerning the feasibility of example-driven modelling techniques

S. Thorne, D. Ball, Z. Lawson
UWIC and Cardiff University
SThorne@uwic.ac.uk, DBall@uwic.ac.uk

1.0 Introduction

We report on a series of experiments concerning the feasibility of example driven modelling. The main aim was to establish experimentally within an academic environment; the relationship between error and task complexity using a) Traditional spreadsheet modelling, b) example driven techniques. We report on the experimental design, sampling, research methods and the tasks set for both control and treatment groups. Analysis of the completed tasks allows comparison of several different variables. The experimental results compare the performance indicators for the treatment and control groups by comparing accuracy, experience, training, confidence measures, perceived difficulty and perceived completeness. The various results are thoroughly tested for statistical significance using: the Chi squared test, Fisher’s exact test for significance, Cochran’s Q test and McNemar’s test on difficulty.

1.1 Example-Driven Modelling

The principle concept of Example Driven Modelling (EDM) is to collect example attribute classifications, provided by the user, to compute the mathematical function of those examples and construct a generalised model via a machine learning technique.

To clarify, figure 1 shows the concept from start to end. Firstly the user would have to provide example attribute classifications for the problem they wish to model. The examples are then formatted into a data set and fed through a learning algorithm. The algorithm learns from the example data, provided which results in a general model, which is able to generalise to new unseen examples in the problem domain.

![Figure 1 Example-Driven Modelling (EDM)](image-url)
Concerning the feasibility of example-driven modelling techniques
Thorne, Ball & Lawson

This approach eliminates the need for the user to produce formulae, the user only gives example data for the problem they wish to model. This therefore eliminates errors in constructing formulae since the user is no longer required to produce them.

The burden of calculation is placed on the computer, which using a machine learning algorithm, computes the function of the examples. As the literature suggests, this may be a more effective use of human and computer strengths (Michie, 1989)

However in the case of example giving for EDM this is only theory and some investigation into the feasibility of such an approach is required, i.e. how feasible is it for humans to think up examples for a given problem.

2.0 Investigating the feasibility of giving examples

To investigate if giving examples works in practice an experiment was designed to compare traditional spreadsheet modelling techniques and the novel approach of giving examples. The first group, the “treatment” group, were required to give example data to complete the tasks. The other group, the control group, were given the same tasks to complete using a spreadsheet application.

2.1 Experimentation

The experiment into feasibility was designed in accordance guidelines cited by Shadish et al. (2002) and Campbell and Stanley (1963). Also, published work using experimental methodologies in spreadsheet research were considered (Hicks and Panko 1995, Javrin and Morrison 1996, Panko and Halverson 1998, Javrin and Morrison 2000, Howe and Simkin 2006)

2.2 Experiment aim

The main aim of the experiment was to establish experimentally within an academic environment, using postgraduate students:

1. The relationship between error and task complexity using a) spreadsheet modelling techniques, b) example giving
2. The (hypothesised) superiority of Example giving over traditional spreadsheet modelling.
3. A satisfactory statistical measure of overconfidence.
4. The relationship between previous spreadsheet experience and accuracy for both traditional spreadsheet modelling and example giving

From these aims and objectives, we will be able to determine the feasibility of Example giving via three performance indicators

1. Whether the participants understand the instruction of giving examples, i.e. can users understand the instructions of giving examples and generate valid examples in the context of the experiment tasks.
2. The accuracy of the examples provided by the participants, i.e. what is the error rate for examples provided by participants
3. The comparative error rate when compared to traditional modelling, i.e. how does the error rate compare to that of traditional modelling and does this warrant further investigation.

2.3 Experimental design

The experimental model chosen to evaluate the aims of the experiment is the “Randomised two-group no posttest design”. Figure 2 shows the standard design of such experiments, this diagram is read from left to right and shows the

![Figure 2 Randomised two group no post test (Shadish et al. 2002)](image)

The diagram shows the two randomised (R) groups, the treatment group (X), the control group (which is left blank) and the two outcomes (O). In this case the control group receive ‘standard’ treatment, i.e. they develop spreadsheet formulae using the constructs and syntax in a spreadsheet application, such as Excel. The treatment group receive the novel approach, this allows relative comparison between the control and treatment groups.

2.4 Sampling

This sampling for this experiment is a cluster random sample as described by Shadish et al. (2002) and Saunders et al. (2007). Cluster sampling identifies a suitable cluster of participants and then randomly selects from within that group.

Considering similar development experiments in spreadsheets (Hicks and Panko 1995, Javrin and Morrison 1996, Panko and Halverson 2001), postgraduate Masters students were selected as an appropriate cluster.

Selection within the cluster was random, participants were not divided upon ability or any other basis.

Participants were invited to attend a session arranged for the experiment. Upon arriving participants were divided into two groups, the control and treatment groups. The appropriate materials for each group were distributed and the experiment began.

2.5 Research materials

The research materials for this experiment comprise two different packs handed to the participants.
Both packs contained a questionnaire gathering information such as age, sex, experience, number of years using spreadsheets, and a personal rating of their skill. This questionnaire was completed first, before the participants started the tasks. The point of this questionnaire is to gather demographic information and to determine the experience of spreadsheet use for a participant.

Once questionnaire 1 was completed, the participants started the tasks for the group they were assigned to (control or treatment). The scenarios contained in tasks for the participants, regardless of group, were identical. The manner in which the groups completed the tasks differed, the control group produced formulae in a spreadsheet using the syntax and functionality of the application (Microsoft Excel). The treatment group produced example attribute classifications for each task.

After completing the tasks as best they could, the final questionnaire, questionnaire 2, was completed. This questionnaire gathered information on the participant’s perception of their own performance, i.e. they were asked how difficult they felt each task was and then asked to indicate how confident they were that the provided answers were correct.

2.6 Experiment tasks

The five tasks for the experiment were identical, the method of completing them varied for each group. The control group submitted answers created using Microsoft Excel, the treatment group submitted attribute classifications written on paper.

The experiment tasks were designed to be progressively more difficult, requiring progressively more complex answers from both groups.

2.7 The tasks

The tasks given to the control and treatment group were identical, the method in which they answered varied.

For example, in the control group task 1 was to create a formula that could give a grade (Pass or Fail) based upon a single mark (Exam mark). The formula was required to distinguish between pass and fail, where fail was < 40 and pass was ≥ 40.

For the same task, task 1, the treatment group were required to give attribute classifications (examples) for every classification in the problem. The two classifications are pass and fail, the participants therefore had to submit an attribute classification of pass and fail.

The tasks were also designed to be progressively more complex. For example task one uses one value (exam mark), 2 classifications (pass and fail) and two parameters for those classes (<40 Fail, ≥ 40 Pass).

In contrast, task 5 uses 2 values (exam and coursework mark), 4 classifications (fail, pass, merit and distinction) 4 parameters (< 40 fail, ≥ 40 pass, ≥ 55 merit and ≥ 70
Concerning the feasibility of example-driven modelling techniques
Thorne, Ball & Lawson

distinction) and 1 conditional rule (Both exam and coursework values must fall in same class to award that class).

2.8 Marking the control group

Determining the mark of participants in the control group was based upon whether the answer provided was a valid formula in excel and whether the formula satisfied the specification in the task. If the formula fulfilled both criteria, it was deemed as correct, otherwise it is incorrect.

For incorrect formula, degrees of incorrectness were measured by counting the number of errors made in the submission. Errors can either be Mechanical, Logic or Omission, see Panko (1998) for a definition of these error types.

Once the number of errors was totalled, the submission was given a classification. The classifications were as follows: 0 errors = 5, 1 error = 4, 2-3 errors =3, 4 or more = 2, No attempt = 1.

These above classifications are used in the confidence calculation only, the other statistics are generated from dichotomous data.

2.9 Marking the treatment group

Determining the mark of the participants in the treatment group was based upon the whether the attribute classifications were valid and whether the attribute classifications provided satisfied the specification of the problem.

For incorrect attribute classifications, the number of errors per task was totalled and then given a classification. The classifications were as follows: 0 errors = 5, 1 error = 4, 2-3 errors =3, 4 or more = 2, No attempt = 1.

These above classifications are used in the confidence calculation only, the other statistics are generated from dichotomous data.

3.0 Summary statistics from experimentation

In this section performance indicators are compared between the treatment and control groups. This indicates the usefulness of example giving in comparison to spreadsheet modelling.

3.1 Accuracy

By comparing accuracy results gained from both the treatment and control groups, it is evident that the treatment group were more accurate than the control group. See Figure 4
Concerning the feasibility of example-driven modelling techniques
Thorne, Ball & Lawson

As can be seen, the treatment task accuracy ranges between 78 and 60 percent, the control group accuracy ranges between 66 and 30 percent. So comparatively, producing examples is more accurate than producing formulae.

3.2 Confidence

The confidence calculation indicates whether the group were perfectly calibrated, over or under confident. The formula for overconfidence is given in Figure 8 below.

$$Confidence = \frac{\text{Ratio perceived error rate}}{\text{Actual error rate}}$$

Figure 4 Confidence ratio calculation (Thorne et al. 2004)

Further details of this calculation are contained in Thorne et al. (2004)
The baseline on the graph shows the division between over and under confidence, a value of less than 1 indicates under confidence, over 1 indicates overconfidence. A value of 1 exactly indicates perfect calibration between expected outcome and performance.

As can be seen, both groups were under confident in their work. This is an unusual finding since the literature indicates that spreadsheet developers are usually overconfident (Panko, 2003).

Although the data in figure 9 shows that both groups were mostly under confident, there are some distinguishing features between them.

The treatment group’s data points are less erratic than the control group, indicating a more consistent approach to evaluating their performance. This erratic grouping is clearer if perceived difficulty (how difficult was this task?) and Perceived completeness (did you complete the task successfully?) are mapped against each other, see figure 6.

In figure 6, the treatment group’s data points are bunched together, suggesting the values are similar. The values are responses to difficulty and completeness questions, this suggests that the treatment group found the task’s difficulty and perceived completeness didn’t change as the tasks progressed. In figure 6, the data points read right to left as tasks 1 to 5.

The control groups are more dispersed, indicating that the values change as the tasks progress, i.e. as the tasks progressed they were harder and perceived to be less complete.

**4.0 Testing for statistical significance in the results**

**4.1 Introduction**
The raw data for both experiments, when graphed, allows conclusions to be drawn based up some basic statistics such as the mean value. Whilst this serves a purpose, it does not tell us if the results are statistically significant.

In order to see if the results are statistically significant a number of significance tests have been applied to the accuracy data. For example, the Chi squared test is used to determine if the differences in accuracy are statistically significant in the control and treatment groups. One can then determine if the increased accuracy observed in the treatment group was due to the treatment or not.

4.2 Chi squared test on accuracy data

The Chi squared test determines if the differences in accuracy for the treatment and control groups are due to the treatment and not chance. Once calculated, chi squared indicates if the “null hypothesis” should be accepted or rejected. The null hypothesis is usually the opposite of what the researcher wants to find, i.e. the null hypothesis is “There is no difference between the groups”.

The raw data consists of 1’s and 0’s, the tasks were either correct (1) or incorrect (0). This characteristic of the data allows us to use the chi squared statistic in figure 7.

\[
\chi^2 = \sum \frac{(O - E)^2}{E}
\]

Figure 7 Chi squared statistic

In cases where the sample size is small, Fisher’s Exact test can be used to complement or replace the chi squared test (Fisher, 1922).

4.3 Fisher’s exact test on accuracy data

Fisher’s exact test determines the probability of the scenario being tested, or one more extreme, occurring. For clarity the test determines the probability of the same scenario or a more favourable one arising. Fisher’s is applied when sample sizes are small, how small is unclear. Some cite less than 30 participants overall, some cite that less than 10 in a cell and some cite less than 4 in cell.

4.4 Summary of chi squared and Fisher’s exact statistics

The combined results obtained from chi squared and Fisher’s exact are contained in table 1 below.
Concerning the feasibility of example-driven modelling techniques
Thorne, Ball & Lawson

<table>
<thead>
<tr>
<th>Task</th>
<th>Chi squared test</th>
<th>Fisher’s exact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.396</td>
<td>0.205</td>
</tr>
<tr>
<td></td>
<td>0.5 &lt; P &lt; 0.01</td>
<td>80%</td>
</tr>
<tr>
<td></td>
<td>Accept Null</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.673</td>
<td>0.301</td>
</tr>
<tr>
<td></td>
<td>0.5 &lt; P &lt; 0.01</td>
<td>70%</td>
</tr>
<tr>
<td></td>
<td>Accept Null</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.03</td>
<td>0.128</td>
</tr>
<tr>
<td></td>
<td>0.5 &lt; P &lt; 0.01</td>
<td>88%</td>
</tr>
<tr>
<td></td>
<td>Accept Null</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2.03</td>
<td>0.128</td>
</tr>
<tr>
<td></td>
<td>0.5 &lt; P &lt; 0.01</td>
<td>88%</td>
</tr>
<tr>
<td></td>
<td>Accept Null</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4.22</td>
<td>0.038</td>
</tr>
<tr>
<td></td>
<td>0.05 &lt; P &lt; 0.02</td>
<td>96%</td>
</tr>
<tr>
<td></td>
<td>Reject null at 95% level.</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 Combined Chi squared and Fisher's exact statistics

The data in table 1 and the data graphed in figure 8, show that for both Chi squared and Fisher’s exact, tasks 1 to 4 are not statistically significant, assuming that 95% is the minimum level of significance.

However, both show on task 5 statistical significance which therefore rejects the null hypothesis on that test. We can conclude that for task 5 the observed difference in accuracy was due to the treatment not chance.

![Chi squared and Fisher's exact significance levels](image)

Figure 8 Chi squared and Fisher's exact significance levels

Since the tasks were designed to be progressively more difficult, one could interpret the results to show that the treatment is only effective in sufficiently complex scenarios.
Concerning the feasibility of example-driven modelling techniques
Thorne, Ball & Lawson

Using Cochran’s Q test determines if the difficulty between tasks was statistically significant.

4.5 **Cochran’s Q test on difficulty**

Cochran’s Q test allows us to test if the difficulty between all five tasks in a particular group was significantly different. The test therefore has to be performed on both the control and treatment group. The formula for Cochran’s Q test is given in figure 13.

\[
Q = \frac{\frac{1}{c-1} \sum_{j=1}^{c} \left( \frac{c_j - \bar{c}_j}{\bar{R}_j \left( c - R_j \right)} \right)^2}{\sum c_j \left( 1 - \frac{1}{c} \right)},
\]

Figure 9 Cochran's Q

4.5.1 **Cochran’s Q for the Control group**

The calculation for Cochran’s Q statistic in the control group is as follows:

\[
5 \times 4 \times (16 + 4 + 1 + 1 + 16) = 760 / (270 - 194) = 10.00
\]

DOF = 4

0.05 < P < 0.02

This shows that there is a significant difference in difficulty between tasks for the control group, we reject the null hypothesis at the 95% level.

4.5.2 **Cochran’s Q for the Treatment group**

The calculation for Cochran’s Q statistic for the treatment group is as follows:

\[
5 \times 4 \times (10.24 + 0.04 + 0.64 + 0.64 + 3.24) = 296 / (390 - 364) = 11.386
\]

DOF = 4

Look up on Chi Squared table

0.05 < P < 0.02

This shows that there is a significant difference in difficulty between tasks for the treatment group, we reject the null hypothesis at the 95% level.

4.5.3 **Conclusions on Cochran’s Q test**
The calculations of Cochran’s Q test show that at the 95% confidence level, the null hypothesis, all the tasks are the same difficulty, for both the control and the treatment group is rejected. Therefore we can conclude that there is significant difference in difficulty between tasks.

This supports the theory that as the difficulty increases, the treatment effect becomes significant.

However, tasks 3 and 4 both show the same result for chi squared and Fisher’s exact, see table 14. This might suggest that these two tasks were of similar difficulty based on the results.

In order to establish if this is the case, we must compare the two sets of data for the control and treatment group to see if there is statistical significance between them. One method to compare two data sets for difference in difficulty is McNemar’s test on difficulty (McNemar, 1947).

### 4.6 McNemar’s test on difficulty

The McNemar’s statistic allows us to test for significant difference in difficulty between the two groups, in this case the results for task 3 and 4. The test is $X^2$ using 1 DOF, see figure 10 for the equation.

$$X^2 = \frac{(b - c)^2}{b + c}. \quad (1)$$

**Figure 10 McNemar's test on difficulty**

**McNemar’s Calculations:**

- **Control Group**: $M = \frac{(3-3)^2}{(3+3)} = 0/6 = 0$

  We therefore accept the null hypothesis, there is no difference between the two groups, i.e. there is no significant difference in difficulty between tasks 3 and 4 for the control group.

- **Treatment group**: $M = \frac{(2-2)^2}{(2+2)} = 0/4 = 0$

  We therefore accept the null hypothesis, there is no difference between the two groups, i.e. there is no significant difference in difficulty between tasks 3 and 4 for the treatment group.

### 4.9 Conclusions on significance testing

The chi squared and Fisher’s tests indicate that in both the control and treatment groups, for tasks 1 to 4, there is no statistically significant difference in accuracy.

However, both chi squared and Fisher’s indicate that for task 5, in both control and treatment groups, the observed increase in accuracy is statistically significant. i.e. the
Concerning the feasibility of example-driven modelling techniques
Thorne, Ball & Lawson

difference in accuracy is due to the treatment and not chance, ergo giving examples in task 5 is more accurate than producing the equivalent formula. Cochran’s Q test indicates that between all five tasks, there is a significant difference in difficulty. McNemar’s test on the observed accuracy in tasks 3 and 4, which have the same values, demonstrates that there is no significant difference in difficulty between the tasks.

One possible explanation is that during the design of the materials, i.e. the tasks were not sufficiently different to yield a significant change in difficulty, hence the same accuracy values.

To conclude, there is a relationship between difficulty and statistically significant accuracy for the treatment. The results suggest that if the task or problem is sufficiently difficult, there is a statistically significant accuracy advantage in using the treatment over the control.

5.0 Conclusions

The conclusions of the experimental comparison between the Treatment group, i.e. giving examples and control group, i.e. producing formulae

5.1 Experimental Conclusions

1. The treatment group (giving examples) were considerably more accurate than the control group (producing formulae), see figure 4. Accuracy in task 5 only was task to be statistically significant, see table 2 and figure 14.
2. Both the treatment group (giving examples) and the control group (producing formulae) were consistently under confident, see figure 10.
3. Both groups found the tasks progressively more difficult as Cochran’s Q test indicated, except tasks 3 and 4 which showed no significance of this type, see section 3.5.7.

5.2 Limitations

Limitations to this experimental study include both general criticisms of experimental work and specific conditions that relate to the experiment. Also so criticism could be made of the statistical significance tests due to the way that they are marked.

5.2.1 Criticisms of the experiment

Firstly, the sample of participants is from an academic environment, experimentation with participants from a non academic environment would provide a broader view of the usefulness of this method.

Although there was no time limit imposed on the participants to complete the tasks, participants were not permitted to take the materials away from the venue. Some
might argue that this imposes a time pressure on the participants and that in reality they are more likely to complete the tasks over a longer time period.

However, to keep control of the experimental conditions one must insist that participants stay in the arranged venue until they have completed. Allowing them to remove and complete materials at another venue may allow collusion and thus the integrity of the experiment would be compromised.

It could be argued that the sampling approach taken in this experiment is not truly random. A clustered random approach was taken, i.e. a cluster of individuals were targeted and then randomly assigned to either the treatment or control group.

5.2.3 Criticisms of the significance testing

The significance tests show that only task 5 is statistically significant. The Cochran’s Q statistic shows that the difficulty difference between the tasks is statistically significant.

