Risk Reduction in End User Computing
Best practice for spreadsheet users in the new Europe

15-16 July 2004 University of Klagenfurt, Austria

Hosted by:

UNIVERSITÄT KLAGENFURT

2004
European Spreadsheet Risks Interest Group

EuSpRIG 2004 Conference

"Risk Reduction in End User Computing - Best practice for spreadsheet users in the new Europe"

Editors:
Patrick Cleary
David Ward

Conference Hosted By:
Prof. Roland Mittermeir,
University of Klagenfurt, Austria

Conference Sponsored By:
Frontline Systems UK Ltd
Austrian Computer Society
Austrian Society for Organisation
European Computer Driving Licence Foundation
Information Systems Audit & Control Association
(Northern England Chapter)
KPMG LLP

July 2004
University of Klagenfurt, Austria
Proceedings of EuSpRIG 2004 Conference
"Risk Reduction in End User Computing -
Best practice for spreadsheet users in the new Europe
First published 2004
Printed in United Kingdom by KPMG

Held at University of Klagenfurt; 15 to 16 July 2004

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

ISBN : 1 902724 94 1

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
in collaboration with
Prof. Roland Mittermeir, University of Klagenfurt, Austria

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 of KPMG with photographs donated by University of Klagenfurt, Austria
EuSpRIG 2004 Conference

"Building Better Business Spreadsheets - from the ad-hoc to the quality-engineered"

Organising Committee

Chair: Prof. Roland Mittermeir, University of Klagenfurt, Austria
Patrick Cleary, University of Wales Institute Cardiff, UK
Graham MacDonald, HM Customs & Excise, UK

Programme Committee

Chair: Patrick Cleary, University of Wales Institute Cardiff, UK
Prof. Roland Mittermeir, University of Klagenfurt, Austria
Patrick O'Beirne, Systems Modelling Ltd, Ireland
Ray Butler, Highways Agency, UK
Markus Clermont, University of Limerick, Ireland
David Colver, Operis, UK
Jocelyn Paine, Bristol University, UK
Barry Pettifor, PriceWaterhouseCoopers, UK
Leon Strous, De Nederlandsche Bank, Netherlands

EuSpRIG Management Committee

EuSpRIG Chair : Patrick O’Beirne
Chair of Conference Organizing Committee : Prof. Roland Mittermeir
EuSpRIG Secretary and Chair of Conference Programme Committee : Patrick Cleary
EuSpRIG Membership Secretary : Grenville Croll
EuSpRIG Treasurer : Graham MacDonald
Member : Ray Butler
Member and past Chairman : David Chadwick :
Member : Markus Clermont
Member : Jocelyn Paine
Member : Barry Pettifor
Member : Leon Strous
Member : David Ward
# Table of Contents

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table of Contents</strong></td>
<td>v</td>
</tr>
<tr>
<td><strong>Preface</strong></td>
<td>vi</td>
</tr>
<tr>
<td>Why Banks use Spreadsheets</td>
<td>1</td>
</tr>
<tr>
<td>Dean Buckner - Financial Services Authority (UK)</td>
<td></td>
</tr>
<tr>
<td><strong>ECDL Spreadsheets Best Practice Module</strong></td>
<td>15</td>
</tr>
<tr>
<td>Gary Cleere - European Computer Driving Licence</td>
<td></td>
</tr>
<tr>
<td><strong>A Paradigm for Spreadsheet Engineering Methodologies</strong></td>
<td>23</td>
</tr>
<tr>
<td>Thomas A. Grossman, University of San Francisco, Özgür Özüik SFSU San Francisco</td>
<td></td>
</tr>
<tr>
<td><strong>XIstruct: A Tool for Building Structured Error-Resistant Spreadsheets</strong></td>
<td>35</td>
</tr>
<tr>
<td>Gary K. Arakaki</td>
<td></td>
</tr>
<tr>
<td><strong>TellTable Spreadsheet Audit:</strong> from technical possibility to operating prototype</td>
<td>45</td>
</tr>
<tr>
<td>John Nash, Andy Adler and Neil Smith, University of Ottawa</td>
<td></td>
</tr>
<tr>
<td><strong>Identification of logical errors through Monte Carlo simulation</strong></td>
<td>57</td>
</tr>
<tr>
<td>Hilary L. Emmett, Decisioneering (UK) Ltd, and Lawrence I. Goldman, Decisioneering Inc</td>
<td></td>
</tr>
<tr>
<td><strong>End User Computer Applications - Auditability and Other Benefits Derived from a Temporal Dimension</strong></td>
<td>67</td>
</tr>
<tr>
<td>Ralph Baxter, ClusterSeven Ltd</td>
<td></td>
</tr>
<tr>
<td><strong>A novel approach to formulae production and overconfidence measurement to reduce risk in spreadsheet modelling</strong></td>
<td>71</td>
</tr>
<tr>
<td>S. Thorne, D. Ball and Z. Lawson, University of Wales Institute Cardiff</td>
<td></td>
</tr>
<tr>
<td><strong>Complexity Metrics for Spreadsheet Models</strong></td>
<td>85</td>
</tr>
<tr>
<td>Andrej Bregar, University of Maribor</td>
<td></td>
</tr>
<tr>
<td><strong>A Toolkit for Scalable Spreadsheet Visualization</strong></td>
<td>95</td>
</tr>
<tr>
<td>Markus Clermont, University of Limerick</td>
<td></td>
</tr>
<tr>
<td><strong>Using Layout Information for Spreadsheet Visualization</strong></td>
<td>107</td>
</tr>
<tr>
<td>Sabine Hipfl, Universität Klagenfurt</td>
<td></td>
</tr>
<tr>
<td><strong>Spreadsheet Structure Discovery with Logic Programming</strong></td>
<td>121</td>
</tr>
<tr>
<td>Jocelyn Paine</td>
<td></td>
</tr>
<tr>
<td><strong>Spreadsheet good practice: is there any such thing?</strong></td>
<td>135</td>
</tr>
<tr>
<td>David Colver, Operis</td>
<td></td>
</tr>
<tr>
<td><strong>When, why and how to test spreadsheets</strong></td>
<td>145</td>
</tr>
<tr>
<td>Louise Pryor</td>
<td></td>
</tr>
<tr>
<td><strong>Computational Models of Spreadsheet-Development Basis for Educational Approaches</strong></td>
<td>153</td>
</tr>
<tr>
<td>Karin Hodnigg and Roland T. Mittermeir, Universität Klagenfurt, and Markus Clermont, University of Limerick</td>
<td></td>
</tr>
</tbody>
</table>
Preface

2. WELCOME

I am delighted to welcome you to our biggest conference yet!

Your presence is important to us and to the wider community of risk management professionals:

- as a delegate, you can bring back what you will learn and help raise awareness and improve standards;
- as a speaker, your expertise and contribution is deeply appreciated and will enrich those who hear it.

3. OUR DEVELOPMENT

In five years, EuSpRIG has grown in strength to not just a European, but a worldwide network of academics and practitioners. Our contributors this year are from Austria, Canada, Ireland, Japan, Slovenia, the UK, and USA. In previous years we had papers from Australia and New Zealand.

Our mission is to increase the awareness of spreadsheet risk and promote methods of assessing risk, detecting errors, and improving the productivity and quality of spreadsheet development.

The most significant step forward in the past year has been our joint work with the European Computer Driving Licence Foundation (ECDLF) in the drawing up of a syllabus of good spreadsheet practice, which will form the foundation of a new certification for end user competence. This will be welcomed by all stakeholders who are concerned about exposure to risk from uncontrolled spreadsheet development.