The tasks were designed to be progressively more difficult. The conclusion is therefore that the treatment effect is only statistically significant in sufficiently difficult tasks.

The statistics generated from the raw data are sensitive to the marking applied to the answers provided to each question. The answers were dichotomous, i.e. attempts were either correct or incorrect. In both the control and treatment group this mark was based upon whether the solution provided was a valid solution that covered the specification of the task.

If the method used to mark the answers provided for each task differed, one would expect to see a change in the statistics. If the statistics were calculated data that had been processed according to an invented marking criteria, the sensitivity of the statistics would be greater.

However, since all of the statistics were strictly marked in a dichotomous fashion, this sensitivity is not a limiting factor in this research.

5.3 Conclusion on the novel approach

The results of the experiment demonstrate that giving examples is more accurate, easier and less prone to overconfidence than creating formula. It is therefore feasible to use “giving examples” as the basis for a modelling method.
Concerning the feasibility of example-driven modelling techniques
Thorne, Ball & Lawson

References


McNemar Q., (1947), ‘Note on the sampling error of the difference between correlated proportions or percentages’. Psychometrika, 12, 153-157.


It Ain’t What You View, But The Way That You View It: documenting spreadsheets with Excelsior, semantic wikis, and literate programming

Jocelyn Paine
www.j-paine.org/ and www.spreadsheet-factory.com/
popx@j-paine.org

ABSTRACT

I describe preliminary experiments in documenting Excelsior versions of spreadsheets using semantic wikis and literate programming. The objective is to create well-structured and comprehensive documentation, easy to use by those unfamiliar with the spreadsheets documented. I discuss why so much documentation is hard to use, and briefly explain semantic wikis and literate programming; although parts of the paper are Excelsior-specific, these sections may be of more general interest.

1. INTRODUCTION

It is terribly important that computer programs should be easy for people to read and understand. Imagine that one of your clients needs an urgent fix to a program, but its author has gone on holiday or left the company; and although you have other staff with the general skills needed to fix it, none has ever worked on this particular program, or even on the phenomenon it models. How easily can they find its documentation, and how well will it tell them everything they need to know?

That, applied to spreadsheets, is the topic of this paper. I have been converting a 400,000-cell financial spreadsheet into code for my spreadsheet generator Excelsior [Paine, Tek and Williamson, 2006], in order that bespoke versions of the spreadsheet can quickly and reliably be produced for specific clients. Such a big spreadsheet needed extensive documentation. This led me to experiment with semantic wikis and with literate programming; the objective being to make the code the last thing that the modeller writes, ensuring that an easy-to-understand spec of each module has been written first. The experiments are preliminary, mainly concerned with implementation rather than evaluating readers’ reactions, but may still interest others.

This paper is organised as follows. Section 2 is a general introduction to Excelsior. Section 3 diagnoses the diseases suffered by today’s documentation, with reference to Parnas and Clements’ paper A rational design process: How and why to fake it [Parnas and Clements, 1986]. Section 4 explains semantic wikis in general; Section 5 applies them to documenting Excelsior programs. Section 6 moves from wikis to the more general topic of literate programming — writing programs that are easy for people to read. Sections 7 and 8 are short evaluations and conclusions.
Some parts of the paper are Excelsior-specific, and will not apply directly to Excel. However, you may find the wiki material useful if you need to store off-spreadsheet documentation, or if you are using spreadsheet generators other than Excelsior, such as the one blogged in [Voyce, 2007]. The references on what’s wrong with documentation, as well as on literate programming, I recommend to everyone.

2. BACKGROUND ON EXCELSIOR

At last year’s EuSpRIG conference, I described how my co-author Emre Tek and I had converted a 10,000-cell housing-finance spreadsheet to Excelsior code, using Excelsior’s features for listing spreadsheets as text, probing their structure, and “reverse-engineering” or “structure discovering” them. The paper is available as Rapid Spreadsheet Reshaping with Excelsior: multiple drastic changes to content and layout are easy when you represent enough structure [Paine, Tek and Williamson, 2006]. I shall summarise Excelsior briefly from that paper and show how I am using it in the current project.

Excelsior is a programming language for describing spreadsheets as sets of equations between groups of cells, or tables. The Excelsior compiler reads files containing the equations and generates an Excel spreadsheet. These files are submitted to it as plain text, accompanied by a layout specification indicating where the tables must go in the generated spreadsheet.

A single spreadsheet can be composed from more than one Excelsior file; indeed, Excelsior’s most significant claim is that spreadsheets can be developed from modules which can be written, tested, and documented separately. This makes spreadsheeting very flexible: we can develop a whole family of spreadsheets, producing different but related versions merely by substituting one module for another, or changing parameters such as table sizes.

As well as a compiler, Excelsior has tools for listing existing spreadsheets and converting them to Excelsior code. This usually involves guessing which cells can be grouped together into tables, naming the tables, and relisting the spreadsheet to use these names rather than the original cell addresses. This structure discovery uses, amongst other things, a run-detector to find blocks of related formulae differing only by constant increments in a cell address or constant.

New since last year is a tool for displaying sheet-dependency networks. This works by collecting, for each sheet, references in its formulae to all other sheets. It outputs this information as statements in the graph-description language DOT used by the freeware Graphviz graph visualization package. I mention this because Graphviz is useful for displaying well-laid out network diagrams, and may help others needing to diagram spreadsheets. To those trying it, I also recommend the interactive ZGRViewer. See [Graphviz], [DOT] and [ZGRViewer].

2.1 Excelsior Code Example

This example is from Section 2 of [Paine, Tek and Williamson, 2006]. Consider a spreadsheet for forecasting housing numbers. Let’s assume we have the one-dimensional tables Builds and Demolitions that give the number of houses built and demolished per
year, and Net that gives the net new houses per year. Then, if years 2000 and 2001 are valid subscripts for these tables, we could write:

\[
\begin{align*}
\text{Net}[2001] &= \text{Builds}[2001] - \text{Demolitions}[2001]
\end{align*}
\]

Tables can have more than one dimension. We could make the type of house a second dimension:

\[
\begin{align*}
\text{Net}[2000, 1] &= \text{Builds}[2000, 1] - \text{Demolitions}[2000, 1]
\end{align*}
\]

Excelsior has an "all" construction which generates a set of equations relating all elements of tables. Thus, Excelsior will expand the following into as many equations as there are elements of \text{Builds}, \text{Demolitions} and \text{Net}:

\[
\begin{align*}
\text{Net}[\text{all } y, \text{all } ht] &= \text{Builds}[ y, ht] - \text{Demolitions}[ y, ht]
\end{align*}
\]

### 2.2 Listing And Run-detecting Spreadsheets

Excelsior can list the formulae in an existing spreadsheet, for example:

\[
\begin{align*}
'\text{House Stocks}'!F1 &= "\text{Newly built houses}" \\
'\text{House Stocks}'!G1 &= "\text{Demolished houses}" \\
'\text{House Stocks}'!H4 &= '\text{House Stocks}'!F4 - '\text{House Stocks}'!G4
\end{align*}
\]

This can be useful, but if one formula is repeated many times, the listing will become too large and repetitious to be intelligible.

To avoid this, we can run-detect a sheet and then list it. When listing run-detected sheets, Excelsior uses a special notation to indicate repetition. For example,

\[
\begin{align*}
'\text{House Stocks}'!H[V0 \text{ in } 4:13] &= '\text{House Stocks}'!F[V0] - '\text{House Stocks}'!G[V0]
\end{align*}
\]

Here, the \([V0 \text{ in } 4:13]\) indicates repetition down a column. \(V0\) is a row variable; in the listing it varies over rows 4 to 13. The equation is therefore short for:

\[
\begin{align*}
'\text{House Stocks}'!H4 &= '\text{House Stocks}'!F4 - '\text{House Stocks}'!G4 \\
\cdots \\
'\text{House Stocks}'!H13 &= '\text{House Stocks}'!F13 - '\text{House Stocks}'!G13
\end{align*}
\]

As another example, suppose listing with run-detection displayed this equation:

\[
'\text{House Stocks}'![C:D]1 = "\text{Year}"
\]

This uses the same notation, but between columns rather than rows. The equation means that cells C1 and D1 of the sheet both contain "Year".
Finally, suppose that run-detection and listing displayed this:

```
'Home Sales'!V0 in C:D[V1 in 4:13] = 'House Sales'!V0+2[V1-1] - 'Flat Sales'!V0+3[V1+1]
```

This is short for:

```
'Home Sales'!C4 = 'House Sales'!E3 - 'Flat Sales'!F5
...
'Home Sales'!D13 = 'House Sales'!F12 - 'Flat Sales'!G14
```

### 2.3 Run Detection With Renaming

Consider again the equation:

```
'House Stocks'!H[V0 in 4:13] = 'House Stocks'!F[V0] - 'House Stocks'!G[V0]
```

It seems likely that cells H4:H13, F4:F13, and G4:G13 form three distinct tables. We could tell Excelsior to relist the spreadsheet while giving H4:H13 the name Net, F4:F13 the name Builds, and G4:G13 the name Demolitions. If further, we told it that row 4 corresponds to subscript 2000, and column F to subscript 1, listing would give the equation:

```
Net[ all V0, all V1 ] = Builds[ V0, V1 ] - Demolitions[ V0, V1 ]
```

These are clearly equivalent to the first equation, after renaming the cell groups as stated. This is the essence of how I have been converting the 400,000-cell spreadsheet mentioned in my introduction. Even if one doesn’t want to convert a spreadsheet to Excelsior, listing it with run-detection and renaming is a very good way to make it intelligible.

This, therefore, is how one can convert a spreadsheet to Excelsior code. But code is never enough. Code can only tell you how a program does behave, not how it should behave. This brings me to an important question.

### 3. WHY IS DOCUMENTATION SO BAD?

The rest of this paper describes approaches to documenting Excelsior programs so they are easy for people to read. I shall start by referring to a paper that makes valuable points about why documentation is bad and how to improve it, and is relevant to my work with wikis. I thoroughly recommend it to all programmers.

#### 3.1 We Must Fake Rationality

This paper is *A Rational Design Process: How and Why to Fake It* by David Parnas and Paul Clements [Parnas and Clements, 1986]; [Bredereke, 2002] is a summary. The authors argue...
that although we aspire to be rational software designers, we don’t act rationally. We may, for example, use a technique because we like experimenting with it, rather than because it’s best for our project.

Nevertheless, even though we can’t follow a rational design process completely, we should do so as closely as we can. Moreover, we should write the documentation we would have produced if we had followed the ideal process. We should “fake a rational design process”.

3.2 Why Is Documentation Hard To Use And Infrequently Read?

Section VI.A of the paper diagnoses the problems afflicting documentation. (It was published 20 years ago, but the problems remain). The authors state these to be (I quote):

(1) Poor organisation. Most documentation today can be characterised as “stream of consciousness,” and “stream of execution.” “Stream of consciousness” writing puts information at the point in the text that the author was writing when the thought occurred to him. “Stream of execution” writing describes the system in the order that things will happen when it runs. The problem with both of these documentation styles is that subsequent readers cannot find the information that they seek. It will therefore not be easy to determine that facts are missing, or to correct them when they are wrong. It will not be easy to find all the parts of the document that should be changed when the software is changed. The documentation will be expensive to maintain and, in most cases, will not be maintained.

(2) Boring prose. Lots of words are used to say what could be said by a single programming language statement, a formula or a diagram. Certain facts are repeated in many different sections. This increases the cost of the documentation and its maintenance. More importantly, it leads to inattentive reading and undiscovered errors.

(3) Confusing and inconsistent terminology. Any complex system requires the invention and definition of new terminology. Without it the documentation would be far too long. However, the writers of software documentation often fail to provide precise definitions for the terms that they use. As a result, there are many terms used for the same concept and many similar but distinct concepts described by the same term.

(4) Myopia. Documentation that is written when the project is nearing completion is written by people who have lived with the system for so long that they take the major decisions for granted. They document the small details that they think they will forget. Unfortunately, the result is a document useful to people who know the system well but impenetrable for newcomers.

On a personal note, I notice such problems in my own software documentation. Suppose I have written one module that defines a data structure and several others that use it (in any programming language, not just Excelsior). I am often tempted to explain in each module those facts about the structure that are most relevant. Thus, I may end up with several explanations of the structure. Each overlaps the others in coverage, without being complete in
itself. Later, I may rename a subsidiary structure or an operation in some modules but not others. Inconsistencies multiply, and a reader must examine the comments in many different modules, trying to reconcile them.

4. SEMANTIC WIKIS

The problems noted above include duplication of facts, difficulty in finding information, and lack of consistent and precise terminology, leading to ambiguity and confusion in naming things. These led me to experiment with semantic wikis.

A semantic wiki is an interlinked collection of Web pages which can be collaboratively edited through a Web browser, and which can automatically generate (amongst much other content) indices showing which pages link to which other pages. By convention, each page is written to be about one topic. Topics can be given a category, and the indexing tools can subdivide indices by category.

This seemed a promising framework for Excelsior documentation, because we could write one page for each module or important concept. Links between pages would indicate, for example, which modules use which other modules: thus the wiki’s organisation would reflect the spreadsheet’s structure. This structure would clarify where particular topics should be documented, making it easier to find information and eliminating duplication. Ambiguity and confusion in naming would be reduced because it is a wiki convention to use a page’s title as the name of its topic. For a link to work properly, the person who writes it must therefore use its correct name.

I shall now explain how wikis can be edited, displayed, and implemented. In the section following this one, I shall explain how I applied this to Excelsior.

4.1 Wikis

A wiki, as stated above, is a Web site that allows collaborative editing. The word has become associated with a particular style of notation for writing the pages, and a particular way of displaying them. There are many different implementations of wikis, and many wiki sites: the best known site is Wikipedia [Wikipedia].

4.2 Wiki Markup

Because wikis are Web sites, their pages ultimately get converted to HTML. One could therefore, write them in HTML. However, HTML’s tags and brackets are distracting, making it hard to focus on content. So although some wikis do let you insert HTML, they also have their own simple markup language. The “wiki engine”, i.e. the program that runs on a Web server and handles tasks such as editing, deleting, and creating pages, will render the markup into HTML before sending it out to a browser.

Here is an example, from the Wikipedia article about Yodel Bank:

'''Yodel Bank''' was an online [[anonymous banking]] system which ended operations during November of 2005. Yodel Bank
was not a registered company in any country, its operator's identity is unknown, and it existed entirely outside any countries' laws.

==Operations==

Yodel Banks operations functioned on top of various forms of [[electronic money]] including the [[Digital Monetary Trust]], [[e-gold]], [[Pecunix]], [[1MDC]], [[FreeTraders]], and the e-[[Liberty Dollar]].

The three quotes surrounding the first two words display them in bold. Double square brackets around a term mark it as a link to another article. The equals signs indicate that Operations is a section heading.

4.3 Semantic Wikis

A semantic wiki is a wiki crossed with a database. Pages can include semantic markup, also known as semantic annotations. Semantic markup can assert facts, about which the wiki’s users can ask questions using a semantic search interface. The wiki engine will generate a page on the fly to answer the search.

A page’s semantic markup can contain questions as well as facts. The wiki will again answer them, but this time, will render the answer as part of the page. This is how one includes machine-generated, but human-readable, content in a page. For example, suppose we have a semantic wiki with a page for each capital city. Suppose also that each of these pages has semantic markup stating which country its city is capital of. Then we could write a page about capitals in general, and place in it a question asking for all the capitals to be listed with their countries. Whenever anyone visited that page, the wiki would generate such a list; so it would always be up to date, no matter how many capital-city pages were added or deleted.

4.4 Links Are Facts; Indices Answer Questions About What Links Where

Semantic wikis typically treat links as facts. A link from page A to page B is treated as the fact “Page A links to page B”. If another page’s semantic markup includes a question that asks “which pages link to page B?”, the wiki will answer it by, in effect, generating an index of links. Typically, the person writing the markup can indicate how the index should be laid out, for example as a list of comma-separated words, or as a bulleted list.

4.5 Implementing A Semantic Wiki

Anyone can mount a wiki on their Web server. There are many different wiki programs available; I used the same one that Wikipedia does, MediaWiki [MediaWiki]. This was partly because it is free and well-tested, with many experienced users and good support. Also, I wanted to run it on my Web site. This is a commercial internet service provider (Mythic Beasts Ltd.), and the facilities provided by ISPs are often restricted, so the wiki had to work with what I had available. As it happens, Mythic Beasts ran MediaWiki for their own work, liked it, and recommended I install my own copy.
Installing MediaWiki involved downloading the distribution and then following the installation instructions. This was straightforward, but it helps to have experience with software installation and Web servers, so that you can recognise and correct typos and other blunders. Otherwise, it is worth asking an expert.

MediaWiki can be converted to a semantic wiki by installing Semantic MediaWiki [Semantic MediaWiki] on top of it. This also was straightforward.

5. ORGANISING THE WIKI: MODEL PAGES, MODULE PAGES, CONCEPT PAGES, AND INDICES

I have explained how one writes wiki content, how semantic wikis generate content from semantic markup, and that one can run a wiki on one’s own Web site. But how did I structure the knowledge within a semantic wiki so as to document an Excelsior-generated spreadsheet? That is the subject of this section.

5.1 Model Pages And Module Pages

The Excelsior version of any reasonably interesting spreadsheet will contain more than one module. (For the purposes of this paper, modules are the separate source files that together constitute the Excelsior code of a complete spreadsheet.) Our documentation must therefore distinguish between a spreadsheet and its component modules.

In a wiki, as Section 4 explains, it is conventional to have one page per topic. Semantic wikis allow each topic to be given a category. (Strictly speaking, some non-semantic wikis do too, but with far less flexibility in using them.) To apply this, I decided to have two categories of page: model pages and module pages. (To avoid confusing mathematical readers, I should say that I use the word “model” in the informal sense in which many people talk about “spreadsheet models”, not as in “model theory”. I use the word “category” in the sense of “classification”, not as in “category theory”.)

A module page documents a module. A model page documents an entire spreadsheet, and should at least state the modules from which the spreadsheet is composed. It can do so by linking to their pages: the wiki’s author can then write semantic markup to generate, for each module, a list of the models that use it.

As an example, the following question, placed within a page, will list all the pages that have that page as a module. For an explanation, see [Help:Inline queries]:

<ask>[[has module::{{PAGENAME}}]]</ask>

5.2 Concept Pages

Section 3.2 discussed the problems that arise when there is no clear place to put information. To avoid this in the wiki, I added another category of page, the concept page. The idea is that any information relevant to more than one module should live in its own concept page, to which other pages will link if they want to refer to the information. The page’s title, which must be unique within the wiki, can be an unambiguous name for the concept.
Although I’ve not evaluated its merits, I would expect a good writing style for concept pages to be the *inverted pyramid* [Inverted pyramid], favoured by journalists who must convey key facts quickly to a probably bored and inattentive reader. This style presents the gist immediately and succinctly, before descending into gradually increasing levels of detail. Hopefully, staff new to the material, who may have to read it at short notice, can thus assess its relevance before being overwhelmed by detail.

### 5.3 Semantic Wikis As An Interface To Excelsior

This section is more technical than the rest, but may interest wiki specialists. Its point is that the list of modules on a model page (Section 5.1) is an essential part of the Excelsior program defining the spreadsheet. If we’re going to include it in the wiki, shouldn’t we also include the rest of the program?

This leads to the idea of using the wiki as an interface to Excelsior; that we could write a list of modules and then submit it to a wiki add-on that finds the modules it links to, extracts their code, invokes Excelsior to compile it, and delivers up the resulting spreadsheet over the Web. Something similar might also be done with other spreadsheet generators than Excelsior.

Such add-ons are feasible, but I haven’t experimented with them. A decision needed here is how to represent the Excelsior code. The information about which modules constitute a spreadsheet is on the page as links, which the semantic wiki regards as semantic annotations (Section 4.4). Should the Excelsior code also be semantic annotations? This could make it possible for the wiki to generate table cross-reference listings, for example; or even cross-reference graphs, which could be generated by the nice graphing software written by “Tels” [Tels]. However, it would require extra markup, making the code harder to read and write.

It also hits one specific problem. Semantic MediaWiki allows one to write binary relations in the semantic annotations. A binary relation (this is standard mathematical terminology) relates two items, as in the statements “1 is less than 2”, “Model A uses module B”, or “Page A links to page B”. However, Semantic MediaWiki does not allow one to relate an arbitrary number of items, and this might be needed. There is a heavily modified version that does — used in BOWiki [BOWiki] — but it is not included with the standard distribution.

### 6. LITERATE PROGRAMMING

As an alternative to the wiki, I have been experimenting with “Literate Excelsior”, a version of Excelsior where the syntax of source files is modified to make it easier to document the code in them. This allows me closer control over the syntax than does the wiki, and also makes it possible to work away from an internet connection.

*Literate programming* is the discipline of writing programs so that they’re easy for people to read. This name, well-known in computing, was coined by Donald Knuth, who advertises his book [Knuth, 1992] on the topic by saying [Knuth, b]:

> Literate programming is a methodology that combines a programming language with a documentation language, thereby making programs more robust, more portable, more easily maintained, and arguably more fun to write than programs that are written only in a high-level language. The main idea is to treat a program as a piece...
of literature, addressed to human beings rather than to a computer. The program is also viewed as a hypertext document, rather like the World Wide Web. (Indeed, I used the word WEB for this purpose long before CERN grabbed it!)

Chris Lee gives a good summary at [Lee, 2000]. A nice collection of examples can be found at [Literate Programming Wiki] which calls each of its 407 articles “simultaneously a document and a piece of code that you can view, download, compile, and run”.

6.1 Input To Literate Programming Systems

In most literate programming systems, documentation goes in the same input file as the code it documents. Such systems therefore contain both a compiler and a tool that generates the display form (usually HTML) of the document. The literate programming system must therefore be able to distinguish code from documentation, and may impose conventions to do so. Thus the literate-programming form of a programming language is usually different from its standard form.

6.2 Literate Excelsior

Following the above, whereas “plain” Excelsior uses comment markers to delimit non-code text, I have implemented Literate Excelsior so it treats all text in input files as comment, unless indented by at least two spaces, when it becomes code. I found this an easy convention to follow when editing, and it reduces clutter both when editing and when reading the final documentation. A preprocessor extracts the code and feeds it to Excelsior’s compiler.

A separate preprocessor converts Literate Excelsior files to Web pages for display as documentation. This styles code as Wikipedia does, indenting it and setting it off within a pale blue box, thus making it immediately clear which parts of the document are code, without being distracting. The preprocessor recognises some simple wiki-style markup, such as equals signs to indicate a heading. I implemented this for the reasons noted in Section 4.2: such markup is easier to write than is HTML.

6.3 Narrative Style And The Order Of Program Pieces

The order in which things are explained is important for ease of understanding. Thus, a literate programming system must let you arrange your text in the order you deem most intelligible. Since code should be near the text that explains it, the system must let you do the same with the source code.

Many programming languages don’t allow such freedom. Pascal, for example, has rigid rules about the order of declarations. This is not a problem with Literate Excelsior, because the equations making up a program can be written in any order.

However, text ordering may be one area where literate programming will cause problems. People use documentation for many purposes. An order that works when reading about a program for the first time may not help once you know the program and need to find something specific within it. (I find this a problem with language books: a chapter that
introduces some easy touristic conversation in Portuguese or Greek is usually not a good reference.)

6.4 Literate Excelsior And Function Introduction

A very common pattern in Excel is the following conditional:

$$\text{IF} \left( \text{NOT}(\text{ISNA}(\text{expr})), \text{expr}, \text{other_expr} \right)$$

This is testing whether \text{expr} yields a sensible result, and returning it if so. If not, \text{other_expr} is returned. Similar patterns are seen, for example, in testing whether an expression is zero and dividing by it if not.

When \text{expr} is complicated, such formulae are hard to read. To mitigate this, Excelsior allows the programmer to define macros: functions which it will macro-expand (replace by their definitions) before feeding the formulae to Excel. Thus the programmer need only write \text{expr} once. Macros can be defined anywhere in a source file. This gives extra freedom when writing literate programs, because one can code important parts of a formula as macros, thus making them a named item in their own right, which can have its own explanation. The macro, and its explanation, can go either before or after the formula that calls it: sometimes one seems appropriate, sometimes the other.

7. EVALUATION

The work described here is preliminary, and I am still its main user, so there is not yet much to evaluate. Installing Semantic MediaWiki is straightforward, even under a commercial internet service provider. It is easy to define different categories of page within it, and to automatically generate link indices. The precise naming and lack of duplication enforced by concept pages is useful. However, when writing them, it was sometimes difficult to decide, when describing a concept that could be categorised into subconcepts, whether to give subconcepts their own page. Similarly, when concept pages already written turned out to be closely related, it was difficult to decide when to merge them. I would be interested to hear of others’ experiences.

As with the wiki, I am still the main user of Literate Excelsior. As explained in Section 6.2, the change in notation from “plain” Excelsior is small: no explicit comment markers, all text being documentation unless indented. But this makes a surprisingly big change to the “feel” of the writing: much more like writing expository text than like coding. Perhaps this will fade; at the moment though, it feels like a good thing.

8. CONCLUSION

One thing seems clear. Programmers — including spreadsheeters — need to document. They must therefore be good expository writers. A good starting point is Michael Covington’s How to Write More Clearly, Think More Clearly, and Learn Complex Material More Easily [Covington, 2002]. For examples of clear writing, I recommend any volume of essays by Isaac Asimov.
Documenting spreadsheets with Excelsior, semantic wikis, and literate programming

8. REFERENCES

BOWiki. onto.eva.mpg.de/bowiki/index.php/Main_Page 10:14pm 10/6/07.


Literate Programs Wiki. en.literateprograms.org/LiteratePrograms:Welcome 8:44pm 10/6/07.


“Tels”. <graph>-Plugin for MediaWiki, bloodgate.com/perl/graph/ 10:18pm 10/6/07. See also the same author’s Graph::Easy - Manual, bloodgate.com/perl/graph/manual/index.html 10:19pm 10/6/07.


Wikipedia. wikipedia.org/ 10:10pm 10/6/07.

ZGRViewer. ZGRViewer, a GraphViz/DOT Viewer, zvtm.sourceforge.net/zgrviewer.html 9:52pm 10/6/07.
Software solutions for Spreadsheet Risk

Philip Howard – Research Director

www.Bloor-Research.com
www.IT-Analysis.com

Agenda

What I'm not going to talk about:
- Types of errors
- Security
- What happens when Excel crashes
- Auditing
- Spreadsheets as an enterprise resource
- Productivity issues
- Enterprise management issues
- Best practices for building spreadsheets
- Best practices for preventing errors

All of these are discussed in the first half of the Bloor Research report: "Enterprise Spreadsheet Management"
Agenda

What I am going to talk about:

Software approaches to these issues and the products that address them

Types of approach

1. Do away with spreadsheets
   – not under discussion
2. Auditor’s tools
3. Automation tools
4. Control & compliance tools
Auditor's tools

- Used for both internal and external auditing
- Typically consist of a suite of (not particularly related, in most instances) utilities

Auditor's tools – typical facilities

- Circular reference checking
- Formulae - error identification, expansion, derivations, use of names
- Spreadsheet mapping
- Logic checks
- Precedent and dependent analysis
- Spell checking
- Spreadsheet comparisons
- Number derivation, number & range finders
Auditor's tools – additional facilities

- Sensitivity analysis
- Breakeven analysis
- Model reporting
- Spreadsheet analysis - statistics
- Usage reporting
- Development aids

Auditor's tools – products & vendors

Operis OAK
Sheetware XLSpell & others
Spreadsheet Advantage
Spreadsheet Detective
Spreadsheet Innovations (Professional)

& others
Auditor's tools - differentiators

1. Core functionality
2. Additional functionality
3. Visualisation

Auditor's tools - visualisation

Spreadsheet Advantage – mapping
Auditor’s tools - visualisation

Spreadsheet Detective – mapping

Sheetware - XLMapper
Automation tools

Apply formal development processes to the development of “spreadsheet applications”

Template-based with dynamic population of spreadsheets at runtime

No capability with respect to existing spreadsheets

Greatly reduce auditing, control and compliance requirements for new applications

Automation tools - facilities

- Cell-level version control
- Cell-level access control
- Template-based
- Wizard-based development
- Formatting separate from logic
- Federated data access
- Dynamic serving
- Audit trail
- Workflow
- Printed authentication
- Scheduling
- Alerts
Automation tools – products & vendors

Actuate
Qtier

Also Risk Integrated – not full solution

Control and Compliance tools

Sub-categories:

1. Compliance only
2. Includes Auditor’s tools
3. Focus on Security
4. Collaborative capabilities
5. SaaS
Control & Compliance - other major functions

- Discovery
- Risk Assessment

Control & Compliance - visualisation

Mobius workflow (approvals process)
Software solutions for Spreadsheet Risk
Howard

Control & Compliance - visualisation

Prodiance workflow (change control)

Control & Compliance - vendors

Cimcon, Compassoft, Prodiance
ClusterSeven – compliance only
Liquidity – SME/departmental
Mobius – DO NOT BUY
ROIsoft ExSafe – security but no risk or audit tools
SmartDB eXpresso – SaaS & collaboration, discovery planned, limited audit tools

As reported in IT- Director: On the same day that ASG acquired Mobius, ASG laid off almost the entire support staff. [http://www.it-director.com/business/change/content.php?cid=9591&mode=full&hilite=14299](http://www.it-director.com/business/change/content.php?cid=9591&mode=full&hilite=14299)
& Finally

Auditor's tools ~ £200-300
Automation tools
   ~ Qtier/Actuate: 5 figures
   ~ Risk Integrated: 3 figs
Control & Compliance
   ~ most: 6/7 figures
   ~ Liquidity: 4 figures
   ~ SmartDB: SaaS

Conclusion

There are a range of solutions to suit different pockets, which means:

There is no excuse for not, at the least, improving the condition, state and accuracy of your spreadsheets.

And for larger companies, for not providing much improved security, auditing and compliance.
Thank You

Enterprise Spreadsheet Management available at www.bloor-research.com

... optimise your IT investments
Voice-controlled Debugging of Spreadsheets

Derek Flood, Kevin Mc Daid,
Dundalk Institute of Technology, Dublin Road, Dundalk
derek.flood@dkit.ie, kevin.mcdaid@dkit.ie

ABSTRACT

Developments in Mobile Computing are putting pressure on the software industry to research new modes of interaction that do not rely on the traditional keyboard and mouse combination. Computer users suffering from Repetitive Strain Injury also seek an alternative to keyboard and mouse devices to reduce suffering in wrist and finger joints. Voice-control is an alternative approach to spreadsheet development and debugging that has been researched and used successfully in other domains. While voice-control technology for spreadsheets is available its effectiveness has not been investigated. This study is the first to compare the performance of a set of expert spreadsheet developers that debugged a spreadsheet using voice-control technology and another set that debugged the same spreadsheet using keyboard and mouse. The study showed that voice, despite its advantages, proved to be slower and less accurate. However, it also revealed ways in which the technology might be improved to redress this imbalance.

1. INTRODUCTION

It is not convenient or efficient for all spreadsheet users at all times to control spreadsheet technology through the usual keyboard and mouse combination. With the move towards mobile and smaller battery powered computer devices and the growing number of sufferers of Repetitive Strain Injury (RSI), alternatives to the keyboard and mouse interface systems are receiving much attention. As most mobile computers feature a built-in microphone, speech interface to applications is an obvious technological option. Speech interfaces have the additional benefit in that they allow users communicate in a natural way. This paper explores the use of a speech interface for spreadsheet debugging.

Spreadsheets are becoming more complex in nature and as such the risk of errors remaining in spreadsheets is increasing. Often spreadsheets are used as the basis for important business decisions and therefore it is important that they are error free. Unfortunately this is not the case, with many studies recording an unacceptably high level of errors in spreadsheets[Brown 1987, Panko 1998]. One such example caused stock values at Shurgard Inc to fall sharply after two employees were overpaid by $700,000 [Fisher, 2006].

In order to ensure that errors like this do not occur, spreadsheets should be thoroughly checked and debugged. This process involves reviewing the spreadsheet to ensure that no errors exist and if any errors do exist then they are repaired. The removal of these errors is known in software development as debugging. Section 2 of this paper highlights a number of issues with spreadsheet development.

There are existing speech-based technologies for the development and debugging of spreadsheets, the most popular being Dragon NaturallySpeaking developed by Nuance [nuance, 2007]. These systems allow users to control the spreadsheet application through voice commands, providing users with shortcut phrases for navigation, data and formula entry, and formatting. Section 3 reviews the state of the art in voice-control technology.
This work evaluates the effectiveness of current state of the art technology for voice control of spreadsheet debugging. We have conducted a trial in which a set of users debug a spreadsheet through voice and compare the results with users who performed the same task using traditional keyboard/mouse control. We found that participants that used voice technology found on average 15% fewer errors and took almost twice as long to complete the task. There were also significant differences in the time to navigate through the spreadsheet and the time to implement changes. Section 4 details the trial and the environment. An analysis of both the quantitative data, gathered through an experiment, and the qualitative data gathered through a structured questionnaire-based interview is presented in Section 5.

Section 6 concludes the paper with a summary of proposed future work in the area.

2. SPREADSHEETS

Spreadsheets are an important decision making tool. They help companies make multi-million euro decisions on a daily basis. The financial district within the city of London is responsible for 3% of the United Kingdom’s Gross Domestic Product (GDP). Within this one square mile, spreadsheets have been described as the primary front line tool of analysis [Croll, G., 2005]. Despite the importance of the decisions being made based on these spreadsheets, there is very little done in practice to ensure their quality.

A number of researchers and consultancy firms have looked at the quality of spreadsheets. One consulting firm in England, Coopers and Lybrand, found that 90% of spreadsheets with over 150 rows contained errors [Panko, 1998]. These errors range in severity from simple spelling errors to complicated formula errors.

A range of techniques have been proposed to improve the development of spreadsheets. A number of the initiatives focus on the importance of development methodologies similar to existing rigid document driven lifecycle processes in place for software development. More Flexible approaches have been adapted from agile software development methodologies. [O’Beirne, 2002; Rust, A., Bishop, B., Mc Daid, K., 2006].

Interesting research is ongoing in the use of Assertions [Burnett, M., Rothermel, G., Cook, C., 2006]. This method allows developers to place restrictions on the value of a given cell. The cell may be just a number or may include a formula. When a cell contains a formula the application calculates a value that the cell should be based on the assertions placed on the input cells. If there is a conflict between the value within the cell and the assertion placed on the cell then the user is notified as to a potential error.

3. VOICE COMMAND TECHNOLOGY

Voice command technology refers to the use of speech and voice recognition to enter information into a computer system. The system listens for certain key words or commands and performs a predefined action upon hearing these commands. This technology is not new. However, with the increase in computer power in recent years the technology has improved to the point that accuracy of the tools is advertised to be of the order of 99% [Nuance, 2006].

Improvements in technology have also allowed computers to shrink in size so that a computing device can fit easily into your pocket. For these portable computers, traditional input methods are no longer practical and alternatives must be explored. Voice command technology is the obvious candidate for control of such devices.
Voice command technology could also help address the issue of Repetitive Strain Injury (RSI) in the workplace. The pain of this injury can prohibit sufferers from using a keyboard. In the UK 1 in 50 of all workers have reported an RSI condition [RSIA, 2007]. As a result every day six people in the UK leave their jobs [RSIA, 2007]. While voice technology is not the complete solution, it could remove the need for sufferers to use a keyboard while performing most computer related tasks.

Voice command technology has been used for creating and modifying text documents [Begel, 2005], and has had widespread uptake. While Tony Stockman [2005] has looked at the sonification of spreadsheets, to the best of the authors’ knowledge there has been no research published in the area of voice-controlled spreadsheet development.

With the increased interest in the area of voice technology, there has been a flurry of new applications across multiple domains. Andrew Begel [2005] has looked at using voice technology to allow developers to develop java programmes through speech with the development of Spoken Java. Spoken Java is a naturally verbalizable alternative to java. It allows developers to speak code in a natural way, and have the java source code generated based on their statements.

Voice control of in-car navigation systems has also been investigated. While driving in a foreign country drivers frequently have to consult maps in order to find where they are going. Coletti et al [2003] have developed an in-car assistance where drivers can ask questions in a natural language and the system will provide the driver with the results through voice. This way the driver does not have to take their eyes off the road. The system allows the driver not only to get directions but can also allow them to make hotel and restaurant reservations.

Voice commands have also been used by Wang [2006] to develop a 3-D animation application. As with traditional movie sets, the director tells the actor what they need them to do and the actors perform as instructed. Wang has developed an application that allows a 3-D animator to act like a director and tell the objects on screen what to do and the objects will perform as instructed. The system is based on voice commands where the director issues the commands and the actors and objects behave based on predefined actions for the commands.

There are many voice recognition tools available; the best of these tools is Dragon NaturallySpeaking. Dragon NaturallySpeaking began in the early 1980’s as DragonDictate [wikipedia, 2007]. Since then it has passed through 4 companies and is currently being sold by ScanSoft Inc. There are currently 5 versions of this popular dictation software on the market. The most basic of these, Standard edition, features support for Microsoft© Word and web browsing through Microsoft© Internet Explorer. The next edition, the preferred edition, includes support for Microsoft© Excel. The professional Edition builds on this again, by offering support for all Microsoft© Office applications. There are also two special editions, Legal edition and Medical edition, which provide special recognition for specialised legal and medical terms [nuance, 2007].

Microsoft Speech API is a competing voice recognition technology. It has two modes of operation, “command and control” and “dictation”. The “command and control” mode allows users to use predefined commands. While in this mode the software will only recognise words that have been defined as commands. While in “dictation” mode the software will interpret words and phrases as text and will ignore any commands that may be associated with these words. Users are required to manually switch between the two
modes, which is often a source of frustration for new users. Dragon on the other hand integrates both modes.

The voice recognition software evaluated to date tends to be slow to process input. In order to recognise input the voice recognition software takes time to interpret the sound waves and understand what the user said. While the time lag is only of the order of seconds, any application would involve the processing of a large number of commands which can on occasion result in a significant net delay.

4. TRIAL AND RESULTS

The aim of the trial is to investigate the efficiency of voice control technology for debugging spreadsheets. The evaluation will focus on looking at the following three tasks, numeric input to cells, formula input and modification, and navigation through the workbook.

Three participants with significant experience in the development of spreadsheets partook in the trial. None of the participants had any prior experience using Dragon NaturallySpeaking or any voice recognition software. For the duration of the trial they were placed in a quiet room to minimise the amount of background noise and thus maximise the effectiveness of the voice recognition system. In the room was one additional observer, the first author, to answer any questions participants may have had during the trial.

The trial was run on a Dell Notebook D610 with an Intel Pentium M 1.60 GHz, 1.0 GB of RAM running Windows XP Service Pack 2. For the voice recognition software Dragon NaturallySpeaking 9 Preferred Edition was used. A Creative HS – 600 headset provided the audio input.