4. STANDARDS

Our review process involves three referees who provide feedback comments to authors. We apply strict standards of references and evidence to the academic papers, and appropriate standards of relevance and usefulness to the submissions from practitioners. This year, we received more papers than we could cope with, even after extending the conference from the previous format of two half-days to a day and a half. We were sorry to have to disappoint the authors of some good work, but the competition was strong.
5. THANKS

Special thanks are due to the committee who have given voluntarily of their time, expertise, and organisational resources, to bring this conference to its present success:

- Pat Cleary, who took on the roles both of secretary and Programme Committee chair and ever-present chivvying;
- Roland Mittermeir of the University of Klagenfurt, who with his staff has organised our 2004 conference in the very heart of the new EU, on top of his academic programme and review role;
- Graham MacDonald, our treasurer who set up our euro processing facilities this year and keeps a keen eye on correct books;
- David Ward of KPMG for preparing the proceedings;
- Barry Pettifer of PwC who along with David Ward of KPMG provided very valuable sponsorship of our teleconferences;
- Grenville Croll of Frontline systems UK who not only managed our membership database, but secured prime sponsorship of this year's conference;
- Ray Butler for the support of ISACA Northern England for EuSpRIG, their sponsorship of the Student Prize, and his own sage advice as a past chair;
- David Chadwick, our 2003 Chair and a founder member who brought EuSpRIG to the prime position it holds today; regretfully his health did not permit his attendance this year, but our appreciation of his academic rigour is shown in the naming of the Student Prize;
- Leon Strous who performed the very important task of processing credit card payments for the conference;
- and all members of the EuSpRIG committee who gave of their time and thoughtful advice in our monthly teleconference calls: Markus Clermont, Barry Phillips, David Colver, and Jocelyn Paine - and that was on top of their speaking commitments to this conference.

Enjoy the conference, contribute to the discussions, benefit from the networking, and keep in touch!

Patrick O'Beirne, Chairman 2004, EuSpRIG
Systems Modelling Ltd., Ireland, July 2004
Why banks use spreadsheets

Dean Buckner
Financial Services Authority

July 2004

Outline

- Distinguish business critical from non-business critical applications
- How banks typically use spreadsheets
- Two case studies
- The cost of controls
Business Critical Applications

Non business critical applications

• Spreadsheets used to solve some (typically numerical) problem
  – Single user
  – Used once only
  – Typically thrown away after use
  – Typically a model of some kind
  – User knows the "right" answer
Example: The mother of all spreadsheets
- BASEL II model
  - QIS 3
- Took 6 months to develop
  - Team of people
- 2 Gb
  - Needed dedicated server!
- Still "disposable"

Business Critical Applications
- Any system whose function is critical to running of business
  - More than one user
    - Probably doesn't know "right answer"
  - Performing cyclical task
  - Sometimes model
  - But usually data processing of some kind
    - Aggregation
    - Enrichment
    - Checking
    - MIS
Business Critical Applications

- Financial regulators are worried about business critical applications
  - In banking businesses, they are critical to the stability of financial markets, and the fight against financial crime
- Financial regulators are not so worried about non-business critical applications
  - These are not, by definition, critical systems

Examples

- Aggregation and MIS
  - “Currently, regulatory reporting … is heavily reliant on spreadsheets. Whilst amendments to systems necessary for compliance with BASEL II … will significantly enhance our ability to automatically feed returns we cannot envisage that manual intervention won’t be necessary.”
Data manipulation

- Pivoting
  - Column-row format changes
- Mapping
  - E.g. change "DEM" to "EUR"
- Consolidating
  - E.g. "short 100,000" to "-100000.00"
- Enrichment
  - User types or copies

Calculation

- Creation of secondary or "derived" data set from primary set
  - Very often involves matrix multiplication on large data sets
  - 50Mb spreadsheets
  - Common use of replicated formulae
Spreadsheets as GUI

- In my experience, the most common form of spreadsheet application
- Spreadsheet is used as interface to
  - Standard, robust database back end
  - Applications library in C++ with standard change control
  - Spreadsheet calls functions, reads and writes data to
  - Allows user to interact with powerful system in way that conventional GUI’s don’t allow

Case study

- In reaction to the rising trend of spreadsheet trades, [firm] has made its primary goal for [project] in 2004 to implement the new [...] systems for the principal volume trading areas. Even with these roll-outs taking place, they still forecast the number of trades on spreadsheets may be higher by the end of this year compared to 2003’s year-end figure, given that overall business volumes are expected to continue to grow. However, [firm] assure us that the derivatives operating environment continues to be adequate for current and envisaged trading levels, given other mitigating controls that are place, such as a regular programme of structured trade reviews, undertaken by product control.
Regulatory concern

- FSA to receive quarterly sign-off from European CEO on the adequacy of the operating environment for [firm's business]. In addition, we will also ask [firm's] senior management to provide their assessment of the progress they expect [project name] to have achieved by the end of this year in reducing operational risks caused by the use of spreadsheets as sources for risk/valuation data.
- ...we will ask for them to provide to us year-end forecasts for certain operational metrics to enable us to more accurately quantify the progress being made.
- We will take the decision on what form this review should take, [use of internal audit or a Section 166 report], following update Q3 2004.

Replacement strategy

- Firm’s primary goal for [project] in 2004 is implementing [Sys1] and [Sys2] for the principal volume trading areas
- VERY expensive project
- However, even with these roll-outs taking place, they still forecast the number of trades on spreadsheets to be higher by the end of this year compared to 2003's year-end figure.
FSA metrics

- number of high-risk spreadsheets;
- number of external trades valued on raw spreadsheets;
- number of spreadsheets over 40 Mb.

Good news, bad news

- Good news
  - The total number of spreadsheets remained broadly flat over last year.

- Bad news
  - the number of trades valued and risk managed on spreadsheets rose throughout 2003 and now stands in the tens of thousands.
Case study 2

- Firm uses interconnected spreadsheet system to integrate its Value at Risk system
- Fully developed end-user development policy
- No intention to replace

Costs and benefits

- Panko’s “iron law”
  - All systems and controls have a cost
  - No error-checking for free
- But spreadsheet systems are much cheaper than single systems solution
- Question
  - Is cost difference explained by the cost of systems and controls?
The cost of a system

• System cost = cost of controls + cost of functionality + slippage cost

• Where
  – cost of controls is cost associated with error trapping, code reading, change control,
  – Cost of functionality is cost of “doing the business”
  – Slippage cost is billable hours “wasted”

• Question: do large systems have high slippage costs that explain the discrepancy with spreadsheet solutions?