The session included three elements and took approximately 2 hours. The first element asked participants to read a set of predefined text so that Dragon NaturallySpeaking could be calibrated for their voice. This took approximately ten minutes with no significant difficulties experienced by any of the participants.

The second element of the trial allowed participants to become familiar with the commands available to them within the spreadsheet environment. Participants were given a basic task involving input of text, data and basic formulas. This task was not assessed in any way and was designed purely to give the participants the opportunity to familiarise themselves with the technology and the range of commands. Participants were encouraged to experiment with the various inbuilt commands. Forty minutes was assigned to this element of the trial.

The final element of the trial included the experiment central to the study. Participants were given a pre-designed spreadsheet containing forty-two errors, which can be divided into 4 categories, Clerical, Formula Input, Rule Violation and Data Input, and were asked to find and repair as many errors as possible. Participants were only allowed to use the voice recognition technology and were not allowed to use the keyboard or mouse. Participants were handed a list of rules and an explanation of what was required of the various parts of the spreadsheet. They were not however told the types of errors to look for. There was no time limit set for this activity with participants finishing when they felt they had completed the task. Note that the workbook contained three sheets named, “Payroll”, “Office Expenses” and “Projections”.

158
This experiment is identical in structure to one developed by Bishop and McDaid [2007] in which 13 professionals completed the task. The analysis section will compare the performance of the three voice interface subjects with the professional subjects that used keyboard and mouse technology in the Bishop and McDaid Study.

It is important to note that technology was used to record the time of all selection and editing actions in the spreadsheet. As this was also the case for the study of keyboard users a more detailed comparison of the results could be made. Furthermore, audio recordings of the participants’ commands throughout the experiment were kept using Easy MP3 sound recorder.

Finally, upon completion of the experiment participants were interviewed through a structured questionnaire. This was used to establish their views on the technology and the study.

5. ANALYSIS OF RESULTS

<table>
<thead>
<tr>
<th></th>
<th>Average Performance</th>
<th>Average Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voice Group</td>
<td>57 %</td>
<td>64 Minutes</td>
</tr>
<tr>
<td>Keyboard Group</td>
<td>72 %</td>
<td>28 Minutes</td>
</tr>
</tbody>
</table>

Table 1 Overall Performance

5.1 Overall

It was found that, compared with 13 professionals who used keyboard/mouse technology, participants of this study took significantly longer. The average time for the voice participants was almost 64 minutes compared to 28 minutes for traditional users. The fastest voice participant was just over 55 minutes, almost twice as long as traditional input. These time differences could have serious consequences within a business environment.

Despite the longer time, it was found that voice participants under performed when compared with their keyboard counterparts, with an average of 57% of the errors corrected whereas keyboard users found and corrected 72% of the errors. The results for the three subjects where 45%, 50% and 76%.

Although the participants in the voice study differed from those in the keyboard/mouse study all subjects had extensive experience of spreadsheet development and would have been expected to perform to a similar standard in the debugging experiment.

<table>
<thead>
<tr>
<th></th>
<th>Numeric Input</th>
<th>Formula Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voice Group</td>
<td>18.1</td>
<td>73.5</td>
</tr>
<tr>
<td>Keyboard Group</td>
<td>5.5</td>
<td>7.3</td>
</tr>
</tbody>
</table>

Table 2: Average Input Times in Seconds
5.2 Editing Formulas

The analysis examined in detail the time to navigate and edit the cells of the workbook. It was found that most of the time was spent editing formulas. With a keyboard this process is fairly straightforward, however when using voice technology this proved to be quite difficult. It was found that on average voice users took 73 seconds to edit a formula whereas the keyboard group were able to perform the same tasks in an average of 7 seconds. This was based on a set of 9 errors that where edited by all voice subjects and the average value for the cell was compared with the average for those professionals that edited the cell. The average time to edit the nine cells is given in Table 3. This striking difference comes from the method by which voice users had to edit the formulas. In order to move a single character to the left within the formula, the voice users need to say “Move Left” and wait for the recognition engine to recognise what they have said and then to perform the move. In contrast keyboard users just need to hit the left arrow key.

In cell I14 of the payroll worksheet the voice users took on average 290 seconds. The main reason for this large time is that one participant took almost 7 minutes to edit this cell. The issue was with the word “sum” and the letter “m” which was repeatedly interpreted by the recognition software as “n”.

<table>
<thead>
<tr>
<th>Cell</th>
<th>Voice Group</th>
<th>Keyboard Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payroll F10</td>
<td>30.4</td>
<td>4.6</td>
</tr>
<tr>
<td>Payroll G16</td>
<td>19.2</td>
<td>5.5</td>
</tr>
<tr>
<td>Payroll I10</td>
<td>32.2</td>
<td>6.1</td>
</tr>
<tr>
<td>Payroll I14</td>
<td>290.3</td>
<td>6.2</td>
</tr>
<tr>
<td>Office Expenses F8</td>
<td>6.0</td>
<td>4.4</td>
</tr>
<tr>
<td>Office Expenses F5</td>
<td>23.7</td>
<td>9.7</td>
</tr>
<tr>
<td>Office Expenses F20</td>
<td>154.7</td>
<td>13.7</td>
</tr>
<tr>
<td>Projections B17</td>
<td>87.4</td>
<td>7.6</td>
</tr>
<tr>
<td>Projections G22</td>
<td>18.1</td>
<td>7.6</td>
</tr>
</tbody>
</table>

Table 3: Average time to edit formulas in seconds

5.3 Entering Data

Participants using voice-control technology found it easier to enter numeric data as opposed to formulas. The time it took to enter numbers on average for voice control subjects was found to be 18 seconds in comparison with keyboard users who took on average almost 6 seconds. This was based on 6 cells and the complete set of values is shown in Table 4.
Another important aspect to spreadsheet debugging is navigation of the spreadsheet. It was found that using voice technology proved to be slower. In particular we examined the time taken to traverse five cells containing a copied formula without errors. An initial investigation showed that voice technology subjects take significantly longer to traverse spreadsheets. We are also exploring the time to navigate between spreadsheets. Preliminary studies indicate that it is extremely time consuming to switch between sheets when attempting to investigate formulas linked to cells in other sheets.

### 5.5 Workarounds

When assessing the efficiency of voice controlled technology it should be stated that in certain situations users were unable to complete a task in the usual way and used workarounds to complete the task. An example of this relates to one of the errors within the spreadsheet where the word “Mar” was misspelled as “Mac”. One particular user was having trouble getting the recognition software to recognise the letter “r”, as it repeatedly produced the number “4”. In the end the user abandoned trying to spell the word and instead chose to copy the word from elsewhere and pasted it into the cell.

These results show that, at present, there are real issues with the use of voice technology to debug spreadsheets.

After they had completed the task, users were asked a series of questions to determine their experience of using the voice recognition software in the trial. Two out of the three users felt that using voice commands distracted them from the task at hand. However one user found that using voice made him concentrate more so that they would not have to redo part of the spreadsheet again. All of the participants agreed that learning the voice commands was relatively easy and intuitive.

### 6. CONCLUSIONS

Voice recognition technology can be used to debug spreadsheets, however it was found that it is associated with poorer performance and takes more time than traditional keyboard and mouse entry. By providing users with a spreadsheet containing errors and asking them to find and repair them, one group using voice and one group using traditional input methods, it was found that voice recognition users took nearly twice as long and found 15% less errors.
From what we have learned, we do not believe that voice technology in its current form is a very suitable means for spreadsheet debugging. However, we feel that with certain improvements it may in the future be possible to use the technology.

In particular we believe that the system could be improved through the addition of an intelligent navigation system that suggests the most useful navigational actions for the user. A recommender system [McCarey, 2004] may be used to identify the most useful actions. These actions will not be automatically executed; instead they will be displayed so that the user can if they wish take advantage of them through a simple set of voice commands. Another possible enhancement could be the development of a more intuitive formula input system. This system could use Begel’s Spoken Java [Begel, 2005] as a starting point.

Another area of interest is kinaesthetic memory which relates to the memory in relation to space. In [Tan, 2002] the authors compare the use of touch screen technology with mouse technology. They found that users using the touch screen could remember more accurately than those using a mouse. It would be interesting to see how this might impact on voice controlled development of spreadsheets.

7. REFERENCES


ABSTRACT

Very little is known about the process by which end-user developers detect and correct spreadsheet errors. Any research pertaining to the development of spreadsheet testing methodologies or auditing tools would benefit from information on how end-users perform the debugging process in practice. Thirteen industry-based professionals and thirty-four accounting & finance students took part in a current ongoing experiment designed to record and analyse end-user behaviour in spreadsheet error detection and correction. Professionals significantly outperformed students in correcting certain error types. Time-based cell activity analysis showed that a strong correlation exists between the percentage of cells inspected and the number of errors corrected. The cell activity data was gathered through a purpose written VBA Excel plug-in that records the time and detail of all cell selection and cell change actions of individuals.

1. INTRODUCTION

The ubiquity of spreadsheet programs within all levels of management in the business world indicates that important decisions are likely to be made based on the results of these, mainly end-user developed, programs. The financial sector is particularly dependent on spreadsheets [Croll, 2005]. Unfortunately, the quality and reliability of spreadsheets is known to be poor following empirical and anecdotal evidence collected on the subject [Panko, 1998], [Rajalingham et al, 2000] and [Chadwick, 2004]. From the experience of one consulting firm, Coopers and Lybrand in England, 90% of spreadsheets with over 150 rows of data were found to contain one or more faults [Panko, 1998], and due to the nature of spreadsheets, when failures do occur, the results can be quite significant. For example, sudden budget cuts were necessary at the University of Toledo after an erroneous spreadsheet formula inflated projected annual revenue by $2.4 million [Fisher et al, 2006].

Many spreadsheet auditing tools have been developed and are widely available, but to develop auditing tools that compliment end-users natural auditing and debugging behaviour, research into this behaviour needs to be conducted. To date, we have found only one study that addresses end-user behaviour/processes in the inspection and debugging of spreadsheets, [Chen & Chan, 2000]. The study was somewhat limited as cognitive processes were captured using video taping and a thinking-aloud protocol from four participants without spreadsheet and accounting expertise. To this end, we undertook to investigate and unintrusively record the behaviour of industry-based professionals and students during the spreadsheet debugging process. Thirteen industry-based spreadsheet developers and 34 accounting and finance students took part in the experiment.

The layout of the paper is as follows. Section 2 introduces the topic of spreadsheet error detection and correction and compares the activity of spreadsheet inspection and
debugging with that of imperative programming language verification, validation and debugging. Section 3 details our research goals and experiment methodology. In Section 4 initial findings of the experiment are presented. A conclusion and proposed future research are detailed in Section 5.

2. SPREADSHEET ERROR DETECTION AND CORRECTION

Very little research has been conducted on the error detection process for spreadsheets. The emphasis of the small amount of spreadsheet research available has been on the prevention of spreadsheet errors through spreadsheet design and testing methodologies. The notable exceptions to this are [Chen & Chan, 2000] which is mentioned in the previous section, [Galletta et al, 1993], [Galletta et al, 1996], [Panko, 1999] and [Howe & Simpkin, 2006] in which studies on error-finding performance, the effect of spreadsheet presentation in error detection, applying code inspection to spreadsheet testing and the factors affecting the ability to detect spreadsheet errors were undertaken respectively. Importantly, none of these papers, unlike our work, deal with the cell-by-cell processes by which, or the order in which, these errors are found and corrected. In [Galletta et al, 1996], the author concludes that an increased understanding of the error-finding process could help avert some of the well publicised spreadsheet errors.

Authors have in the past looked to the traditional software development domain for methods and tools that could yield spreadsheet process improvement. Examples include the application of code inspection to spreadsheet testing [Panko, 1999], software visualisation applied to spreadsheets for fault localisation [Ruthruff et al, 2003] and using test driven development, an eXtreme Programming technique, for developing spreadsheets [Rust et al, 2006]. The application of traditional software verification and validation (V&V) and debugging research to the spreadsheet paradigm would seem like a natural course of action for spreadsheet error detection and correction research, but error detection and debugging in spreadsheets is a combination of static and dynamic V&V, and debugging associated with traditional programming languages. A comparison between spreadsheets and traditional programming languages and the ramifications of the differences with regard to testing techniques and terminology are discussed briefly in the next section.

2.1 Software Verification and Validation: Spreadsheets Vs Traditional Programming Languages

In traditional software development, verification and validation (V&V) are processes used to determine if the software is being built correctly and if it the correct software is being built, respectively [Jenkins et al, 1998]. V&V is concerned with establishing the existence of defects in a system, as distinct from debugging, which is concerned with locating and repairing these defects once their existence is established [Sommerville, 2004].

Software verification is composed of both static and dynamic verification. Static verification involves the inspection of code and other development artifacts. It is static in nature in that the system is not exercised in the examination of the code or documents. Dynamic verification involves program testing, concerned with exercising and observing program behaviour [Sommerville, 2004]. They are two distinct processes. In the spreadsheet paradigm, code inspection (individual phase) and the debugging process invariably become amalgamated. In the spreadsheet debugging process, where the sought error is found and corrected, the spreadsheet updates all cells. In this way the implication of the change can been seen immediately. This is similar to the dynamic verification
process of software testing. What can be taken from this is that static verification, dynamic verification, and debugging in traditional software development are three distinct processes, whereas spreadsheet debugging involves the integration of these into a single process. Further differences between spreadsheet and imperative programming paradigms, and the ramifications these differences may have for spreadsheet testing methodologies have been addressed in [Rothermel et al, 2001].

The terms spreadsheet auditing, debugging, inspection, testing, error finding etc. are sometimes used interchangeably in spreadsheet literature. For the purposes of this paper, spreadsheet error detection refers to the inspection by an individual of the spreadsheet code and the discovery of any errors, error correction refers to the successful correction of discovered errors, and spreadsheet debugging refers to the combination of these processes.

3. ANALYSING END-USER BEHAVIOUR: RESEARCH GOALS AND METHODOLOGY

3.1 Research Questions

Some questions we seek to answer are as follows:

- Will industry-based spreadsheet developers outperform students in detecting and correcting certain types of errors?
- Is there a correlation between the number of cells inspected and error detection and correction performance?
- Can common patterns of spreadsheet debugging behaviour be identified, and are there particular patterns that are more effective than others?

The research is in its early to mid stages, and many questions and hypothesis have yet to be answered and tested respectively. Initial analysis and some interesting findings from our first experiment are presented in Section 4. The following section details the experiment methodology.

3.2 Methodology

Experimental Spreadsheet Model

Regarding the detail of the experiment, a spreadsheet model has been developed consisting of three worksheets seeded with errors (see Appendix A). The names and functions of each of the three worksheets are as follows: Payroll, compute typical payroll expenses; Office Expenses, compute office expenses; Projections, perform a 5-year projection of future expenses. Each worksheet has different error characteristics. Payroll has data entry, rule violation and formula errors; Office Expenses has clerical, data entry and formula errors; Projections has mostly formula errors.

Participants were asked to debug the spreadsheet, and each error found was to be corrected directly in the spreadsheet itself. The spreadsheet model was adapted from a previous experiment carried out by Howe & Simpkin [2006], in which 228 students took part in an experiment designed to identify the factors which influence error-detection capability. Among other advantages, using a similar spreadsheet model to the one detailed in [Howe & Simpkin, 2006] allows us to compare results obtained from a large number of students with those of industry-based professionals.
Although other error classification systems exist [Teo & Tan, 1997], [Panko, 1998], the error classification system from [Howe & Simpkin, 2006] was utilised for this experiment, mainly to allow for detailed comparisons to be made between the error detection results of the 13 professionals and 34 students from our study and the 228 students from the experiment detailed in [Howe & Simpkin, 2006]. The error categories, and number of seeded errors of each category, are as follows.

- Clerical/Non material errors (4), such as spelling errors.
- Rule Violations (4) are cell entries that violate company policy, for example paying an employee overtime when that employee is not eligible for overtime.
- Data Entry errors (8) include negative values, numbers entered as text etc.
- Formula Errors (26), such as inaccurate range references, illogical formulas etc.

Spreadsheet Cell Activity Tracking Tool

Crucially, a tool has been developed (in VBA) to record the time and detail of all cell selection and cell change actions of individuals while debugging a spreadsheet. The data recorded is as follows: individual cells selected, cell ranges selected, worksheet selections, individual cells edited & the resulting cell value, cell ranges edited & resulting cell values. Timestamps are recorded for all of these activities (in milliseconds). More complex spreadsheet activities can also be identified from the resulting data log. These include copy and past, undo typing, redo typing and drag-and-fill.

Sample

Thirteen industry-based spreadsheet developers took part in the experiment, along with thirty-four second year accounting & finance students. The backgrounds of the professional participants are as follows: Accountants, 8; Financial Analysts, 2; Actuaries, 1; Software Developers, 2; with all the professional participants, including the two software development participants, having an industry based working knowledge of spreadsheet development and use. The need for spreadsheet experiments with industry professionals as opposed to the student population has been voiced by many researchers in this area, including the authors of the aforementioned paper, Howe & Simpkin [2006]. For comparison purposes the experiment was also carried out with 34 second year accounting and finance students.

Process

The subjects were given a copy of the experimental spreadsheet along with an instructions page. A short introduction on the instructions page explained the purpose of the task, namely to investigate how effectively spreadsheet users discover and correct errors. Subjects were asked to correct any errors found directly on the spreadsheet itself. The instructions page also contained some rules with regards to the data in the worksheets e.g. only employees with codes B or C are eligible to receive overtime pay.

Both the spreadsheet and the instructions were emailed to each of the industry-based professional subjects after they had been contacted and had agreed to take part. The student subjects were given the opportunity during a single 60 minute class period to participate in the study. No time limit was given to the professional subjects, but as pretests suggested, professional subjects completed the task in an average of 28 minutes, and student subjects in an average 36 minutes. The students knew in advance that a spreadsheet debugging exercise had been arranged, and the general feeling was that participants, both professional and student, approached the task as an interesting challenge. Subjects were told that cell activities were being recorded during the
debugging process, and that all individual results would remain confidential. Subjects were given contact details of the authors to request individual results.

4. INITIAL FINDINGS

Initial findings from analysis of the data recorded during the experiment and error correction results are detailed in the following sections.

4.1 Overall Results

Error Correction Rates

Industry-based professional subjects corrected 72% of all seeded errors and student subjects corrected 58% of all seeded errors. The results from Figure 1 show a clear distinction between performances of industry based professionals and students for Rule Violation and Formula errors, with professionals correcting 16% more formula errors than students and 20% more rule violation errors.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>No. of Seeded Errors</th>
<th>% Errors Corrected by Professionals</th>
<th>% Errors Corrected by Students</th>
<th>Professionals Compared to Students</th>
<th>[Howe &amp; Simpkin, 2006] Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clerical/Non-Material</td>
<td>4</td>
<td>17%</td>
<td>11%</td>
<td>+ 6%</td>
<td>66%</td>
</tr>
<tr>
<td>Rules Violation</td>
<td>4</td>
<td>85%</td>
<td>65%</td>
<td>+ 20%</td>
<td>60%</td>
</tr>
<tr>
<td>Data Entry</td>
<td>8</td>
<td>68%</td>
<td>63%</td>
<td>+ 5%</td>
<td>72%</td>
</tr>
<tr>
<td>Formula</td>
<td>26</td>
<td>79%</td>
<td>63%</td>
<td>+ 16%</td>
<td>54%</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>72%</td>
<td>58%</td>
<td>+ 14%</td>
<td>67%</td>
</tr>
</tbody>
</table>

Figure 1 – Error Correction Results

Error Correction Performance: Expert Vs Novice

The spreadsheet used in this experiment is almost identical (but for six less clerical and five more material errors) to that used by the authors of [Howe & Simpkin, 2006]. This allows for a detailed comparison between results. The mean error detection rate from [Howe & Simpkin, 2006] was 67%, with subjects detecting 66% clerical, 60% rule violation, 72% data entry, and 54% formula errors. Students from both experiments yielded similar results, with the exception of clerical errors. The spreadsheet used by [Howe & Simpkin, 2006] had 10 clerical errors, and students were informed that clerical/spelling errors may be on the spreadsheet; 66% were detected. Students and professionals from our experiment found only 11% and 17% of the 4 clerical errors respectively. They showed little interest in detecting them, and most professionals thought them irrelevant. When comparing the formula and rule-violation error detection rates of students in [Howe & Simpkin, 2006] and the professional subjects in our experiment, the professional subjects corrected 25% more formula errors and 25% more rule-violation errors.

Although the overall average error correction results shown in Figure 1 are similar to those of [Howe & Simpkin, 2006], the difference in error correction rates for some of the
error categories, particularly *formula errors*, between professionals from our experiment and students from both experiments is quite significant. This finding is contrary to the findings presented in [Galletta et al, 1993], in which spreadsheet experts did not outperform novices in detecting spreadsheet formula errors. The findings from [Galletta et al, 1993] suggest that spreadsheet expertise is not crucial for discovering errors strictly affecting spreadsheet formulas and structure. As the spreadsheet model used in our experiment was not complicated and did not require much, if any, domain knowledge, and given that the participating students in our experiment were themselves accounting & finance students (diminishing further the domain expertise factor), this leads to a conclusion that industry-based professionals with a good working knowledge of spreadsheets i.e. experts, find and correct more formula errors in less time than end-users with little or no industry-based spreadsheet experience i.e. novices.

One important element that may be a contributing factor to the differences in the findings of this experiment and that of [Galletta et al, 1993], with regards to spreadsheet expertise and *formula error* detection and correction performance, is the employment of self-reported measures to establish spreadsheet expertise used in [Galletta et al, 1993]. The professional subjects who took part in the experiment detailed in this paper were known to have a good industry-based working knowledge of spreadsheet use and development prior to their involvement in the experiment.

**Determining Possible Group Code-Inspection Phase Benefits**

In this experiment subjects inspected the spreadsheets individually; there was no group code inspection phase of any kind. In an attempt to determine what the average error correction yield might have been if individuals had been placed in groups of three, all combinations of three students were created and the number of separate errors corrected identified. The average performance across all combinations was calculated to represent the likely performance on average of a group of three students. This method assumes that there is no added benefit of working in groups of three beyond the sharing of information. The result of this process suggested that on average 81% of the errors would have been corrected if result pooling in groups of three was performed successfully. In a past study [Panko, 1999] the individual spreadsheet code inspection phase was followed by a group code inspection phase, using groups of three. It was found in that study that the gain from group-work came only from pooling the different errors detected previously by the individuals. The result of 81% is consistent with the 83% yield from group-work in [Panko, 1999].

**4.2 Cells Inspected Versus Debugging Performance**

With the data recorded during the spreadsheet debugging process it is possible to identify any cells that were inspected or edited. It is also possible to determine the number of times each cell was inspected and to determine the time spent inspecting each cell. An important research goal was to determine if there was a correlation between the number of cells inspected and error detection/correction performance. To answer this question, analysis was conducted to identify, for each subject, the number of individual cells inspected or edited during the debugging process. A cell was considered inspected/checked if that cell was selected for a specified minimum time or if the cell value/formula was edited or changed directly, and if the cell was within a specified range of cells. The specified ranges of cells for this analysis were cells that contained formulas/values. Blank cells and column/row headings were not included. For this analysis there were 44 usable results, as time data had not been recorded correctly for one of the students and two of the professionals.
Figure 2 shows a scatterplot for errors corrected versus coverage including a linear regression model for 44 subjects (no time data for one student and two professionals), where the minimum time specified for a cell to be considered inspected/checked is >0.3 seconds. It is evident from Figure 2 that a moderate-strong relationship exists between the number of cells inspected and error detection/correction performance: $R^2$ value of 0.6421.

![Figure 2 – Errors Corrected over Cells Inspected (>0.3sec)](image)

Figure 3 shows a scatterplot for errors corrected versus coverage including a linear regression model for the same 44 subjects, with the same specified cell range, where the minimum time specified for a cell to be considered inspected/checked is >1 second: $R^2$ value of 0.607 shows a similarly moderate-strong linear relationship.

![Figure 3 – Errors Corrected over Cells Inspected (>1sec)](image)
A possible application of this research may be as a predictor of spreadsheet reliability based on relevant past error density and debugging performance information and the percentage of critical cells inspected.

The remainder of the analysis focuses on the debugging behaviour of professional participants.

4.3 Unique Formula Debugging Analysis: IF Statement

Initial analysis was carried out on the time based aspect of the spreadsheet trial regarding professional subjects debugging behaviour of a unique IF formula. The IF formula, see Appendix A – Office Expenses F20, was as follows:

\[ \text{IF}(F10+F18>7000, "Exceeds Limit", "Within Limit") \]

The formula was incorrect, and should have been corrected by changing the 7,000 to 70,000. This change requirement was made known through instructions on the spreadsheet itself. Of the professional subjects, 38% amended the formula correctly. An aspect of professional subject’s behaviour that stood out when the resulting data logs were analysed, is that nearly all professional subjects checked/debugged the formula exactly twice, with none amending the formula correctly on their second visit to the cell. The average times spent inspecting the cell for professional participants who amended the cell correctly was 14 seconds on the first visit and 12 seconds on the second visit, those who did not correct the cell formula spent slightly less time inspecting the cell. Although time based analysis has not been carried out on this formula cell for the student subjects, results showed that 58% of the students amended this cell correctly. It may be the case that the professional subjects easily understood the logic of the IF formula, and missed the simple error by not inspecting the formula more thoroughly.

4.4 Remote Sheet Reference Debugging

Analysis was carried out on the behaviour of professional participants in debugging an incorrect formula that included a remote cell reference (a reference to a cell value on a different worksheet within the same workbook). The original and correct value of this formula, see Appendix A – Projections B19, is as follows:

<table>
<thead>
<tr>
<th>Original Formula</th>
<th>Correct Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>'=Office Expenses'!F10*4</td>
<td>'=Office Expenses'!F18</td>
</tr>
</tbody>
</table>

Thirty-eight percent of professional participants corrected this formula error. It was identified that although only 38% of participants discovered and correctly amended the formula, 46% discovered the error but amended it incorrectly. This finding implies that while error detection and error correction are both part of the debugging process, they are still two distinct disciplines, and an individual’s error detection capability should not be used as an automatic indication of their ability to correct errors found. Of the student subjects, only 12% corrected this error.

A reason for the very low correction rate for this formula could be that participants only discovered one of the two errors in the formula, and did not consider that there could be more than one error; a mechanical and a logical error. The cell value of Office Expenses'!F18 was a yearly estimate, calculated by a formula that added a range of 3-
month estimates which had been multiplied by 4. An incorrect cell was referenced - a mechanical error, and multiplying the remote cell value by 4 was a logical error; as the relevant remote cell value had effectively been multiplied by 4 already. Of the 46% of professional subjects who discovered the error but amended it incorrectly, 66% of them found the mechanical error but missed the logical error.

Microsoft Excel provides a useful show-precedents auditing tool for inspecting cell formulas, where arrows indicate cells that are referred to by a formula. A problem with this tool occurs when a precedent is on another worksheet. Excel simply lets the user know that a remote cell is being referenced by displaying an icon. Double clicking this icon allows one to go directly to the referenced cell, which entails leaving the sheet currently being inspected. This can be very confusing. The error described above may have had a higher correction rate if relevant information on the remote cell was available without leaving the worksheet currently being inspected. This information could include the remote cells value and formula, the column and row headings associated with the remote cell, the remote cells precedents etc. Further analysis regarding the most and least detected errors was conducted, but is not detailed here.

5. CONCLUSION

Spreadsheets are designed, built and used by a variety of users, many of whom are not professional programmers and are not inclined towards following or learning software development and testing methodologies. This is a major contributing factor to the unreliability of spreadsheets. With the aim of aiding end-users in improving spreadsheet reliability, many spreadsheet auditing and debugging tools have been developed and made available, but tools should support users’ natural debugging behaviour. This paper describes an experiment conducted with thirteen industry-based professionals and thirty-four accounting & finance students designed to unintrusively record end-user behaviour in spreadsheet error detection and correction activities. An experimental spreadsheet model was developed and subjects were asked to correct any errors found directly on the spreadsheet itself.

Overall results show that professionals (experts) are more efficient and effective spreadsheet debuggers than students (novices). Professional subjects outperformed student subjects in detecting and correcting errors of certain categories, namely formula errors, with a 16%-25% greater correction rate. Future analysis will aim at identifying the factors and behaviours that contribute to better debugging performance. An important finding is that a relationship exists between the percentage of critical cells inspected and the number of errors detected and corrected. In traditional software testing, predicting the reliability of software programs based on code coverage and defect density is a tried and tested method, which could possibly be applied to the spreadsheet paradigm. This study utilises a small, well-structured spreadsheet. But the question remains whether the findings can be applied to larger, poorly-structured spreadsheets. We believe that experts would outperform novices in debugging regardless, but that greater variance in debugging behaviour would occur with larger, real-world spreadsheets. The future aims of this study are to provide practical information for improving spreadsheet reliability by conducting further experiments and analysis in the near future, and possibly developing a spreadsheet debugging tool based on the experiment findings.

REFERENCES

An Empirical Study of End-User Behaviour in Spreadsheet Error Detection & Correction

Bishop & McDaid


174
Appendix A – Experimental Spreadsheet Model with Errors Colour Coded

**Error Colour Codes**

<table>
<thead>
<tr>
<th>Error Types</th>
<th>Colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clerical</td>
<td>Blue</td>
</tr>
<tr>
<td>Rule Violation</td>
<td>Yellow</td>
</tr>
<tr>
<td>Data Entry</td>
<td>Green</td>
</tr>
<tr>
<td>Formula</td>
<td>Red</td>
</tr>
</tbody>
</table>

**Payroll Worksheet**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Office Expenses Worksheet**

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

175
### An Empirical Study of End-User Behaviour in Spreadsheet Error Detection & Correction

#### Bishop & McDaid

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Choi Construction Company</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Today's Date</strong></td>
<td>04/06/2006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Projections Worksheet

<table>
<thead>
<tr>
<th>Current Year</th>
<th>Year 1</th>
<th>Year 2</th>
<th>Year 3</th>
<th>Year 4</th>
<th>Year 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Payroll Expenses</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category A</td>
<td>82% Payroll</td>
<td>-8151</td>
<td>1B34</td>
<td>-8151</td>
<td>1B34</td>
</tr>
<tr>
<td>Category B</td>
<td>-72% Payroll</td>
<td>-8161</td>
<td>1B37</td>
<td>-8161</td>
<td>1B37</td>
</tr>
<tr>
<td>Category C</td>
<td>-52% Payroll</td>
<td>-8171</td>
<td>1B39</td>
<td>-8171</td>
<td>1B39</td>
</tr>
</tbody>
</table>

| **Office Expenses** |        |        |        |        |        | 235600 |
| **Fixed Expenses**   | -Office Expenses'F19' | -6199 | 1B35 | -6199 | 1B35 | -6199 | 1B35 | -6199 | 1B35 | -6199 | 1B35 | -6199 | 1B35 |
| **Variable Expenses** | -Office Expenses'F19' | -6209 | 1B39 | -6209 | 1B39 | -6209 | 1B39 | -6209 | 1B39 | -6209 | 1B39 | -6209 | 1B39 |
| **Total**             | =SUM(F15:F20)       | =SUM(G15:G20)      | =SUM(H15:H20)      | =SUM(I15:I20)      | =SUM(J15:J20)      | 235600 |

| Does yearly increase in expenses exceed 4% | F122021b(1, 1, 'Yes', 'No') F122021b(1, 1, 'Yes', 'No') F122021b(1, 1, 'Yes', 'No') F122021b(1, 1, 'Yes', 'No') F122021b(1, 1, 'Yes', 'No') |

---

**Note:** The formulas in columns B to G are calculated based on the data provided in columns A to D. The formulas are designed to calculate the projected expenses for different categories and years, and to assess if the yearly increase in expenses exceeds 4%.
INCLUSION ANALYSIS

David Colver
Operis, 8 City Road, London EC1Y 2AA
dcolver@operis.com

ABSTRACT

Inclusion analysis is the name given by Operis to a black box testing technique that it has found to make the checking of key financial ratios calculated by spreadsheet models quicker, easier and more likely to find omission errors than code inspection.

1 CALCULATE MORE THAN ONCE

One approach among many [Pryor, 2004] [Panko, 2006] for improving the chance that a spreadsheet delivers the results that it is intended to deliver is to calculate quantities two or more different ways [Ettema et al, 2001]. If independent methods produce identical answers, there is a good chance, though no guarantee, that those answers are trustworthy. If the answers are not identical, the act of reconciling them and fathoming out what is causing the difference is often illuminating, not just about what mistake has been made, but about the underlying problem addressed by the spreadsheet and the assumptions being made in solving it.

Calculating a result by an alternative method is a standard weapon of a spreadsheet auditor. Most of the recent development focus of the Operis Analysis Kit [Oak, 2010], a software application for developing and reviewing spreadsheets public by the author’s own company Operis, has been directed at accelerating the production of parallel reconstructions and of identifying where and why they have diverged from the original model. At least one of the large international accountancy practices declines to perform spreadsheet audits [Croll, 2003], or does so only grudgingly, preferring to reconstruct the spreadsheet and reconcile the outputs. This process no doubt delivers outstanding results, but not every client values them enough to pay five times as much for them.

Spreadsheet developers don’t have to wait for an outside auditor to subject their spreadsheet to parallel reconstruction. They can calculate results repeatedly themselves, in their own spreadsheet. An obvious example concerns a table of numbers.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>R</td>
<td>C</td>
<td>G</td>
<td></td>
</tr>
</tbody>
</table>

TABLE 1

<table>
<thead>
<tr>
<th>T</th>
<th>Table of numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Column totals</td>
</tr>
<tr>
<td>R</td>
<td>Row totals</td>
</tr>
<tr>
<td>G</td>
<td>Grand total</td>
</tr>
<tr>
<td>A</td>
<td>All of the above</td>
</tr>
</tbody>
</table>

Copyright © 2007 EuSpRIG & The Author

177
The table has rows totals, column totals, and a grand total. It therefore offers opportunities to calculate the table total four times over. There are some simple things that ought to be the case. To pick just two:

$$\text{SUM}(R) = \text{SUM}(C)$$

(author’s favourite)  $$\text{SUM}(A) = G * 4.$$  

Since these tests are trivial to code, a conscientious spreadsheet developer will include them as a matter of course. A well constructed spreadsheet can easily devote 25% of its formulae to checking that relationships that should hold do in fact do so.

2  FINANCIAL STATEMENTS ADD UP

Many spreadsheet models concern themselves with the making of projections of the future financial statements of ventures that participants are contemplating launching or funding. Those financial statements have a happy property: they add up. At least, they ought to add up. The first steps in reviewing an unfamiliar financial model are

• to locate the financial statements (often quite a bit harder to do than one might imagine)

• to fathom out which entity the statements purport to relate to (again, not always crystal clear)

• because we are about to rely heavily on this property, to check that the financial statements add up.

Oversimplifying dramatically for clarity, we can say that both a cash flow forecast and a profit and loss account amount to a list of revenues and costs that result in a bottom line. The bottom line for the cash flow is some measure of cash generated, and that for the profit and loss statement is some measure of earnings retained. Symbolically

$$R - C = \text{bl}.$$  

where R represents Revenue, C represents Costs, and bl is the relevant bottom line.

If we follow the convention that costs will always appear on the financial statements as negative numbers, then we can write

$$R + (C) = \text{bl},$$

the brackets indicating that the costs are expressed as negatives. If now we break the list of revenues R into components $r_1$ to $r_4$, and the costs C into $c_1$ to $c_4$, we can write

$$r_1 + r_2 + r_3 + r_4 + (c_1) + (c_2) + (c_3) + (c_4) + (\text{bl}) = 0.$$  

Now we are free to regroup these terms in any clusters we like, for example

$$[r_1 + r_2 + (c_1) + (c_3)] + [r_3 + r_4 + (c_2) + (c_4) + (\text{bl})] = 0.$$  

Much the same arguments apply to a balance sheet. Because it balances,

$$A = L,$$

where A represents the assets and L represents the liabilities. And because it adds up too,
\[ a_1 + a_2 + a_3 + a_4 + (l_1) + (l_2) + (l_3) + (l_4) = 0 \]

where \( a_1 \) to \( a_4 \) are components of \( A \), and \( (l_1) \) to \( (l_4) \) are components of \( L \), expressed as negative numbers.

These items can be regrouped in arbitrary clusters, as before, such as

\[
[ a_1 + a_2 + (l_4) ] + [ a_3 + a_4 + (l_2) + (l_3) ] = 0
\]

### 3 INCLUSION ANALYSIS: SIMPLE EXAMPLE

These results are hardly profound enough to be placed on the current frontier of finance theory. But they are useful in developing a simple method for testing the ratios that are often what can be considered the outputs that financial models finally deliver after many megabytes of striving.

Consider a simple project in which shareholders invest in the capital costs of a factory. Some of the money it generates once up and running is delivered to the shareholders as dividends. The rest is retained to cover closing costs, and to repay to the shareholders their initial investment.

<table>
<thead>
<tr>
<th>Year</th>
<th>2008</th>
<th>2009</th>
<th>2010</th>
<th>2011</th>
<th>2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revenue</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Costs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>construction</td>
<td>(60)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>operating</td>
<td></td>
<td>(20)</td>
<td>(20)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decommissioning</td>
<td></td>
<td></td>
<td></td>
<td>(30)</td>
<td></td>
</tr>
<tr>
<td>Shareholders</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>initial investment</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dividends</td>
<td></td>
<td>(30)</td>
<td>(30)</td>
<td>(30)</td>
<td></td>
</tr>
<tr>
<td>return of capital</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(60)</td>
</tr>
<tr>
<td>Increase in cash at bank</td>
<td>-</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>(90)</td>
</tr>
<tr>
<td>IRR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to project</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>83.93%</td>
</tr>
<tr>
<td>to shareholders</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23.38%</td>
</tr>
</tbody>
</table>

The ratios at the foot of the table are the outputs of the model to which most attention will be paid, and it is important to check carefully that they have been correctly calculated. We will start with the project IRR.

The first step is to get the cash flow in such a form that it adds up to zero. The costs are already expressed following the convention that, since they are outflows, they appear as negative numbers. If this was not the case, we would invert them. We do, though, need to invert the line “Increase in cash at bank”, which is the equivalent in this example of what we referred to as \( bl \), or bottom line, above.

After inverting the bottom line and inserting a row to check that the cash flow really does total zero, we get the following. (Shaded lines show what has changed.)
If at this point we found that the “zero check” line did not show zeroes, we would know, either that some of the rows don’t follow our chosen sign convention, or that the cash flow didn’t add up. The latter possibility is more common than one might imagine. Sometimes large teams of highly paid bankers have worked on multi-billion dollar projects for a year or two without recognising this basic flaw in the projections upon which they have been relying.

Now we seek to reproduce the 83.93% IRR. That cell uses Excel’s IRR function to calculate the rate of return implicit in a row of net cash flows which is not shown in this table, and is to be found elsewhere in the model. We follow the IRR formula back to its precedents and reproduce those at the top of our spreadsheet. We then recalculate the IRR from these cash flows and show that we can reproduce the reported return.