Single system projects

• Can be wasteful because
  – They require large infrastructure
  – Many chiefs, few Indians
  – Find it difficult to interact with business users
  – Find it difficult to understand business requirements
User-developed systems

- Can be efficient because
  - Users understand exactly their own requirements
  - Have incentive to keep costs low
  - Do not need project infrastructure, hierarchy &c

However

- There is also strong evidence that user developed systems have few of the standard controls associated with "proper" systems
- FSA "fit for purpose" test
Conclusion

- Little research on code production efficiency
  - Software theft example
- At a guess, non-user developed systems are too expensive
  - Because of slippage and waste
- User developed systems are too cheap
  - Because users no real incentive to impose controls

And finally …

- =+$S:\DTM\Ps\Subjects\RISK\Retail\mortgage lending statistics\Mortgage stats\[Summary Q4 2002.xls]Summary'!SD$52+$S:\DTM\Ps\Subjects\RISK\Retail\mortgage lending statistics\Mortgage stats\[Summary Q4 2002.xls]Summary'!SD$51+$S:\DTM\Ps\Subjects\RISK\Retail\mortgage lending statistics\Mortgage stats\[Summary Q4 2002.xls]Summary'!SD$50+$S:\DTM\Ps\Subjects\RISK\Retail\mortgage lending statistics\Mortgage stats\[Summary Q4 2002.xls]Summary'!SD$49+$S:\DTM\Ps\Subjects\RISK\Retail\mortgage lending statistics\Mortgage stats\[Summary Q4 2002.xls]Summary'!SF$49
Questions & Comments
ECDL Spreadsheets Best Practice Module

Gary Cleere

(Context and Rationale)

1. INTRODUCTION

The European Computer Driving Licence (ECDL) is the world’s leading computer skills certification programme. The ECDL programme now brings together more than 4 million participants worldwide.

The expertise established in the ECDL Foundation and throughout the ECDL community is now being harnessed to develop further ECDL programmes promoting practical IT skills. The ECDL Advanced Programme has been launched and in excess of 50,000 Candidates are already participating in this programme. Further to this a new product ECDL Spreadsheets Best Practice is in development. The product aims to assure good approaches to spreadsheet design use and to help individuals and organisations alike recognise some of the issues and considerations at stake. The same rigorous validation process is applied to all ECDL programmes.

Following on from its particular experience in the end-user spreadsheets space, with the ECDL core level, Module 4, Spreadsheets, certification product and the ECDL Advanced offering, ECDL AM4, the ECDL Foundation has recently undertaken development work with Subject Matter Experts (SME’s) from EUSPRIG, (The European Spreadsheets Risks Interest Group), to develop an ECDL certification module for best practice in the use of spreadsheets. The working title for the product is ECDL Spreadsheets Best Practice, ECDL Module 4SSBP.

2. THE ECDL AND ITS MISSION

To provide some background about ECDL. The ECDL is an internationally recognised and accepted qualification that enables people to verify their competence in core computer skills and Information Technology (IT) knowledge.

The ECDL consists of seven modules: Module 1 is a theoretical test of computing knowledge at a general level and Modules 2-7 are practical skills tests.

Module 1: Concepts of Information Technology (IT)

Knowledge of main concepts of IT at a general level. Make up of a personal computer in terms of hardware, software, storage & memory. General knowledge of communications, networking, software applications, health & safety issues, security, protection and legal issues.

Module 2: Using the Computer and Managing files

Ability to use a computer and its operating system. The skills to adjust settings, use help, restart from failure, operate in desktop environment, organise files and folders. Use text editing, print management, compression and virus protection software.

Module 3: Word Processing

Ability to use a word processing application for every day tasks creating, formatting and finishing small sized documents. Copying & moving, using tables, pictures & images and mail merge tools.
Module 4: Spreadsheets

Ability to use a spreadsheet application. Developing, formatting, modifying a spreadsheet of limited scope understanding the concepts. Generating and applying formulas and using graphs/charts.

Module 5: Database

Ability to use a database application and understand the main concepts. Create & modify tables, queries, forms & reports and prepare outputs. Relate tables, manipulate and retrieve information using query and sort tools.

Module 6: Presentation

Ability to use presentation tools on a computer. Create, format, modify and prepare slide layouts for presentation. Ability to create simple drawn objects and create charts. Manipulate text, pictures and images and animate a presentation.

Module 7: Information and Communication

This module is divided into two parts. The first part covers the ability to use the Internet, understand the basic concepts and appreciate security issues. Use search engine tools for common browsing and information retrieval tasks. Completing a web-based form, using bookmarks and printing pages. The second part covers the ability to use e-mail and understand the basic concepts including security issues. Create, send, receive, reply to & forward e-mail, handle attachments and organise message folders in e-mail.

The ECDL is for everyone. It is an inclusive and recognised standard. It certifies that the holder has knowledge of the essential concepts of Information Technology and is able to use a personal computer and common computer applications at a recognised level of competence both at work and at home.

The ECDL is based on a single agreed Syllabus. The Syllabus lists the facts to be known and the skills that must be mastered for a candidate to achieve ECDL certification. The ECDL Syllabus has been continually refined by experts and practicing professionals since the launch of the ECDL programme in 1997.

The ECDL examination consists of seven tests that are designed to assess the knowledge and skills needed to use a personal computer and common computer applications. In each test, competencies are assessed in a straightforward manner, with ample time allowed for completion of the test.

The ECDL mission and ethos have been clearly stated. The ECDL mission is to:

- Help raise the general level of computer skills in society.
- Provide an essential qualification that allows all people to participate in e-Society.
- Establish a global benchmark for core computer skills competency and IT knowledge.
- The ethos of ECDL is that the programme is inclusive. ‘Open to everyone’.

This expression of the mission of the ECDL certification programme is central to the quality and validity of the programme.
3. ECDL SPREADSHEETS BEST PRACTICE:

ECDL Module 4, Spreadsheets Best Practice, requires the Candidate to appreciate their responsibility for good spreadsheet design and use given the scope and significance of spreadsheets use in the contemporary workplace. The Candidate shall appreciate the need for good spreadsheets specification, be aware of the key security considerations, and appreciate the benefits of clearly organised, well presented and easy-to-use spreadsheets.

The Candidate shall be able to construct sound spreadsheets, find and correct common errors, and test for input and output accuracy. It is anticipated that the candidate will already have mastered the skills and achieved the knowledge detailed in ECDL Module 4, Spreadsheets, or ECDL Module AM4, Spreadsheets, Advanced-Level8.

4. DEVELOPMENT PLANS AND PILOTING

The ECDL Foundation has worked closely with EUSPRIG Subject Matter Experts (SME’s), Patrick O’Beirne, David Chadwicke, Pat Cleary, Grenville Croll and Loise Pryor to develop this module over the last number of months and is keen now to begin piloting the module for use by individuals and organisations alike.

Plans are to pilot the new module from September 2004. As part of its participation at the EUSPRIG Klagenfurt event in Austria, the ECDL-F are also keen to gain inputs and impressions about the product from the EUSPRIG community at large. A test base for the product is currently in development and some items will be illustrated at the Klagenfurt event.

As part of its participation at the EUSPRIG Klagenfurt event the ECDL-F is keen to invite comment and feedback from the EUSPRIG community in respect of the Syllabus defined a this stage. To this end, the draft Syllabus is attached as an appendix herewith in section 5. Comments can be provided in person to Garry Cleere at the Klagenfurt event or by e-mail to the following address, 
garry.cleere@ecd.com.
European Computer Driving Licence

Spreadsheets Best Practice
Module 4SSBP Syllabus Version 1.0

Copyright © 2004 The European Computer Driving Licence Foundation Ltd. All rights reserved. No part of this publication may be reproduced in any form except as permitted by the ECDL Foundation. Enquiries for permission to reproduce material should be directed to the publisher.