### TABLE 4

<table>
<thead>
<tr>
<th>Year</th>
<th>2008</th>
<th>2009</th>
<th>2010</th>
<th>2011</th>
<th>2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero check</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Revenue</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Costs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Construction</td>
<td>(60)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating</td>
<td>(20)</td>
<td>(20)</td>
<td>(20)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decommissioning</td>
<td>(30)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shareholders</td>
<td></td>
<td>initial investment</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>dividends</td>
<td>(30)</td>
<td>(30)</td>
<td>(30)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>return of capital</td>
<td></td>
<td></td>
<td></td>
<td>(60)</td>
</tr>
<tr>
<td>Increase in cash at bank</td>
<td>-</td>
<td>(30)</td>
<td>(30)</td>
<td>(30)</td>
<td>90</td>
</tr>
<tr>
<td>IRR to project</td>
<td>83.93%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRR to shareholders</td>
<td>23.38%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### TABLE 3

<table>
<thead>
<tr>
<th>Year</th>
<th>2008</th>
<th>2009</th>
<th>2010</th>
<th>2010</th>
<th>2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero check</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Revenue</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Costs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Construction</td>
<td>(60)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating</td>
<td>(20)</td>
<td>(20)</td>
<td>(20)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decommissioning</td>
<td>(30)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shareholders</td>
<td></td>
<td>initial investment</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>dividends</td>
<td>(30)</td>
<td>(30)</td>
<td>(30)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>return of capital</td>
<td></td>
<td></td>
<td></td>
<td>(60)</td>
</tr>
<tr>
<td>Increase in cash at bank</td>
<td>-</td>
<td>(30)</td>
<td>(30)</td>
<td>(30)</td>
<td>90</td>
</tr>
<tr>
<td>IRR to project</td>
<td>83.93%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRR to shareholders</td>
<td>23.38%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Having established that we can reproduce the rate of return, we can be confident that the “relevant cash flow” is indeed the one that the model is using to calculate the project IRR. We now partition the elements of the original cash flow statement into two clusters, with the aim of showing which of the items form part of the “relevant cash flow, and which ones don’t. This is quite easy to do by eye in this example, though that is not always the case.

### TABLE 5

<table>
<thead>
<tr>
<th>Year</th>
<th>2008</th>
<th>2009</th>
<th>2010</th>
<th>2011</th>
<th>2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant cash flow</td>
<td>(60)</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>-</td>
</tr>
<tr>
<td>IRR calculated from above</td>
<td>83.93%</td>
<td>83.93%</td>
<td>0.00%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrepancy</td>
<td>included-relevant cash flow</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Included</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Excluded</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>-</td>
</tr>
<tr>
<td>Total of items</td>
<td>Included</td>
<td>(60)</td>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>ITEMS INCLUDED</td>
<td>Revenue</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Costs</td>
<td>construction</td>
<td>(60)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>operating</td>
<td>(20)</td>
<td>(20)</td>
<td>(20)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITEMS EXCLUDED</td>
<td>Costs</td>
<td>decommissioning</td>
<td>(30)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shareholders</td>
<td>initial investment</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dividends</td>
<td>(30)</td>
<td>(30)</td>
<td>(30)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>return of capital</td>
<td>(60)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increase in cash at bank</td>
<td>-</td>
<td>(30)</td>
<td>(30)</td>
<td>(30)</td>
<td>90</td>
</tr>
<tr>
<td>IRR</td>
<td>to project</td>
<td>83.93%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to shareholders</td>
<td>23.38%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The newly introduced lines, again shaded, serve merely to
• indicate which lines are in the included cluster and which in the excluded
• add up those included and excluded clusters
• verify that the included total matches the “relevant cash flow” at the top of the table
• verify that the included and excluded items combined add still add to zero, which is
  the duty that was previously performed by the “zero check” line.

Now we can interpret the result. The “Items included” section shows us that the
project IRR quoted concerns a cash flow that is made up of some revenues less some
costs. We could have learned that by looking at the formula used by the underlying
model to calculate the “relevant cash flow”; but we didn’t have to look at the formula
to work out what must be in it.

A project IRR made up of revenues less various costs sounds plausible. But now we
get a second bite of the cherry. We can look also at the items excluded, the ones that
are on the cash flow statement but don’t play any part in the project IRR. There are
the various involvements of the shareholders, since they are the providers of finance,
their participation is correctly excluded from a measure of the underlying project
economics. But also excluded is the cost of decommissioning the plant at the end of
its useful life. Leaving out the cost of abandoning the project is a material
misstatement of the project’s economics.

It would be hard, but possible, to examine the formula that derives the “relevant cash
flow” and notice that some cost is missing, if that cost appears in nearly every
financial projection. But not every financial model includes a decommissioning cost,
and it would be very hard to look at the formula and notice that one relatively small
and rather specialised cost is missing. Here, though, its exclusion is obvious.

The inclusion analysis demonstrates, in a prominent position near the top of the table,
that the excluded items are equal in magnitude to the included ones. To this extent, it
is exhaustive, in the sense that there can be no elements of the financial statement
which are not considered for possible inclusion in the ratio.

4 ANOTHER EXAMPLE

Having tested the project IRR, we can repeat the process for the shareholders’ IRR.
All we have to do is
• copy the analysis already completed
• relink the line “relevant cash flow” so that it points to the source of the cash flows
  used in the equity return calculation
• verify that we can reproduce the shareholders’ IRR from those cash flows
• move the lines around between the included and excluded sections until we can
  explain the new relevant cash flow.
<table>
<thead>
<tr>
<th>Year</th>
<th>Relevant cash flow</th>
<th>2008</th>
<th>2009</th>
<th>2010</th>
<th>2011</th>
<th>2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRR</td>
<td>calculated from above</td>
<td>23.38%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>reported below</td>
<td>23.38%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>discrepancy</td>
<td>0.00%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrepancy</td>
<td>included-relevant cash flow</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>included+excluded</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total of items</td>
<td>included</td>
<td>60</td>
<td>(30)</td>
<td>(30)</td>
<td>(30)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>excluded</td>
<td>(60)</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>-</td>
</tr>
<tr>
<td>ITEMS INCLUDED</td>
<td>Shareholders</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>initial investment</td>
<td></td>
<td>(30)</td>
<td>(30)</td>
<td>(30)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dividends</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITEMS EXCLUDED</td>
<td>Revenue</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Costs</td>
<td>(60)</td>
<td>(20)</td>
<td>(20)</td>
<td>(20)</td>
<td>(30)</td>
</tr>
<tr>
<td></td>
<td>construction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>operating</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>decommissioning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shareholders</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(60)</td>
</tr>
<tr>
<td></td>
<td>return of capital</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Increase in cash at bank</td>
<td>-</td>
<td>(30)</td>
<td>(30)</td>
<td>(30)</td>
<td>90</td>
</tr>
<tr>
<td>IRR</td>
<td>to project</td>
<td>83.93%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>to shareholders</td>
<td>23.38%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Here we can see that the cash flow whose rate of return is being tested is the dividends received by the shareholders, net of their initial investment. This looks reasonable enough. But it isn't until we inspect the items excluded from the ratio calculation that we notice that the eventual return of capital to the shareholders has been omitted from the ratio. It's one of the cash flows experienced by the shareholders and certainly belongs in the ratio.

### 5 BALANCE SHEET RATIOS

Exactly the same method can be used in tests involving balance sheet quantities. As we've seen, balance sheets conveniently sum to zero if the liabilities are expressed as negative numbers.
Consider, for example, the following simple balance sheet.

**TABLE 7**

<table>
<thead>
<tr>
<th>ASSETS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed assets</td>
<td>100</td>
</tr>
<tr>
<td>Current assets</td>
<td></td>
</tr>
<tr>
<td>cash</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>101</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LIABILITIES</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Debt</td>
<td></td>
</tr>
<tr>
<td>senior loan</td>
<td>79</td>
</tr>
<tr>
<td>equity bridge loan</td>
<td>10</td>
</tr>
<tr>
<td>shareholder loan</td>
<td>5</td>
</tr>
<tr>
<td>Equity</td>
<td></td>
</tr>
<tr>
<td>share capital</td>
<td>5</td>
</tr>
<tr>
<td>retained earnings</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>101</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RATIOS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Debt:equity ratio</td>
<td>89%</td>
</tr>
</tbody>
</table>

We wish to prove the debt:equity ratio, reported as 89%. As before, our first step is to restate the balance sheet so that it adds to zero. We invert the liabilities, remove the balance sheet footings and extraneous headings, and put in a test that it does adds to zero.

**TABLE 8**

<table>
<thead>
<tr>
<th>Zero check</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed assets</td>
<td>100</td>
</tr>
<tr>
<td>Current assets</td>
<td></td>
</tr>
<tr>
<td>cash</td>
<td>1</td>
</tr>
<tr>
<td>Debt</td>
<td></td>
</tr>
<tr>
<td>senior loan</td>
<td>(79)</td>
</tr>
<tr>
<td>equity bridge loan</td>
<td>(10)</td>
</tr>
<tr>
<td>Equity</td>
<td></td>
</tr>
<tr>
<td>share capital</td>
<td>(10)</td>
</tr>
<tr>
<td>retained earnings</td>
<td>(2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RATIOS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Debt:equity ratio</td>
<td>89%</td>
</tr>
</tbody>
</table>
Now we examine the debt:equity formula and seek to reproduce its result.

### TABLE 9

<table>
<thead>
<tr>
<th>Ratio components</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>top of fraction: debt</td>
<td>(A)</td>
<td>79</td>
</tr>
<tr>
<td>bottom of fraction: debt + equity</td>
<td>(B)</td>
<td>10</td>
</tr>
<tr>
<td>Debt:equity ratio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>recalculated from above</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reported below</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discrepancy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Debt:equity ratio reported below 

Now that we know what elements make up the ratio, we can seek to match those to the elements of the balance sheet, identifying what participates in the top of the fraction, what participates in the bottom, and what takes no part in it. The result is in Table 10, over the page.

As before, the items that are included in the ratio under discussion are interesting, but every bit as interesting are the items that play no part in the ratio. One of these is the equity bridge loan. Spotting that, we can ask immediately, if the senior loan considered to be debt for the purposes of this ratio, shouldn’t the equity bridge loan be considered too? Or maybe it should be part of the equity, since it is standing in for investment that would otherwise be provided by the shareholders, and is being guaranteed by them?

There’s a case to be made for either alternative, but not one for ignoring the bridge loan altogether. Leaving equity bridge loans out of ratios is the serious fault of the moment, which has almost derailed several recent deals. Like other errors of omission, it is spotted quickly and with certainty by an inclusion analysis, and is relatively hard to spot by simply studying the formulae.

While we are at it, we can ask whether the ratio is correct to consider the share capital as the only kind of equity, or whether it should not also address the retained earnings, which the inclusion analysis shows firmly to have been excluded from the calculation.
This inclusion analysis is a little more sophisticated than the earlier ones because it considers the two elements of a ratio rather than a single quantity. The author terms them three-way inclusion analyses.

6 MISMATCHES

With luck, one can quickly find how the elements of a financial statement need to be partitioned between included and excluded amounts to reproduce the components of a ratio. Then all one has to do is to study the result, and work out whether the included items really belong inside the ratio, and (the key point of the analysis) the excluded items really belong outside.
But often there is no combination of the rows in the financial statement that will reproduce the sought-for numbers.

- One reason is that the model being tested is simply defective. It has marshalled the wrong items in assembling the ratio.

- Another reason is that the financial statements are too coarse-grained. For example, a model may correctly factor some costs into a ratio but not others. Under these conditions, it becomes necessary to replace the single operating costs line by its components, and to allocate those between the included and excluded groupings. All the time, the rule that the financial statement add up, and the fact that there is a test at the top of the analysis constantly monitoring what follows for this crucial property, are aids in ensuring that the decomposing of lines is done quickly and correctly.

- A further possible reason is that the model has correctly deviated from the financial statements, to reflect some fine detail of the transaction. Under these conditions it becomes necessary to add extra lines to the analysis to show what adjustments are necessary to match the reported ratios. It is rare that more than three or four lines of adjustment are necessary.

The three-way variant also makes it obvious when items appear on both the top and the bottom of a ratio. Measures of debt cover are particularly prone to this kind of double counting. They compare the cash available to service a business’s debt with the cash needed to service that debt. Here, “service the debt” means pay the interest and repay the loan principal. Unfortunately, these aren’t the only payments demanded by a bank. Banks also ask for all manner of impertinent fees. Are the fees part of the debt service, or part of the operating costs that are deducted in working out the cash available to service the debt? In all too many models, they appear in both, a fault that can be detected by formula inspection only by an auditor who knows what to look for, and who remembers to do it, but which can be detected by an employee with a few days’ training through inclusion analysis.

7 WIDER USES OF INCLUSION ANALYSIS

Any situation where some, but not all, of the items in a group contribute to a result is a candidate for inclusion analysis. As just shown, financial ratios typical involve comparing some combination of lines on a financial statement with some other combination. But there are other applications too.

- Under many jurisdictions around the world, some costs are eligible for deductions in calculating the profits on which corporate taxes are levied, and some are not. An analysis showing which costs have been included in a tax calculation, and so treated as eligible for deductions, and which have been excluded from participation in the calculation, and so treated as ineligible, can easily be presented as an inclusion analysis. It will also show which revenues are subject to tax and which are exempt from it.

- Similarly, Value Added Tax is applied in Europe to some revenues and costs but not others. An inclusion analysis will show which is which.

- Every input cell in a model can be partitioned between those that influence operating income or pre-tax cash flow, and those which don’t. An inclusion analysis distinguishes the two and quickly shows if inputs have been provided for costs or revenues that don’t make their through the model appropriately.
These kinds of inclusion analysis are less straightforward to perform than the ones illustrated in this paper, but they have the same power to detect errors of omission. The trick is to make a habit of doing them, so that they become routine with practice, part of the atmosphere and culture of the team. Operis has managed to institutionalise them. Every incoming analyst is taught inclusion analysis on his third day with the firm, and every model that is reviewed has inclusion analysis applied to its inputs, to every ratio it produces, and to every tax calculation during its first few hours of review.

8 RELEVANCE TO SPREADSHEET TESTING

Inclusion analysis is a useful technique. It is easy to do, and can be taught to analysts with relatively little experience. It is easy to check their work because it is easy to specify a standard layout for the analysis.

The process of partitioning the financial statement into included and excluded items can also be trivially automated, with about thirty lines of Visual Basic more than enough to rearrange, in a handful of seconds, the lines of a cash flow statement that sums to zero, so that it shows the split between included and excluded items, and presents the result in a prescribed form. (Or, in some cases, to show that no such rearrangement is possible, which is generally a sign of a fault in the model under test.) However, it is good for the soul and education of the consultant to do the work by hand, takes hardly any longer, and leaves him with a deeper connection to and familiarity with the model being investigated.

Inclusion analysis quickly exposes issues that are hard to find otherwise, particularly ones concerning mismatched timing (comparing 2008 equity to 2007 debt). The excluded part of the analysis is a list of all of the candidates for errors of omission, which are notoriously hard to detect.

In theory, the included part of the analysis gives no information that can’t be gleaned by examining the spreadsheet formula. However, it is not uncommon for a cost to be deducted once in one formula, only to be deducted again in another formula further along the chain of calculation, or perhaps to be added back again, so that rather than being counted once, the cost is counted zero or two times. Such faults are very hard to detect by formula inspection but immediately obvious in the inclusion analysis. Frequently, when model authors are told that they have double counted or omitted a cost, they report that they can’t see where the fault is. Nor can the producer of the inclusion analysis, without looking rather carefully, but he can know with certainty that it must be there somewhere.

Ventures often undergo changes as they proceed. A contract may be subject to an extension or variation, or a company may be refinanced or sold. Under such circumstances, it is common for a financial model to be updated, and for the changes to be subjected to audit. In this way, financial models are often audited formally several times during their lives, giving model auditors the opportunity to revisit their own work of years ago, or recheck work that other firms have performed. Any issues that surface ought to be confined to the portions of model that have changed, since previous audits should have unearthed all the problems that applied to prior versions of the spreadsheet. However, Operis has found that inclusion analysis almost always quickly unearths issues that had been missed by the earlier exercises.

All these are practical benefits. But inclusion analysis is thought provoking in an academic sense too. The best research indicates that the most reliable way to detect
spreadsheet errors is to undertake formula inspection, ideally in pairs or groups. This conclusion is supported by careful testing and measurement, something valuable but rare in this field.

“Although many prescriptive techniques have been put forth, only code inspection has been tested experimentally and has proven to be both safe and effective”. [Panko, 2005]

An inclusion analysis delivers a test that is equivalent to the formula inspection. There is often a perfect mapping between the included elements and the formulae leading to the ratio, one that is made obvious if the formulae are expressed in terms of meaningful names rather than spreadsheet coordinates. At the same time inclusion analysis delivers a second test, the excluded element, that has no analogue in code inspection. We don’t have tests and measurements to prove it, only the anecdotal experience just cited; but it seems unlikely that a pair of complementary tests, demonstrably exhaustive, can be outperformed by a single test equivalent to just one of the pair.

9 SUMMARY

Not every spreadsheet result involves summarising or comparing (in a ratio) quantities that are equivalent to some but not all of the lines in a financial statement or other table of numbers that adds up. But where a spreadsheet result does have these properties, the opportunity is available to check the calculation two different ways.

One of the ways will be equivalent to inspecting the formula. The alternative way amounts to examining the formula’s mirror image, a statement of all the items that have not been chosen for inclusion in the ratio component. This amounts to a list of all the potential errors of omission from the calculation, which is valuable because omissions are by their nature hard to identify.

10 REFERENCES


ABSTRACT

The risks of spreadsheet use do not just come from the misuse of formulae. As such, training needs to go beyond this technical aspect of spreadsheet use and look at the spreadsheet in its full business context. While standard training is by and large unable to do this, task-based training is perfectly suited to a contextual approach to training.

1 THE SITUATION

1.1 What Are The Risks Of Spreadsheet Use?

The main risks associated with spreadsheet use are the prevalence of errors (which can cause inadvertent error and make fraud harder to detect) and the gross misuse of time by spreadsheet operators (designers, users, etc.).

The effects of spreadsheet errors are well documented: bad decision making, non-compliance, ease of fraud etc. “The biggest fear for organisations is losing data or suffering losses due to unchecked inaccurate data.” [Baxter, 2007] It has been reported that as many as 90% of spreadsheets contain errors and that many of those errors can be critical to business [Bewig, 2005]. The EuSpRIG website [EuSpRIG] contains many reports of specific instances of business loss due to spreadsheet error.

The effects of time-wasting are not as well documented, as they hit businesses in a more indirect manner. However, staff productivity is definitely affected by fighting an unfriendly system and by wasting hours doing things a long way. This waste can cost money.

1.2 What Are The Causes Of These Risks?

There are two levels at which spreadsheet errors occur: the user level and the business level.

User Error

We all make mistakes regardless of whether we are designing, auditing or using a spreadsheet system. While natural human error cannot be eradicated by training, an awareness of possible errors can assist with detection and remedy [Purser, Chadwick, 2006].
Lack of technical skills (e.g.: choosing the wrong 4th argument in a vlookup) and lack of awareness of software features (e.g.: cutting and pasting swathes of data rather than using lookup functions) can be addressed by training.

**Business Error**

For most business people – wherever placed in the hierarchy – there is a lack of awareness of what their colleagues actually do. Most users do not know where the information in their spreadsheet comes from or goes to. This can lead to errors in the boundaries between work spaces, roles or tasks. Examples include using misclassifications of products, users repeating calculations because they are unaware that another has the summary they actually need, etc.

Many of these business errors are logical errors or errors of omission, which Panko has stated are more difficult to detect and potentially more dangerous than the mechanical errors characterised by user error. However, all three are damaging and prevalent [Panko, 2005].

## 2 THE SOLUTION

We believe that task-based training is required for most effective management of spreadsheet systems in any organisation.

### 1.3 What Is Task-Based Training?

Task-based training looks at the task that needs to be done, rather than looking in isolation at the way the software works.

Task-based trainers have experience in the way spreadsheets are actually used in the real world rather than a theoretical idea of how they should be used. They combine this experience with knowledge of the client’s circumstances.

The task is put in the context of the business processes that surround it. For example, people are not taught a conditional sum as a spreadsheet feature, they are taught how to report sales activity by client for the past month – it may be that the best way to do that is a conditional sum, a pivot table, subtotals, a macro or perhaps even using a database. That decision is based on factors such as the nature of the organisation, the source-data, the output requirements and the user.

Much of the training that Small Spark conducts is task-based, though we do also engage in standard training. We have talked to our clients and some of their feedback has been included in this paper.

### 1.4 What Are The Alternatives To Task-Based Training?

Task-based training is not the only type of training and is far from the most popular.

Off-the-shelf training packages are the most common form of training. They usually divide training provision broadly by perceived skill level. They are often package and version specific. As they frequently include people with different uses of spreadsheets, the training must be general and is therefore unable to adapt to the particular needs of the trainees.
Why Task-Based Training is Superior to Traditional Training Methods
Kath McGuire

While this is very common, most spreadsheet users have undergone no formal training whatsoever. Many learn on the job from their predecessor or colleagues and some are self-taught.

1.5 Why Is Task-Based Training Superior?

By the end of a task-based training session, trainees will have completed tasks, have set up systems for future tasks, have seen some of their problems solved and have worked on actual spreadsheets. This makes task-based training more effective. The trainee is learning both the mechanical skills associated with the task (the conditional sum, the data pilot, etc.) as well as the reason for use of this particular skill in a given instance. As such the skills being taught can be applied not only in the current circumstance but in further spreadsheet work involving different tasks.

Since task-based training by its very definition takes account of the business processes in which the users finds themselves, a task-based trainer is able to be more effective than a standard trainer of a comparable skill level. When we have engaged in standard training we have received feedback which highlights the drawbacks of standard training in comparison to task-based training: “some a-ha moments that I will find very helpful and then other parts which just seemed a long way of doing things I already do”. In a task-based training session it is possible (and in fact essential) to find out if a solution is appropriate for the trainee and then train them in what is appropriate for them.

When one of our clients was asked about her attitude to off-the-shelf training as opposed to task-based training she replied: “if I went to one of their half-day courses or evening classes, they may not cover exactly what I’m looking for”.

Task-based training can help to ensure that the right spreadsheet solution is implemented appropriately by the right people. Knowing about the function is not enough – it is important to know when to use it: “We covered things that I always knew were there but never knew how to access”.

Also, like other training provision, task-based training can help with user errors such as incorrect use of functions. It is possible to build the user’s confidence by fully explaining the best solution and how to implement it appropriately for their task. One of our clients has said about her staff “They know they’re doing the right thing and I think it makes them more productive”.

Spreadsheets in Context

In order to design and implement an effective spreadsheet solution, it is important to have business knowledge to ensure that data is turned into useful information, and technical knowledge to turn the data into information in a robust and efficient manner.

The business knowledge exists in-house. Technical knowledge is sought through training. If this technical knowledge does not take account of the business knowledge, the solution will not be optimal. Standard training seeks to provide technical skills within a vacuum. Task-based training takes account of the business knowledge to ensure that the appropriate technical skills are imparted.

A client had an Excel-based data manipulation task. She was new to the organisation and had been given very little induction. Her boss knew what needed to be done (business knowledge) but did not know how to do it. The technical staff could have created an
algorithm for the data processing but could not have implemented it in Excel. Standard
training is at a loss in such situations as the trainee does not know which features would
be useful. Task-based training is vital because the business knowledge can be
incorporated into the technical spreadsheet training. “You took it apart and removed the
mystique from it … It's a nice routine. It takes seconds now.”

Business knowledge provides information about the business context of the spreadsheet,
the skills of the users, as well as hardware and software considerations of the business.
All of these affect the design, use and improvement of spreadsheets. Business knowledge
is important to providing relevant training: “you understand my business and that makes
a real difference to the training”.

Business context

The business context of the spreadsheet includes how information flows through the
organisation and who needs to access the information. This can have an impact on
whether existing spreadsheets are rewritten or streamlined.

The correct tools will vary depending on the size and scope of the solution. While pasting
data is usually a bad idea – and standard training should warn against it – it may be that it
is the best solution for a particular system. Task-based training allows for this and the
trainer can explain the relevant dangers and suggest appropriate techniques to mitigate the
risk (such as check sums).

Skill awareness

Training a few key employees in VBA or other ‘advanced’ techniques – pivot tables in
Excel, the data pilot in OpenOffice, etc. – is often popular. But if their colleagues cannot
use these features, this knowledge is not adding value to the organisation. Using lookups
to generate many static tables might be more suitable if this reflects the skill level of all
users. This is a decision that standard training cannot make, and that task-based training
must make.

It is important that the spreadsheet solution being implemented is sufficiently user-
friendly and allows for all the actions a user might want to take, so as to remove the need
for hacking around it. This sort of design consideration is much easier to teach within a
task-based training framework that takes into account the requirements and personalities
of all users.

Software and Hardware

Not all spreadsheet users use the same packages or versions. This variation of software is
found within organisations, not just between them. Task-based training is better able to
appreciate differences in software between users, or between machines used by the same
user.

Hardware and version control systems (if any exist) can also affect the size, speed and
scope of the right solution. A standard training provider is unable to train with this in
mind whereas it is information that a task-based trainer does have access to.

Relevant Error Management

Teaching people what spreadsheets can do is the most important part of training, but it is
also vital to teach people what they cannot do. The limitations and quirks of solutions are
often not explained in training because the nature of these limitations may only become apparent when the solution is used in a particular context.

Task-based training can pre-empt many of these limitations and can make allowances for them. Error checks can be built in, spreadsheets can be commented and staff can be taught what sort of errors to look for and how to spot them.

The nature of task-based training means that external eyes are looking at the solution and can spot actual or potential errors that may elude the existing users.

Documentation

Documentation of spreadsheets is important and unusual. When documentation is created, it is frequently consulted initially but then filed away and forgotten.

Task-based training discusses processes. It questions why things are done the way they are done and brings to the forefront the reasons behind the processes. When notes are provided as part of a task-based training session, they can become a part of the documentation of the solution.

3 THE CONCLUSION

The problems with spreadsheet use go beyond teaching someone how to use a particular spreadsheet function. The right skills to teach and the right solution to design can be appreciated most effectively through task-based training provision.

Task-based training is better able than standard training provision to effectively mitigate the drawbacks of spreadsheet use for ALL spreadsheet users (regardless of skill level, seniority, qualifications, job description, etc.) in ALL organisations (regardless of size, complexity, training budget, etc.).

4 REFERENCES

I’d like to thank Small Spark’s clients who have given their feedback on both our task-based and our standard training provisions.


Bewig, P (2005) “How do you know your spreadsheet is right?” http://www.eusprig.org/hdykysir.pdf 17:30 09/06/07


EuSpRIG website http://www.eusprig.org

Panko, R (2005) “What We Know About Spreadsheet Errors” http://panko.cba.hawaii.edu/ssr/Mypapers/whatknow.htm 17:30 09/06/07

Establishing A Minimum Generic Skill Set For Risk Management Teaching
In A Spreadsheet Training Course
Chadwick D.,

School of Computing and Mathematical Sciences,
University of Greenwich, London, SE10 9LS, UK
D.R.Chadwick@gre.ac.uk

ABSTRACT

Past research shows that spreadsheet models are prone to such a high frequency of errors and data security implications that the risk management of spreadsheet development and spreadsheet use is of great importance to both industry and academia. The underlying rationale for this paper is that spreadsheet training courses should specifically address risk management in the development process both from a generic and a domain-specific viewpoint. This research specifically focuses on one of these namely those generic issues of risk management that should be present in a training course that attempts to meet good-practice within industry. A pilot questionnaire was constructed showing a possible minimum set of risk management issues and sent to academics and industry practitioners for feedback. The findings from this pilot survey will be used to refine the questionnaire for sending to a larger body of possible respondents. It is expected these findings will form the basis of a risk management teaching approach to be trialled in a number of selected ongoing spreadsheet training courses.

1. QUESTIONNAIRE AS A DATA GATHERING TECHNIQUE

The questionnaire reported upon herein was a pilot survey sent to ten persons of whom six responded. It was intended that the results from this would be used to create a fuller more comprehensive questionnaire for eventual sending to at least 100 possible respondents. To establish the skills that needed to be covered in generic spreadsheet risk management training the pilot questionnaire was constructed around a minimum set of 16 questions addressing five training methods to be used and eleven generic skills to be taught. The questionnaire was also constructed with facility for respondents to suggest further areas of concern that needed to be covered. For accuracy, the pilot survey had to simulate the eventual final questionnaire as closely as possible so its preamble, questions, and distribution method were carefully considered. For the theoretical research perspective and research paradigm see appendix C.

2. THE PILOT QUESTIONNAIRE

2.1 The Six Sections

1. Pre-amble: to explain the rationale of the questionnaire to the respondent.

2. Generic Training Methods: a sample of 5 generic training methods suggested by the author. Respondents were asked to comment upon these and rank their suitability for use in a training course by marking a 1-5 Likert scale.

3. Generic Training Content: a sample of 11 generic spreadsheet training content suggestions suggested by the author. Respondents were asked to comment upon these and rank their suitability for inclusion in a minimum generic skill set for a training course by marking a 1-5 Likert scale.
4. ‘Anything else?’ section: five open sections were deliberately added to encourage respondents to add suggestions of their own. Such feedback considered essential as it was expected that not all pertinent issues had been addressed.

5. ‘Some Information About You’ section: collecting data about the respondents themselves e.g. for respondents who were trainers:

<table>
<thead>
<tr>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Please identify what kind of a trainer you are:</td>
</tr>
<tr>
<td>What specific areas of modelling do you teach?</td>
</tr>
<tr>
<td>May I approach you again to discuss your answers?</td>
</tr>
</tbody>
</table>

And for respondents from industry:

<table>
<thead>
<tr>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>What industry are you involved with?</td>
</tr>
<tr>
<td>Do you think that spreadsheet training approaches should be improved?</td>
</tr>
</tbody>
</table>

6. The questionnaire was ended with a completely open section for further comments.

The initial 16 questions of the survey, the rationale for their inclusion, their drawbacks and supporting references are to be found in the table in Appendix A. The actual questionnaire is to be found in Appendix B.

2.2 Presentation Of Questions

It was initially considered to present the questionnaire without an explanatory preamble as the reading of this would take up respondent time. However, a short preamble was eventually included to clearly set the scene for the potential respondent.

The two initial sections of the questionnaire were: ‘Generic Training Methods’ and ‘Generic Training Content’. Each of these was included with an explanation of what these terms actually meant so misunderstandings could be limited – see example 1 below.

<table>
<thead>
<tr>
<th>GENERIC TRAINING METHODS</th>
<th>Aim: every course should have at least one instance of the following practices to raise student’s awareness of error situations and to develop self-reflective practices.</th>
</tr>
</thead>
</table>

Example 1: Explanation of section ‘Generic Training Methods’

Similarly each question about a content or method to be considered was accompanied by a full descriptor – see example 2 below.

<table>
<thead>
<tr>
<th>Peer-audit (non-participative)</th>
<th>Student to find errors (if any) in another student’s model</th>
</tr>
</thead>
</table>

Example 2: Example of question descriptor for question 2

Respondents were encouraged to give their own suggestions in the ‘Anything else?’ sections of the questionnaire and in the last section entitled ‘Please make any further comments below’ – see Appendix B for the Actual Questionnaire sent out.
2.3 Scoring The Questionnaire

2.3.1 Yes/No answers: were considered but discounted because a Yes/No answer, although easier to use for statistical purposes, is too coarse a discriminator of opinion. Similarly, the use of all open questions, although possibly fruitful in new data, were also decided against as respondents may have found them too time consuming with resultant poor return response and/or a poor image of the survey.

2.3.2 Likert scale: was therefore adopted for scoring the questionnaire. Each question reply was given a weighting of importance by use of a Likert scale 1 - 5. Five grades were considered sufficient to obtain worthwhile discrimination – less would have been too coarse. A guide to answering was also shown e.g.

1=Not needed  2  3=Indifferent  4  5= Must have

The end and middle points had concise explanations included: ‘Not Needed’ and ‘Must Have’ were diametric opposites and the mid-point was deliberately chosen as ‘Indifferent’ rather than left to open interpretation as say ‘Ok but would leave out if something better came along’ or ‘Ok but may be optional’.

2.3.3 Free text: a section was included at the end of the questionnaire were included to enable more open answers, suggestions and comments. Interpretation of open section responses was done carefully to avoid personal bias e.g. bias could be introduced by the author taking on board only those comments he liked and so giving a personal weighting to the importance of the comment.

3. ANALYSIS OF THE COMPLETED QUESTIONNAIRES

3.1 Ranking

The Likert scale made analysis fairly straightforward especially when results of the six respondents were placed into an Excel spreadsheet for analysis – see Appendix D. Table 1 shows the final ranking of the sixteen issues in order of the totals of the Likert scale grading. This indicates the order of importance to the pilot survey’s six respondents. It is clear from the responses that the issues mentioned in the questionnaire have different importance to the respondents – the use of Integral Documentation appears to be the most significant issue to be addressed with the teaching of a Taxonomy of Errors as the least important and by a wide margin.

3.2 Free Text Comments

In addition to the feedback on the initial 16 suggestions, the respondents also gave free text comments under the ‘Anything else?’ prompts on the questionnaire sheet. Some of these were useful and covered material initially overlooked – see Appendix E. These suggestions will be included as specific areas in the final more comprehensive questionnaire to be given to a wider audience at a later date.

3.3 Differences In Respondent Cohort

Interestingly, there was a slight but noticeable difference between the answers of the trainers cohort (Total Likert Score 223) and that of the business persons cohort (Total Likert Score 206) – see Appendix D. The latter appeared more cautious in giving a 5-rating (the highest) to any suggestion. It is not clear what this indicates – further research may be necessary. In addition, question 6 about inclusion of ‘Taxonomy of common errors’ had the most marked differential with trainers giving a total score of 13 against a score of 8 from the business persons.
Establishing A Minimum Generic Skill Set For Risk Management Teaching
In A Spreadsheet Training Course
Chadwick D.,

<table>
<thead>
<tr>
<th>Original Question Number</th>
<th>Question Title</th>
<th>Question Descriptor</th>
<th>Total Likert Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Integral Documentation approach</td>
<td>Documentation within the spreadsheet itself.</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>Error-seeded models</td>
<td>Student to find errors in tutor constructed model</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>Case Study</td>
<td>Student to discuss real-world models, and possible error situations</td>
<td>29</td>
</tr>
<tr>
<td>10</td>
<td>Auditing Tools (Integral)</td>
<td>Built-in audit functions i.e those integral to Excel</td>
<td>29</td>
</tr>
<tr>
<td>12</td>
<td>Access Control procedure</td>
<td>Password mechanisms etc</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>Peer-audit (non-participative)</td>
<td>Student to find errors (if any) in another student’s model</td>
<td>28</td>
</tr>
<tr>
<td>15</td>
<td>Formulae Hard-coding controls</td>
<td>Guides to when hard-coding may be permitted and not permitted e.g universal constants in physics?</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>Spreadsheet engineering methodology</td>
<td>An stepped approach of some kind to aid a student in during spreadsheet building</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>Version Control approach</td>
<td>A structured approach to naming and storing past models</td>
<td>27</td>
</tr>
<tr>
<td>16</td>
<td>Named Ranges</td>
<td>When to be used or not</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>Self-Audit</td>
<td>Student checks own work, makes a statement as to how correct they think it is prior to tutor assessment</td>
<td>26</td>
</tr>
<tr>
<td>14</td>
<td>Formulae length limitations .</td>
<td>Heuristics to limit formulae length e.g. no formulae should have more than 8 operators</td>
<td>26</td>
</tr>
<tr>
<td>9</td>
<td>Confidentiality Controls</td>
<td>Spreadsheet encryption methods</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>Peer-audit (participative)</td>
<td>Student to find deliberately placed errors in another student’s model</td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>Auditing Tools (External)</td>
<td>Commercial audit tools and Excel add-ins e.g. OAK, SpACE</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>Taxonomy of common errors.</td>
<td>Classification of common errors that they can add their own errors to over time</td>
<td>21</td>
</tr>
</tbody>
</table>

Table 1: Questions sorted in order of Total Score on the Likert scale (from Appendix D).

4. CONCLUSION

The pilot questionnaire has given an indication of some of the generic skills of spreadsheet risk management that need to be included in a good-practice training course along with some of the training methods that should also be included. Not all the pertinent issues were mentioned in the original questionnaire as the free-text responses show. The results of the pilot questionnaire, along with ideas and suggestions from the free-text comments will be used to construct a more comprehensive questionnaire which will be sent to a larger set of potential respondents. These findings will in turn be used to establish a set of criteria for defining ‘good-practice’ in the training of spreadsheet risk management wherever this may occur.
5. REFERENCES


[7] Rogers A.; Teaching Adults; Open University Press, 2002


[10] EuSpRIG 2006 ; Yahoo Group Discussion Forum, Nov 2006; Formulae Length thread


## APPENDIX A: INITIAL QUESTION SUGGESTIONS:
### RATIONALE, DRAWBACKS AND REFERENCES

<table>
<thead>
<tr>
<th>SUGGESTIONS</th>
<th>RATIONALE</th>
<th>DRAWBACKS</th>
<th>REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Error-seeded models e.g. student to find the errors</td>
<td>Game scenario. Challenging, competitive.</td>
<td>Student sees only as a game – does not draw inference to real life</td>
<td>[4], [5], [6]</td>
</tr>
<tr>
<td>2 Peer-audit (non-participative) e.g. find the errors (if any) in another student’s model</td>
<td>Both parties benefit. Modeller pays more attention, checks model prior to audit.</td>
<td>Modeller embarrassment if any error is found?</td>
<td>[4], [7]</td>
</tr>
<tr>
<td>3 Peer-audit (participative) e.g. find deliberately made errors in another student’s model</td>
<td>Game, Challenging, Competitive. Both parties attentive to beat each other.</td>
<td>Auditor embarrassment if no error is found?</td>
<td>[4]</td>
</tr>
<tr>
<td>4 Case Study e.g. of real-world models audited and found to be with errors.</td>
<td>Exposure to real-world model and possible mistakes.</td>
<td>Real-world models tend very complex and business domain specific.</td>
<td>[1], [6], [7]</td>
</tr>
<tr>
<td>5 Self-Audit</td>
<td>To grow self-awareness and reflection.</td>
<td>May become trivial so needs monitoring.</td>
<td>[4], [12]</td>
</tr>
<tr>
<td>6 Taxonomy of common errors.</td>
<td>Raise awareness of common errors. Student to add to it.</td>
<td>Problem of getting a meaningful taxonomy to start with.</td>
<td>[2], [5]</td>
</tr>
<tr>
<td>7 Spreadsheet engineering methodology</td>
<td>Give students a modelling process to follow.</td>
<td>Problem of getting a good methodology in the first place.</td>
<td>[2]</td>
</tr>
<tr>
<td>8 Version Control</td>
<td>Trail of who did what and when.</td>
<td>Hinders quick- and – dirty usage?</td>
<td></td>
</tr>
<tr>
<td>9 Confidentiality Controls e.g. spreadsheet encryption</td>
<td>Confidentiality of data may be a legal necessity.</td>
<td>Encryption can be complex to explain.</td>
<td>[8]</td>
</tr>
<tr>
<td>10 Auditing Tools (Integral)</td>
<td>Learners need to be aware of audit functions.</td>
<td>Such audit functions are trivial and give false sense of security.</td>
<td>[1]</td>
</tr>
<tr>
<td>11 Auditing Tools (External)</td>
<td>Learners need exposure to commercial audit tools.</td>
<td>Whose products to choose?</td>
<td>[1]</td>
</tr>
<tr>
<td>12 Access Control e.g. password mechanisms</td>
<td>Confidentiality of data may be a legal necessity</td>
<td>Hinders quick and easy use? None</td>
<td></td>
</tr>
<tr>
<td>13 Integral Documentation</td>
<td>Gives useful metadata about the model.</td>
<td>Slow so either not done or not kept current. None</td>
<td></td>
</tr>
<tr>
<td>14 Formulae length limitations</td>
<td>Long formulae are known to be a great source of error.</td>
<td>Splitting a formula may confuse the reader/user.</td>
<td>[10], [3]</td>
</tr>
<tr>
<td>15 Hard-coding of formulae controls e.g. guides to when hard-coding may be permitted</td>
<td>Hard coding of data known to be a great source of error when data needs to be changed.</td>
<td>Some constants need to be hard-coded?</td>
<td>[9], [3]</td>
</tr>
<tr>
<td>16 Named Ranges : use of</td>
<td>Known to have some advantages in clarifying structure.</td>
<td>? None</td>
<td></td>
</tr>
</tbody>
</table>
Establishing A Minimum Generic Skill Set For Risk Management Teaching In A Spreadsheet Training Course
Chadwick D.,

APPENDIX B: THE ACTUAL QUESTIONNAIRE : PAGE 1

The questionnaire pre-amble:

---

Establishing A Minimum Generic Skill Set For Risk Management In A Spreadsheet Training Course.