The European Computer Driving Licence Foundation Ltd
Third Floor,
Portview House
Thorncastle Street
Dublin 4
Ireland
Tel: + 353 1 630 6000
Fax: + 353 1 630 6001
E-mail: info@ecdli.com
URL: http://www.ecdl.com

Module 4SSBP - Spreadsheets Best Practice

The following is the Syllabus for Module 4SSBP, Spreadsheets Best Practice, which provides the basis for the theory and practice-based test in this module domain.
Module Goals

Module 4SSBP, Spreadsheets Best Practice, requires the Candidate to appreciate their responsibility for good spreadsheet design and use given the scope and significance of spreadsheets use in the contemporary workplace. The Candidate shall appreciate the need for good spreadsheets specification, be aware of the key security considerations, and appreciate the benefits of clearly organised, well presented and easy-to-use spreadsheets. The Candidate shall be able to construct sound spreadsheets, find and correct common errors, and test for input and output accuracy. It is anticipated that the Candidate will already have mastered the skills and achieved the knowledge detailed in ECDL Module 4, Spreadsheets, or ECDL Module AM4, Spreadsheets, Advanced-Level.

<table>
<thead>
<tr>
<th>Category</th>
<th>Skill Set</th>
<th>Ref.</th>
<th>Task Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.1 Design</td>
<td>S.1.1 Specification</td>
<td>S.1.1.1</td>
<td>Define spreadsheet specifications; purpose, user requirements, usability features, author, version history.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.1.1.2</td>
<td>Use cell comments, descriptions in the spreadsheet to list source(s) of input data, assumptions made, user access level.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.1.1.3</td>
<td>Make conventions explicit in the spreadsheet: calculation methods, functions, formats, policies.</td>
</tr>
<tr>
<td></td>
<td>S.1.2 Security</td>
<td>S.1.2.1</td>
<td>Make regular secure backups of spreadsheet and related files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.1.2.2</td>
<td>Maintain separately saved versions of spreadsheets in development.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.1.2.3</td>
<td>Understand the security limitations of password protection on worksheets, spreadsheet files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.1.2.4</td>
<td>Use passwords of at least 8 mixed case, non-alphanumeric characters.</td>
</tr>
<tr>
<td>S.2 Input</td>
<td>S.2.1 Set-up</td>
<td>S.2.1.1</td>
<td>Simplify long formulas by referring to separate calculations in related cells and by using named ranges.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.2.1.2</td>
<td>Isolate constants such as: exchange rate, tax rate, in a worksheet.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.2.1.3</td>
<td>Identify and correct units of measure, conversion calculations in a worksheet.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.2.1.4</td>
<td>Understand and appreciate the effects of the “precision as displayed (shown)” setting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.2.1.5</td>
<td>Understand and use manual, automatic calculation.</td>
</tr>
<tr>
<td>S.3 Calculation</td>
<td>S.3.1 Fundamentals</td>
<td>S.3.1.1</td>
<td>Understand order of precedence of mathematical operators.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.3.1.2</td>
<td>Remove circular references.</td>
</tr>
<tr>
<td>Category</td>
<td>Skill Set</td>
<td>Ref.</td>
<td>Task Item</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>S.3 Error</td>
<td>S.3.2 Error Identification</td>
<td>S.3.2.1</td>
<td>Identify missing input values using the precedent tool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.3.2.2</td>
<td>Identify cells with missing dependents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.3.2.3</td>
<td>Use information functions: ISERROR, ISNA.</td>
</tr>
<tr>
<td></td>
<td>S.3.3 Error Correction</td>
<td>S.3.3.1</td>
<td>Correct relative, absolute and mixed cell references in cell ranges.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.3.3.2</td>
<td>Correct #VALUE!, #NAME?, #N/A, #REF, #NUM error values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.3.3.3</td>
<td>Correct inconsistencies in a pattern of formulas.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.3.3.4</td>
<td>Correct mistakes in totals caused by inserting, deleting rows and columns.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.3.3.5</td>
<td>Correct grand totals that double-count subtotals.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.3.3.6</td>
<td>Correct mismatched cross-tot checks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.3.3.7</td>
<td>Correct mistakes created by incorrect use of automatic sum feature.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.3.3.8</td>
<td>Replace linking by cell address with linking by range name between spreadsheet files.</td>
</tr>
</tbody>
</table>

### S.4 Outputs

<table>
<thead>
<tr>
<th>Category</th>
<th>Skill Set</th>
<th>Ref.</th>
<th>Task Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>S.4 Outputs</td>
<td>S.4.1 Appropriate Display</td>
<td>S.4.1.1</td>
<td>Reveal data hidden by: custom formatting, white font colour, suppression of zero values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.4.1.2</td>
<td>Modify ROUND functions in a cell range in order to display numbers correctly.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.4.1.3</td>
<td>Correct cell content of an incompatible data type such as numbers entered as text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.4.1.4</td>
<td>Correct a cell range incorrectly sorted by one column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.4.1.5</td>
<td>Correct database range in a worksheet to get correct query output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.4.1.6</td>
<td>Correct database criteria.</td>
</tr>
<tr>
<td>S.4.2 Charts</td>
<td></td>
<td>S.4.2.1</td>
<td>Modify chart layout so that all data series are clearly visible.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.4.2.2</td>
<td>Modify the scale of chart axes to clarify chart output.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S.4.2.3</td>
<td>Modify chart type to clearly express the meaning of data.</td>
</tr>
<tr>
<td>Category</td>
<td>Skill Set</td>
<td>Ref.</td>
<td>Task Item</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>----------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>S.5 Review</td>
<td>S.5.1 Testing</td>
<td>S.5.1.1</td>
<td>Create and run test cases based on: typical input values, extreme values, covering all logic paths.</td>
</tr>
<tr>
<td>S.5.1.2</td>
<td></td>
<td></td>
<td>Verify outputs by using a different calculation method such as: SUMIF function to verify DSUM function.</td>
</tr>
<tr>
<td>S.5.1.3</td>
<td></td>
<td></td>
<td>Unhide formulas, rows, columns, worksheets.</td>
</tr>
<tr>
<td>S.5.1.4</td>
<td></td>
<td></td>
<td>Show all formulas in a worksheet.</td>
</tr>
<tr>
<td>S.5.1.5</td>
<td></td>
<td></td>
<td>Inspect all formulas in a worksheet.</td>
</tr>
<tr>
<td>S.5.2 Data Integrity</td>
<td>S.5.2.1</td>
<td></td>
<td>Use IF function to test if cell contents are within defined parameters.</td>
</tr>
<tr>
<td>S.5.2.2</td>
<td></td>
<td></td>
<td>Use conditional formatting to highlight specific data attributes.</td>
</tr>
<tr>
<td>S.5.2.3</td>
<td></td>
<td></td>
<td>Set data validation criteria: any value, whole numbers, decimals, from a list, date, time, text length.</td>
</tr>
</tbody>
</table>
A Paradigm for Spreadsheet Engineering Methodologies

Thomas A. Grossman
University of San Francisco, School of Business & Management, San Francisco CA
94117-1045
tagrossman@usfca.edu

Özgür Özlük
ISBA Department, College of Business, SFSU San Francisco, CA 94132
ozgur@sfsu.edu

ABSTRACT

Spreadsheet engineering methodologies are diverse and sometimes contradictory. It is difficult for spreadsheet developers to identify a spreadsheet engineering methodology that is appropriate for their class of spreadsheet, with its unique combination of goals, type of problem, and available time and resources. There is a lack of well-organized, proven methodologies with known costs and benefits for well-defined spreadsheet classes. It is difficult to compare and critically evaluate methodologies. We present a paradigm for organizing and interpreting spreadsheet engineering recommendations. It systematically addresses the myriad choices made when developing a spreadsheet, and explicitly considers resource constraints and other development parameters. This paradigm provides a framework for evaluation, comparison, and selection of methodologies, and a list of essential elements for developers or codifiers of new methodologies. This paradigm identifies gaps in our knowledge that merit further research.