Much research by Raymond Panko [2] and others has shown that human error is at the root of most spreadsheet errors. Other researchers [1] have suggested that human errors may be reduced by teaching not just ‘how to do things correctly’ but also ‘how to avoid doing things incorrectly’.

I am researching what should be present in a ‘good’ training course that would help to reduce the known high frequency of human errors in spreadsheets. Such courses will be those provided by universities, private training organisations or company in-house. This questionnaire is a first attempt to identify those factors for the minimum set of training aims, methods and content that should appear in a good training course. Obviously there are generic and domain-dependent attributes for any training course. For instance we can, at this early stage, only identify those attributes that ALL courses should have regardless of the business area they may specifically be involved with.

This is a pilot – a first attempt at a questionnaire that will ultimately be sent to a wider audience. I am sending this to you as a possibly interested party and hope that you will participate by giving your views on what questions to ask, what factors to consider, and the weightings that should be given to different criteria.

If you would be interested in being more involved in this research please make contact. If we can establish some agreed approaches it may be possible for EuSpRIG to establish some good practice standards on training and perhaps, in time, accredit the courses of training providers meeting the agreed criteria.

Many thanks for your help with this.


Please return to David Chadwick on cd02@gre.ac.uk

Questionnaire follows on the next two pages.
Establishing A Minimum Generic Skill Set For Risk Management Teaching  
In A Spreadsheet Training Course  
Chadwick D.,

APPENDIX B: THE ACTUAL QUESTIONNAIRE: PAGE 2

PART 1 of 2 : Establishing Criteria For A Good Training Course
Please answer the following (circle your chosen answer):

<table>
<thead>
<tr>
<th>GENERIC TRAINING METHODS</th>
<th>Aim: every course should have at least one instance of the following practices to raise student's awareness of error situations and to develop self-reflective practices.</th>
<th>1=Not needed 2 3=Indifferent 4 5= Must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Error-seeded models</td>
<td>Student to find errors in tutor constructed model</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>2 Peer-audit (non-participative)</td>
<td>Student to find errors (if any) in another student’s model</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>3 Peer-audit (participative)</td>
<td>Student to find deliberately placed errors in another student’s model</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>4 Case Study</td>
<td>Student to discuss real-world models, and possible error situations</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>5 Self-Audit</td>
<td>Student checks own work, makes a statement as to how correct they think it is prior to tutor assessment</td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GENERIC TRAINING CONTENT</th>
<th>Aim: every student should be taught the following to give structure to the more domain-specific learning content</th>
<th>1=Not needed 2 3=Indifferent 4 5= Must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 Taxonomy of common errors.</td>
<td>Classification of common errors that they can add their own errors to over time</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>7 Spreadsheet engineering methodology</td>
<td>An stepped approach of some kind to aid a student in during spreadsheet building</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>8 Version Control approach</td>
<td>A structured approach to naming and storing past models</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>9 Confidentiality Controls</td>
<td>Spreadsheet encryption methods</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>10 Auditing Tools (Integral)</td>
<td>Built-in audit functions i.e. those integral to Excel</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>11 Auditing Tools (External)</td>
<td>Commercial audit tools and Excel add-ins e.g. OAK, SpACE</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>12 Access Control procedure</td>
<td>Password mechanisms etc</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>13 Integral Documentation approach</td>
<td>Documentation within the spreadsheet itself.</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>14 Formulae length limitations</td>
<td>Heuristics to limit formulae length e.g. no formulae should have more than 8 operators</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>15 Formulae Hard-coding controls</td>
<td>Guides to when hard-coding may be permitted and not permitted e.g. universal constants in physics?</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>16 Named Ranges</td>
<td>When to be used or not</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>17 Anything else?</td>
<td></td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>18 Anything else?</td>
<td></td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>19 Anything else?</td>
<td></td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>20 Anything else?</td>
<td></td>
<td>1 2 3 4 5</td>
</tr>
</tbody>
</table>
**APPENDIX B: THE ACTUAL QUESTIONNAIRE : PAGE 3**

**PART 2 of 2 : Some Information About You**

If you are a Training Provider
Please answer the following (circle your chosen answer):

<table>
<thead>
<tr>
<th>Please identify what kind of a trainer you are:</th>
<th>University</th>
<th>Private Trainer</th>
<th>Company In-house</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>If other please state:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What specific areas of modelling do you teach?</th>
<th>General</th>
<th>Accounting</th>
<th>Engineering</th>
<th>Medicine</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>If other please state:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>May I approach you again to discuss your answers?</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
</table>

If you are not a Training provider
Please answer the following (circle your chosen answer):

<table>
<thead>
<tr>
<th>What industry are you involved with?</th>
<th>Finance</th>
<th>Accounting</th>
<th>Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Medicine</td>
<td>General</td>
<td>Other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Do you think that spreadsheet training approaches should be improved?</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Please give your reasons: …</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The questionnaire is finished – again, many thanks for your help.

Please make any further comments below:
APPENDIX C: Theoretical Research Perspective

The theoretical research perspectives were a combination of both Positivist and Interpretative paradigms [11].

A Positivist approach was partly adopted for three reasons:

1. A deliberate policy was adopted of making the questionnaire straightforward to answer because the respondents (especially business managers) would have limited time.
2. A positivist approach produces data that is amenable to tabular presentation. This was important so that findings could be clearly presented to all the respondents.
3. The respondent parties would be business consultants, in-house business trainers, private trainers and academic educators. Most of these would be cognizant with the positivist approach (especially the business people whose support was dearly needed).

An element of Interpretative approach was also required for open sections of the questionnaire where respondent suggestions were required. This was problematic in that interpretation was open to subjectivity but was necessary as it was possible that not all pertinent factors may have been covered in the initial survey questions (see paragraph 2.3 ‘Scoring The Questionnaire’).
# APPENDIX D: ANALYSIS OF QUESTIONNAIRE RESULTS

<table>
<thead>
<tr>
<th>Survey Question Number</th>
<th>AT: Academic Trainer</th>
<th>BC: Business Commercial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AT</td>
<td>AT</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>76</td>
<td>77</td>
</tr>
<tr>
<td>Out of</td>
<td>80</td>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Survey Question Number</th>
<th>Total Likert Score</th>
<th>Survey Question Number</th>
<th>Total Likert Score</th>
<th>Rank Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29</td>
<td>13</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>1</td>
<td>29</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>4</td>
<td>29</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>10</td>
<td>29</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>26</td>
<td>12</td>
<td>29</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>2</td>
<td>28</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>27</td>
<td>15</td>
<td>28</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>27</td>
<td>7</td>
<td>27</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>8</td>
<td>27</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>29</td>
<td>16</td>
<td>27</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>24</td>
<td>5</td>
<td>26</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>29</td>
<td>14</td>
<td>26</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>30</td>
<td>9</td>
<td>25</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>26</td>
<td>3</td>
<td>24</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>28</td>
<td>11</td>
<td>24</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>27</td>
<td>6</td>
<td>21</td>
<td>16</td>
</tr>
</tbody>
</table>
### APPENDIX E: SAMPLE OF FREE TEXT COMMENTS

The issues raised in the questionnaire are all important. However, the crucial issue is, how much time one has in a course. Students have to understand that:
- neat layout does not guarantee correctness, but poor layout is a good hint for incorrectness;
- certain working conditions will trigger faulty results
- every thing produced needs to be checked (hence, they should learn about techniques for checking spreadsheets).

Context of spreadsheet based decision-making – Why are they important? What might impact of correct (or incorrect) spreadsheet models be? – Vital to tell them WHY this stuff is important, easy for us old lags to forget and assume we’re preaching to the choir when we’re not.

Just because a syllabus supplied by a training organization contains an item does not mean that the individual trainer on the day effectively teaches it, or that the student learns it. A topic can appear on a course to get accreditation, but never be examined on; or if it appears on the test, is a question that is always skipped. Is it possible to establish criteria that allow us to examine actual training results to verify that these standards have been learnt?
Abstract

Human error research on overconfidence supports the benefits of early visibility of defects and disciplined development. If risk to the enterprise is to be reduced, individuals need to become aware of the reality of the quality of their work. Several cycles of inspection and defect removal are inevitable. Software Quality Management measurements of defect density and removal efficiency are applicable. Research of actual spreadsheet error rates shows data consistent with other software depending on the extent to which the work product was reviewed before inspection. The paper argues that the payback for an investment in early review time is justified by the saving in project delay and expensive errors in use.

'If debugging is the process of removing bugs, then programming must be the process of putting them in' - Anon.

Initial questions (answers later)

1) Are defects good or bad?
2) If your reviewer tells that they found no defects in a spreadsheet, is that good or bad?

Introductory definitions

To err is human; people make errors. In his book Human Error [3], James Reason uses 'error' as a generic term that can be categorised as slips in execution (eg typos) or mistakes in judgment (eg using the wrong word). In programming, human error creates defects (aka faults) in the software which can be immediately visible or latent. Depending on the conditions in production use, some defects may give rise to possibly many incidents of failure, and some may never be encountered. These incidents are reported by the user as fault reports, often called bug reports.

Defect Density is a standard quality measure of the number of defects per thousand lines of code (KLOC) or Function Points (FP) or, in the spreadsheet domain, Unique Formulas (UF). The defect density of a product follows a cumulative curve which rises at the start as defects are created and falls later as they are found and removed.

Injected Defects are defects put into the product usually due to mistakes which people make; but sometimes software is intentionally seeded with known
defects in order to measure the efficiency of the review process. The Defect Injection Rate (DIR) is the percentage of work that is defective. You never know how many injected defects you have in your product – you can only estimate them from previous experience or benchmarks.

Removed Defects are defects which are identified by processes such as design review, code inspection, execution test, or (most expensive of all) user experience. They are also usually classified by severity in order to prioritise the fixing effort and then removed by rework. The Defect Removal Efficiency (DRE) is the percentage of defects detected by the find and fix processes.

The number of defects detected/fixed per unit of time (or effort) is useful for estimating when a product is ready for release.

**Reported Error Rates in Large Spreadsheets**

Robert Lawrence [1] reported these statistics on the thirty most financially significant projects that Mercer Finance & Risk Consulting reviewed year-ending June 2004:

- Average 2,182 unique formulae per model
- Average 151 issues raised during the initial review
- Average six revisions to produce a model that could be signed-off
- One spreadsheet needed 17 revisions to resolve 239 issues
- Average 7% defect injection rate, 75% defect removal efficiency

Figure 1 shows that the defect rate for a model can be estimated as 62 plus 0.41 times the number of unique formulas (UF); or, 2.6 times the square root of UF.
Figure 1: Scatterchart of issues vs no. UF for 30 models reviewed by Mercer.

7% is low by most end-user standards; according to Panko [4] and repeated by many other studies, 20% is a more normal end-user defect rate. It would be reasonable to infer that those developing the models had already done their own checking and cleanup before handing these models over for external audit. Later, we argue that defect removal can be made more efficient by reviewing earlier in the process, rather than waiting to review the final product to find them. Table 1 shows how many revisions one would expect for a given rate of defect injection and removal.

<table>
<thead>
<tr>
<th>D.I.R.%</th>
<th>3%</th>
<th>4%</th>
<th>5%</th>
<th>7%</th>
<th>10%</th>
<th>15%</th>
<th>20%</th>
<th>30%</th>
</tr>
</thead>
<tbody>
<tr>
<td>20%</td>
<td>15</td>
<td>17</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>29</td>
<td>37</td>
</tr>
<tr>
<td>25%</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>23</td>
<td>30</td>
</tr>
<tr>
<td>30%</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>15</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>26</td>
</tr>
<tr>
<td>35%</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>23</td>
</tr>
<tr>
<td>40%</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>50%</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>60%</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>80%</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>100%</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 1: Estimated no. of revisions for 2000 Units of work (Unique Formulas)
For given Defect Removal Efficiency and Defect Injection Rates

Even at only 3% injection rate and 100% removal efficiency you still need **three** revisions:

1 to write the initial model and inject 30 defects;
2 to remove the 30 defects by making 30 changes which causes 1 more defect;
3 to remove the final(?) defect.

Surveys of end-user development that review the products at an earlier stage find typically 20% of defects. Informal inspection efficiency is around 50%; it takes a formal process to reach 75%. For an average size model of 2000 UFs, 20% DIR and 50% DRE, one would expect 12 revisions to reach a sign-off point.

We argue that the payback for this investment in early review time is justified by the saving in project delay and expensive errors in use. So, how can people be encouraged to review their work earlier? We believe that this can be done by helping them to become aware of the real quality of their work.
Overconfidence

Justin Kruger and David Dunning of Cornell University reported [2] that

'People tend to hold overly favorable views of their abilities … their incompetence robs them of the metacognitive ability to realize it. Paradoxically, improving the skills of participants, and thus increasing their metacognitive competence, helped them recognize the limitations of their abilities.'

'Because people usually choose what they think is the most reasonable and optimal option … the failure to recognize that one has performed poorly will instead leave one to assume that one has performed well.'

'Sullivan, in 1953, marveled at "the failure of learning which has left their capacity for fantastic, self-centered delusions so utterly unaffected by a life-long history of educative events".'

Ray Panko [4] has presented analysis to Eusprig on the rates of errors.

'Consistent with human error research, which has seen similar error rates across cognitive tasks of comparable difficulty, spreadsheet error rates are very similar to those in traditional programming. However, while programmers spend 25% to 60% of their development time in testing, testing among spreadsheet developers in industry is extremely rare.'

Making work quality visible

The key, then, is to work in a way that faces the facts of defect injection, that rewards the removal of defects rather than punishes the necessary precondition of discovery. The management of developers is not to 'hold their feet to the fire' but to make the facts inescapably obvious. It is easy to brush aside and forget errors that have been fixed or can be quickly fixed. A written record helps avoid overconfidence and provides a basis for post-hoc analysis that can point to process improvements that can be made. James Reason [3] advises self-knowledge as the first line of defence against error. His research in the medical world has found that the best consultants are those who consistently analyze their work and learn from their mistakes, rather than the characteristic arrogant denial that 'we don't make mistakes here'. To assist individuals to become conscious of their own quality, some minimal schemes have been devised to collect and analyse data. One of these which can be paper-based or supported by (surprise!) a spreadsheet is the Personal Software Process (PSP) described below.
Early visibility

The conventional software development lifecycle (SDLC) is described in phases of Requirements Analysis, System Design, Software Development, and Testing. Where reviews take place during this lifecycle, project managers can use early defect data to estimate the 'troublesomeness' of the product being developed, and get an indication of when it is safe to release.

Given that you collect defect arrival data, and the curve has achieved its maximum at time $t_m$ (e.g., the inflection point), you can calculate, assuming a Rayleigh distribution [11], the likely final number of defects and when they should be removed. The simplest initial assumption is that ~40% of the total defects have appeared by $t_m$.

The identification of separate activities with skills appropriate to each fits the specialised world of the professional software developer constructing projects for delivery to users. For end-user development, the fact that the customer is also the developer abbreviates the early stages as the spreadsheet creator believes that they understand their own requirements. They are often also their own tester. However, when they are building spreadsheets for use by others, they are already in a development role and the enterprise should equally consider the value of a separate testing role.

For applications that are critical to the success of the business, it is already recognised that there is a benefit from separating these stages and consciously documenting requirements, designing a layout and a data flow, and an independent review from a second pair of eyes. Panko [4] has related the types of testing appropriate to the stage of development.

At Eusprig 2006, Kumiega and Van Vliet of the Illinois Institute of Technology [5] described a spiral development methodology for rapid prototyping in financial markets. By focusing on return on investment, they increase the amount of amount of review and testing that a spreadsheet trading model gets in line with the value at risk in using the model.

Disciplined Development


1. Time and Defect recording
2. Software Size measurement
3. Software Size estimation
4. Statistically based estimation using Proxies
5. Time estimation and project scheduling
6. Process management
7. Design and Code reviews
8. Quality management through defect reduction
Paradoxes of measurement

Normal Fenton [9] reported on a conversation with the Robert Grady of HP about a system with no reported field defects. Initially, it was thought to be an example of ‘zero defects’. They later discovered that was because no one was using it.

Capers Jones [10] pointed out in 1991 that there are two general rules for customer-reported defects:

- The number of defects found correlates directly with the number of users
- The number of defects found correlates inversely with the number of defects that exist

This is because if the software has many users, it will have more execution time, and hence, more defects will be uncovered more quickly. Conversely, if the software is buggy, people will not use it, and fewer defects will be uncovered!

Answers

1) Are defects good or bad?

This question is using emotionally loaded words. If defects are 'bad', then those who create them are 'bad' and those who report them are messengers of 'bad' news. People will hide defects or disclaim them or divert responsibility for them. Defects exist. Would you rather not know they are there?

2) If your tester tells that they found no defects in a piece of software, is that good or bad?

It depends on what else you know from previous experience of the software developer's quality record, the efficiency of the tester, and any previous history of problems with the application under test.

Conclusion

If risk to the enterprise is to be reduced, individuals need to become aware of the reality of the quality of their work. We recommend that where there is no independent review, individuals should adopt that role and consciously examine their work and record issues for correction as if they were doing it for
someone else[12]. Such a record will serve as an inescapable reminder of the actual difficulty and quality of the project they are working on, and make it less likely that the product will be released in ignorance of the facts about its state. Based on previous research [8] we propose that in order to inculcate such a discipline research experiments are required towards a practical implementation of a Personal Spreadsheet Process.

References

1. Financial Modelling of Project Financing Transactions, Robert J Lawrence, Institute of Actuaries of Australia Financial Services Forum

The objective of this conference is 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 in spreadsheet models.

The papers cover a broad spectrum of practical experience and research.