1. INTRODUCTION

Our goal is to see spreadsheet research mature into an important, widely-respected field, which generates research results that are routinely used in business. This goal will be achieved when spreadsheet developers regularly consider which spreadsheet engineering methodology they will apply to a particular spreadsheet. A spreadsheet engineering methodology provides prescriptive recommendations for the choices made throughout the lifecycle of a spreadsheet. Four barriers must be overcome to achieve this goal.

The first barrier is lack of a compelling value proposition. Despite the extensive research on spreadsheet errors [Panko 1998] and the occasional major error that appears in the business press, the argument that current spreadsheet development practices are risky is having little impact. As [Pettifor 2003] points out, “the world is not falling apart through spreadsheet errors”. The argument that something must be done about spreadsheet errors is so far achieving little traction in the business world.

With the benefit of hindsight, the limited success of the errors/risks argument is not surprising. In essence, this is an attempt to sell a problem. People don’t invest in problems, they invest in solutions. For spreadsheet research to have impact on business practice, it must look beyond errors and their consequences, to the creation of solutions. These solutions—spreadsheet engineering methodologies—must have a compelling value proposition so that busy spreadsheet developers will invest in learning and applying them. An attractive value proposition must include benefits that are important to spreadsheet developers.
developers who are relatively unconcerned with risks and errors. These might include more enjoyable development, greater job satisfaction, cost and personnel savings, reduced development time, lifecycle productivity, or enhanced quality of analysis and insight. The contents of this value proposition is an important research question that can only be answered by empirical research on spreadsheet developers.

We note that even if the spreadsheet research community were to be successful in persuading spreadsheet developers and senior management to take spreadsheet risks seriously, those risks can be mitigated only by the use of appropriate spreadsheet engineering methodologies. The barriers below militate against such use.

The second barrier is lack of knowledge of spreadsheet practice. Spreadsheets are undoubtedly the most widely used programming language, and are used for countless different purposes with wide variety in development practices. Unfortunately, there is no systematic knowledge about this diversity of usage and development. This diversity makes it difficult to develop useful generalizations or theories regarding spreadsheets. As discussed by [Grossman and Özlük 2003], any recommendations or theories will apply to only a particular class of spreadsheets with similar characteristics. Empirical research is needed to identify the most important spreadsheet classes so that suitable spreadsheet engineering methodologies can be devised.

The third barrier is lack of a roadmap to appropriate spreadsheet engineering methodologies. Take the point of view of a developer about to embark on a spreadsheet development project. The developer has a certain amount of time and other resources available, is working on a particular type of problem, and has certain (perhaps vaguely defined) goals. What spreadsheet engineering methodology should he adopt? The current spreadsheet engineering literature is not easily accessible to such a developer. The developer must select among multiple methodologies. It can be difficult to understand which practices are appropriate to a particular spreadsheet, and to match the resources required by a methodology to the resources available. Indeed, existing spreadsheet engineering recommendations are sometimes contradictory, because different spreadsheet classes require different strategies. There is a need for a roadmap that starts with the spreadsheet class and resource constraints and guides developers to appropriate methodologies. We need a theoretical framework, or paradigm, to rigorously and systematically organize and critically evaluate spreadsheet engineering methodologies, including identification of their classes, benefits, and resource implications.

The fourth barrier is a lack of well-organized, proven solutions with known costs and benefits for well-defined classes. Developing a roadmap requires a portfolio of spreadsheet engineering methodologies from which developers can choose. To do this, various spreadsheet classes must be identified, and provided with appropriate spreadsheet engineering methodologies. These methodologies must then be compared to alternative methodologies to elucidate when they are most appropriate. Finally, the methodologies need to be tested or otherwise proven to be beneficial, and the proven benefits and demonstrated resource needs must be clearly stated. This is a significant, long-term challenge for spreadsheet researchers. We provide a paradigm of spreadsheet engineering that facilitates the efficient development of spreadsheet engineering methodologies by identifying a set of essential elements that any methodology must consider.

2. ESSENTIAL ELEMENTS OF THE PARADIGM

In this section we present a nine-element paradigm for spreadsheet engineering methodologies that facilitates organizing, interpreting, and critically evaluating
spreadsheet engineering recommendations. This paradigm provides a vehicle to compare and contrast different spreadsheet engineering recommendations to aid developers in selecting methodologies, and researchers in understanding and improving them. It provides for explicit statements about the relevant classes, and the resources necessary to use the methodology. This paradigm enables us to evaluate the completeness of a spreadsheet engineering methodology. It provides a list of essential elements to developers of new spreadsheet engineering techniques and codifiers of existing practices.

When working with a spreadsheet, the developer makes a series of choices about what to do and how to do it, such as how to organize the cells in the spreadsheet, and what documentation to provide. These choices can be made consciously or unconsciously. If made consciously, they can be made with careful analysis and reflection, or with only momentary consideration. In aggregate these choices determine the efficiency of development, the accuracy of the spreadsheet, and the ability to modify the spreadsheet in the future. These choices are the essence of spreadsheet engineering.

A spreadsheet engineering methodology provides prescriptive recommendations for the choices made throughout the lifecycle of a spreadsheet. By identifying, organizing, and labeling these choices, we can create a paradigm of spreadsheet engineering methodologies. We structure these choices into nine elements. Every activity in the lifecycle of a spreadsheet fits into one of these elements. The purpose of the paradigm is to clarify and articulate distinct concepts relating to spreadsheet development and usage. Our intention is that any spreadsheet engineering methodology can be mapped into this paradigm. The nine elements of our paradigm are below.

1. Modeling
2. Development Parameters
3. Design
4. Programming
5. Quality Control
6. Debugging
7. Documentation
8. Usage
9. Modification

A meaningful spreadsheet engineering methodology must consider the problem-solving context in which spreadsheets are created and used. Therefore, the scope of spreadsheet engineering begins with the recognition that a spreadsheet shall be used to address a business problem, and includes all spreadsheet activity through to usage and modification of spreadsheet. Note that we do not consider whether a spreadsheet is the “right” software for the problem; the assumption in this paradigm is that the developer has chosen to build a spreadsheet, and will benefit from guidance on using it well.

Our paradigm is correct if every possible spreadsheet engineering methodology can be mapped onto it in only one way. The choice of elements is somewhat arbitrary. What is important is that the elements be individually distinct and collectively exhaustive. We anticipate this paradigm will be refined as spreadsheet engineering research progresses.

Any particular spreadsheet engineering methodology may or may not proceed in the same order that the elements are listed. Many methodologies commingle the elements in the interest of efficiency. (This is desirable, but it makes it difficult to compare and evaluate methodologies.) For example, all spreadsheet engineering recommendations suggest that minimal documentation (element 7) such as row and column labels be done during programming (element 4). When mapping a particular methodology onto our paradigm, it
will be necessary to disentangle the various elements to distinguish between the principles embodied in the methodology, and the recommended process of applying those principles. In the subsequent sections, we find it useful to distinguish between a “developer” and a “user”; a developer is involved with building the spreadsheet through tasks such as choosing column and row labels and writing cell formulas, whereas a user simply enters inputs, and observes and interprets outputs.

3. ESSENTIAL ELEMENT 1: MODELING

Modeling is the act of determining what the spreadsheet shall do. Modeling is a component of business problem-solving. The need to solve a problem motivates modeling, which in turn motivates computation. A spreadsheet is a visual computer implementation of a mathematical model. The model embodied in any spreadsheet can be written as a set of algebraic equations which can, in principle, be computed by hand, or coded in a procedural computer language. A spreadsheet model—like any model—takes a set of inputs, and computes a set of outputs. Therefore, we formally define modeling as determining the inputs and outputs, and detailing how outputs shall be computed from inputs. Modeling includes considerations of the problem domain discussed in [Grossman 2002]. The best overview of modeling in isolation from programming is chapters 1 – 4 of Powell and Baker 2004.

We intentionally avoid the use of the term “specification” in our definition of modeling. A specification, whose roots are the waterfall lifecycle model of traditional software engineering, describes in great detail the function of a computer program prior to programming. This can be a powerful tool when working with procedural computer languages. In contrast, one of the most powerful capabilities of spreadsheets is their capability to program while modeling. It is apparent that most spreadsheets do not have formal specifications, and it is unlikely that spreadsheet developers will become avid specification writers. Therefore, creating a specification is but one choice that a developer can make, and which will often be declined. A key challenge for spreadsheet engineering researchers is to identify those situations where a specification is indeed essential, cost-effective, or otherwise appropriate.

Because spreadsheets are a powerful vehicle for modeling, modeling is often integrated with spreadsheet design and programming. This can obscure the role of modeling as an independent intellectual activity. The relationship between modeling and programming is an essential aspect of any software engineering methodology. This relationship can range from complete separation to complete integration. Methodologies such as the classic waterfall lifecycle model and the use of Jacksonian Structured Programming [Chadwick et al 1999] recommend the completion of modeling before the start of programming. Certain lifecycle models such as the spiral model [McConnell 1996] provide for a sequence of distinct modeling and programming steps. [Nardi and Miller 1991] describe how spreadsheet users and developers cooperate in creating spreadsheets, with programming and modeling partially integrated. [Grossman 2002] discusses how developers can engage in exploratory modeling, where they program a spreadsheet to help them think through and understand their business problem fully integrating modeling and programming.

It is important that any spreadsheet engineering methodology address modeling, which is the process of figuring out what the spreadsheet is to do, and carefully discuss the interaction of modeling and programming.
4. ESSENTIAL ELEMENT 2: DEVELOPMENT PARAMETERS

Development parameters are the planning assumptions of a spreadsheet. This includes the goals of the spreadsheet; the budget in terms of money, time and developer labor; the users in terms of their number, skill, and experience; the frequency of use; the time period of use; the likelihood and nature of modifications after usage; interactions with other information systems; the importance of the spreadsheet; the desired accuracy; and any other considerations that may affect the spreadsheet during its lifecycle.

The selection of development parameters strongly affects all the steps of spreadsheet development. Unfortunately, because development parameters are prospective, they can be wrong. For example, a spreadsheet intended for one-time usage by its developer might see usage by multiple users. Or a spreadsheet that was to be programmed once and deleted is modified for other uses. Poor selection of development parameters at the beginning can cause expense and risk later. Therefore, the establishment of development parameters includes any evaluation of risks, such as errors and development failure.

Development parameters are essentially business judgments about the deployment of resources to create information systems to achieve organizational goals. Therefore, development parameters are controlled by business considerations and resource constraints, not by any inherent properties of the model to be developed.

We believe that consideration of development parameters is an essential component of any spreadsheet engineering methodology. A given spreadsheet engineering methodology is more appropriate for some development parameters than others. However, there is a tendency in the spreadsheet engineering literature to provide insufficient discussion of development parameters. Spreadsheet engineering methodologies should clearly identify any assumptions of development and usage, and discuss the resources required during initial development and potential future modifications.

5. ESSENTIAL ELEMENT 3: DESIGN

The design of a spreadsheet comprises two elements: structural design and visual design. Structural design is the way cells are arranged. Structural design includes the designation of rows and columns to have particular meaning, the use of modularity, and the provision of space for documentation. Visual design refers to the appearance of cells and cell borders. Visual design includes shading, borders, fonts and other formats.

The spreadsheet engineering literature is in agreement that good design is important. However, there is no agreed list of what constitutes good design. Many discussions in the literature mingle design considerations with programming and documentation.

Two principles of structural design are widespread. The first is to organize related concepts using the rows and columns of the spreadsheet. For example, each column of a cash flow statement contains a single year, and each row contains a single accounting concept.

The second structural design principle is “modularity”, which says that logically related elements be grouped into modules. A module might contain model inputs, model outputs, a summary with selected inputs and outputs, a set of computations, or other items. The module(s) that a spreadsheet user interacts with are called the “user interface” and often require special attention. A module can comprise a single cell, a section of a worksheet, an entire worksheet, a workbook, or even a set of linked workbooks. Modules can contain submodules. For example, the authors recently observed a 27 MB workbook of a user interface for a large spreadsheet application. The workbook contains numerous...
submodules in the form of worksheets, with each worksheet containing a number of submodules in the form of sections.

The final structural design principle is to provide space for documentation. Space should be provided for row and column labels, and any other documentation. This is discussed in more detail in the Documentation section below.

Visual design is essentially the formats applied to the spreadsheet. The most important visual design elements are the use of fonts (including font choice, color, and bold/italic), justification within a cell, cell colors, and cell borders. Visual design draws attention to important elements in the spreadsheet; differentiates among inputs, intermediate variables and outputs; communicates and reinforces modularity; and establishes hierarchical relationships within a module.

Visual design elements are important for several reasons. [Nardi and Miller 1990] argue that spreadsheets success relies on the strong visual format opportunities for structuring and presenting data. [Reithel et al 1996] tells us that well-formatted spreadsheets are perceived as more accurate. However, [Raffensperger 2000] argues that certain formats such as excessive color, and non-constant column widths may reduce comprehension or become a distraction.

6. **ESSENTIAL ELEMENT 4: PROGRAMMING**

*Programming* is the creation of cell formulas and other logic in a spreadsheet. Programming techniques range from broad principles such as “build a small-scale prototype and then scale up” to precise recommendations such as “do not hard code constants in cell formulas”.

Here is a limited list of programming techniques. Enter each input exactly once. Replicate cell formulas using copying rather than typing. Use absolute references in cell formulas to facilitate copying. Create a scenario tool to run multiple datasets through the same logic. Check that input values are within established ranges. Use data validation tools to prevent users from entering out-of-range inputs. Use cell protection to prevent accidental or unauthorized modification of cell formulas. Check intermediate calculations for out-of-range values. Use cross-foots and other redundant calculations. Use version control. Make files read-only to prevent accidental overwriting of important information. Use spreadsheet productivity features (such as Insert/Function..., and selecting cell references rather than typing them) to avoid syntax errors in formulas. There are many other techniques.

There are some contradictions and open questions in the literature. Some authorities recommend avoidance of certain spreadsheet functions deemed risky, such as OFFSET, but others recommend OFFSET as being useful. Many spreadsheet auditing packages flag the use of multiply-nested IF functions, yet some well-engineered spreadsheets use deep nesting. The use of range names is recommended, but there are times when range names interfere with copying formulas or modifying a spreadsheet. [Thommes 1994] and [Caine and Robson 1993] recommend splitting lengthy cell formulas into smaller parts to keep the formulas simple and easy to understand, whereas [Raffensperger 2000] argues that splitting formulas may result in bloated hard-to-read spreadsheets.

Spreadsheet programming currently resemble a bag of tricks rather than a well-organized, intellectually coherent toolbox. Research is needed to codify and organize the techniques. Contradictory recommendations should be identified, and these contradictions resolved by specifying the development parameters or design where each technique is appropriate. It would be helpful to bring intellectual coherence to the dizzying array of techniques, by
categorizing them and distinguishing among high-level principles and low-level practices. Of particular interest for spreadsheet engineering is how development parameters affect programming practices and how programming practices affect spreadsheet quality and usage.

7. **ESSENTIAL ELEMENT 5: QUALITY CONTROL**

*Quality Control* is all actions taken to determine whether the outputs of a spreadsheet are satisfactory. There are two aspects to quality control, “verification” and “validation”. Verification is concerned with the programming of the model, and validation is concerned with the meaningfulness of the model as implemented.

**Verification** evaluates accuracy. It asks whether the spreadsheet program correctly implements the model. Verification answers the question “does the spreadsheet contain programming errors?” In principle, verification is an objective evaluation.

**Validation** evaluates model quality. It asks whether the model adequately depicts reality, and how closely model outputs correspond to real world values. Validation answers the question “is the model adequate?” In some situations, such as simple taxation models, validation can be objectively evaluated. In other situations, such as a complex model used by a bank to set the residual prices on car leases, objective evaluation is impossible.

There are two general approaches to verification, code inspection and testing. The software engineering literature and [Panko 1999] argue that multi-person code inspection has the highest error-detection rate. Spreadsheet auditing tools automate code inspection for certain errors.

Testing is entering test data into a model and observing the outputs. The most useful test inputs are those with corresponding outputs that are known to be correct. For spreadsheets that address a problem that has never previously been modeled, the generation of test cases is a significant challenge. The theoretically rigorous approach to testing in [Rothermel et al 2001] assumes that test cases are available. Probably the best reference on testing spreadsheets that lack test cases is the brief and non-comprehensive discussion in chapter 5 of [Powell and Baker 2004].

In some cases, working through the model logic manually may be the only practical way to obtain a test case. In extreme situations, it may be necessary to build independently a parallel system, and compare results. Future research should consider how to test spreadsheets where the correct outputs are not known *a priori*.

Validation and verification are distinct concepts. An inadequate model that is programmed well is verified and invalid. A satisfactory model that has programming errors is inverified and valid. (Note: a model that has not been evaluated for accuracy is “unverified”, a model that has failed the evaluation is “inverified”.) Unfortunately, verification and validation are sometimes conflated in practice. When a developer evaluates a spreadsheet by examining the outputs and judging that they “seem about right” they are engaging in validation rather than verification. They risk making the (often unstated) assumption that validation insures verification. It does not. It simply insures that any errors tend towards what the developer expected for the inputs being used. This is called “confirmation bias” and can mask errors in a spreadsheet. There is no research on the prevalence of this practice, but anecdotal evidence suggests it is widespread. In fact, [Burnett et al 1999] take this approach, where testing is performed by developers “noticing” whether a particular cell contains a correct or incorrect value.

When verification detects an error in the programming, or validation detects an error in the model, it is necessary to fix the problem. This is called debugging.
8. ESSENTIAL ELEMENT 6: DEBUGGING

Quality control finds problems, and debugging fixes them. Debugging is modifying a spreadsheet program to fix an output that has an unsatisfactory value. There are three key issues in debugging: how to locate the source of the problem, how to fix the problem, and how to avoid introducing new bugs.

Unfortunately, the spreadsheet engineering literature contains little guidance on debugging, particularly in a spreadsheet with complex contingent logic programmed with lookup formulas or nested IF formulas. Spreadsheet auditing software can be helpful in locating the problem, and can sometimes provide guidance in fixing it. When the issue with the spreadsheet is the invalidity of the underlying model, it may be necessary to revisit element 1, modeling, to enhance the model. Then, it is necessary to change the spreadsheet to incorporate the enhancements. Making these changes accurately and efficiently is a similar if not more demanding skill compared to making changes to eliminate a programming error.

The software engineering and quality control literatures argue persuasively that preventing errors is cheaper than finding and fixing errors. Thus, incremental investment in design and programming can bring disproportionate savings in debugging. Provided of course, quality control is done at all!

Development of techniques for debugging is an area that has not received enough attention and merits further research. Empirical research on quality control and debugging practices would be valuable.

9. ESSENTIAL ELEMENT 7: DOCUMENTATION

Documentation is any written record regarding the spreadsheet. The most common form of documentation is row and column labels in a spreadsheet. These can range from minimal abbreviations, to lengthy formal names. Documentation can reside in many places. It can be integrated with cell formulas within a module, for example row and column labels, and a notes column. Documentation can reside in its own documentation module within a spreadsheet, or it can be in the form of a separate document such as a full-fledged user’s manual. Microsoft Excel has features such as Comments and text boxes that allow documentation to be placed almost anywhere in the spreadsheet. The programming technique of range names can make cell formulas more readable and serves as a form of documentation.

Any element of a methodology can be documented, including modeling, development parameters, design, programming, quality control, and debugging. Documentation can consume substantial resources, and the amount of documentation to be done depends on the development parameters.

Spreadsheet engineering methodologies must carefully consider the appropriate level and type of documentation, and the resources required to create it. It is well known from software engineering that documentation is often inadequate, and there is anecdotal and empirical evidence that spreadsheet documentation is inadequate. Therefore, it may be desirable for methodologies to distinguish between essential documentation and desirable documentation.

10. ESSENTIAL ELEMENT 8: USAGE

We define the usage of a spreadsheet to be any process where a user provides inputs to a spreadsheet, and observes the outputs. Usage does not involve programming.
Planned usage is considered in the development parameters element. Actual usage may differ from planned usage. This can signal unanticipated success. Such success can be a mixed blessing, because unplanned usage implies the spreadsheet is a poor platform for those newly-discovered uses, whatever they may be.

There is great diversity in usage, but we have limited theoretical and empirical knowledge because spreadsheet usage receives little attention in the literature. Usage can be by the developer, or by other individuals. Usage can be by one individual or many. Usage can be a single observation of a set of model outputs, or can involve multiple sets of inputs used to generate multiple sets of outputs. Usage can be a one-off event, or can take place regularly or irregularly over time. The user may or may not be able to interact with the developer. There are questions about how users can analyze a spreadsheet model (or any model) to systematically extract insight about a business process. There are also questions on how spreadsheets are shared by different people.

This situation is problematic. The usage of a spreadsheet is particularly important to spreadsheet development because expected usage helps determine the development parameters. We are ignorant of usage, and therefore cannot present compelling, evidence-based suggestions for integrating usage expectations into spreadsheet development. We know that usage expectations are sometimes wrong, but we do not know how often or how expected usage correlates to actual usage.

Clearly, rigorous research on spreadsheet usage would be beneficial.

11. ESSENTIAL ELEMENT 9: MODIFICATION

*Modification* refers to changes made to the spreadsheet after it has been used. This includes terms such as “maintenance”, “enhancement” and “extension”. Like any software, modification of spreadsheets can be substantially more expensive than building in features from the beginning, and provision for modifications made early in development can significantly reduce the time, cost and risk of making modifications.

We know little about modifications. We know that spreadsheets whose development parameters indicated no modifications may indeed be modified after usage. Even when modifications are included in the development parameters, it may not be possible to anticipate the nature of the modifications. There is no systematic research on the origin of modification requests, and how these connect to usage, development parameters, design and programming decisions.

There is a clear opportunity for research on spreadsheet modifications. It would be helpful to have a categorization of the kinds of modifications that are made, who proposes them, and the effect of early-development planning (or its lack) on later modifications. In particular, it would be valuable to better understand the twin risks of over-engineering for modifications that never happen vis-à-vis the risk of under-engineering for unexpected modifications that later prove necessary.

12. CLASSES OF SPREADSHEETS

Because of the great diversity among spreadsheets and spreadsheet developers, it is difficult to make detailed spreadsheet engineering recommendations that are widely applicable. In contrast, recommendations with narrow scope, pertaining to specific classes of spreadsheets, can provide detailed and specific guidance. The level of detail and specificity of spreadsheet engineering recommendations is inversely proportional to the scope of the recommendations. Therefore, it is important that any spreadsheet engineering methodology carefully define the class of problems to which it applies. In our nine-
element paradigm, the defining characteristics of a class are to be found in the development parameters and modeling elements. When the class is well-defined, highly specific methodologies can be provided. For example, [Conway and Ragsdale 1997] consider the narrowly-defined class of small scale optimization models, and are able to provide very specific recommendations. Therefore, it is essential that a spreadsheet engineering methodology clearly indicate in the development parameters and modeling elements the class of spreadsheet to which it applies. We note that many existing spreadsheet engineering methodologies provide insufficient class information, and have hidden assumptions about where they can usefully be employed. This reduces their effectiveness, because some of their recommendations are ineffective or even inappropriate in certain classes, and developers for whom the recommendations are most appropriate may not recognize the relevance of the methodology.

13. CONCLUSIONS

Our hope is to see spreadsheet research mature into an important, widely-respected field, which generates research results that are extensively used in business. This entails prescriptive research with sufficient power and applicability to motivate adoption and employment by busy spreadsheet developers. This power will emerge only with specific, detailed methodologies. Applicability will emerge with carefully specification of class. In principle, a specific spreadsheet engineering methodology can be defined for any class of spreadsheet. In practice, methodologies will probably only be defined for classes where significant value can be obtained through better practices.

Currently, it is difficult to compare, contrast, and critically evaluate spreadsheet engineering methodologies. This is because the methodologies are organized to support a particular development process, and it is challenging to decompose them into their components to observe commonalities and differences. More important, it can be difficult to recognize the hidden assumptions that underlie many methodologies, particularly assumptions about class. We present paradigm of spreadsheet engineering that will help with these difficulties. By mapping spreadsheet engineering recommendations onto the nine essential elements of this paradigm, it will be easy to compare and evaluate methodologies, and determine their completeness. This paradigm provides a framework for developing new spreadsheet engineering methodologies, and makes explicit provision for identifying the modeling approach and development parameters which together define the class. This paradigm identifies gaps in our knowledge that can guide further research. Important research opportunities include systematizing and organizing the wealth of programming techniques; devising techniques for testing spreadsheets that lack test cases; systematic methods for debugging spreadsheets; better understanding spreadsheet usage; and increasing our knowledge of spreadsheet modifications. Finally, there is an opportunity to systematically interpret existing spreadsheet engineering methodologies in light of our nine-element paradigm, and compare models to identify commonalities.

Future research on spreadsheet engineering should identify high-value classes with large numbers of developers who are sensitive to the investment they make in spreadsheets. These are the audiences most likely to adopt new spreadsheet engineering methodologies. Spreadsheet developers should be interviewed to identify what value proposition would induce them to invest in deploying new methodologies in their organizations.
With the class carefully defined, and a clear sense of the benefits that developers need to see, researchers can use our nine-element paradigm to devise an appropriate, detailed spreadsheet engineering methodology, which they can then test against current practice to determine the benefits and required resources.

REFERENCES


XI Struct: A Tool for Building Structured Error-Resistant Spreadsheets

Gary K. Arakaki
Seio Hills 401, 4-6-10 Ogikubo
Suginami-ku, Tokyo 167
Japan
garvarakaki@yahoo.com

ABSTRACT

Spreadsheets are extremely useful but are very susceptible to errors. This susceptibility to errors is direct consequence of the way spreadsheets are built. The problem is the usual way of building spreadsheets makes it easy to make an error. XI Struct provides a new way of building spreadsheets that is much less susceptible to errors. XI Struct is the result of a complete rethinking of how spreadsheets should be built. XI Struct is an add-in for Microsoft Excel for building structured, error resistant spreadsheets.

1. INTRODUCTION

Spreadsheets are very useful but very prone to errors. This is really a direct consequence of the way spreadsheet applications allow users to build spreadsheets. Although, there are methodologies that users can follow which will help them to build more reliable spreadsheets, it is very likely that spreadsheets will continue to be built without using these methodologies as it is unreasonable to expect that average user will be conversant with these methodologies. It may be better if spreadsheet users were given a new easy way of building spreadsheets that naturally forces the spreadsheets to be built in a way that is conducive to preventing many of the errors from occurring.

The basic technology for building spreadsheets has not changed for a very long time. In the meantime computers have become vastly more powerful. These vastly more powerful computers make it possible to consider innovative new ways of building spreadsheets that were impractical on the earlier generations of computers. XI Struct is the result of a complete rethinking from scratch of how spreadsheets should be built. XI Struct is an add-in for Microsoft Excel that provides the user with a new easy way of building structured spreadsheets that automatically eliminates many of the errors that would be caused by incorrect cell references and by incorrect replication of cells.

Structuring the spreadsheet allows XI Struct to provide alternatives to three basic Excel features: relative references, the SUM() function and the copy and paste operation. Although, much of the power of Excel is derived from using these three features, using these three features to build spreadsheets also make it easy to produce spreadsheets with errors. XI Struct provides simple, intuitive alternatives to these three features that automatically make it much harder to produce spreadsheets with errors. XI Struct supports localized references, a flexible, reliable alternative to Excel's relative references, the SumNamed() function, a powerful alternative to Excel's SUM() function and the clone operation, an alternative to Excel's copy and paste operation that permanently links the cloned region with the original region. These new capabilities are made possible by using a conceptually simple but powerful technology for structuring the cells on a spreadsheet into regions. This technology has the following three features:
1. localized references – referencing cells by name and locality

2. SumNamed() functions – sum selectively by name some of the cells in a region

3. cloning of a progenitor – a technology for cloning a region from a progenitor and for synchronizing the cells of a cloned region with the cells of the progenitor after the progenitor has been modified

2. REGIONS

XlStruct uses regions to structure the cells on a spreadsheet. A region is a named rectangular block of cells. A region may completely contain another region or be completely contained in another region but a region may not partially overlap another region. Therefore with respect to containment the regions of a spreadsheet form a tree hierarchy where the region consisting all cells of the spreadsheet is the root of the tree hierarchy. A single cell may be a region. A container is a region of more than one cell. A region that is contained in a container is said to be a member of that container. Multiple regions may have the same name. In XlStruct a name is not intended to uniquely identify a region but rather, a name really specifies the type of data contained in a region. This slight paradigm shift will make names much more useful in formulas as we will soon see. The regions should be chosen to correspond to natural hierarchy of the underlying data in the regions of the spreadsheet. For example, the regions may organize the cells of the spreadsheet geographically by city, state and country.

3. LOCALIZED REFERENCES

A formula in a cell may be able reference another cell by its XlStruct name. However, since multiple cells may have the same XlStruct name another condition is necessary to give uniqueness. The basic idea behind the technique that XlStruct uses to resolve the ambiguity of multiple cells having the same XlStruct name is to select the cell with the specified XlStruct name that has the same “locality” as the cell containing the formula referencing the cell by XlStruct name. Specifically, if multiple cells have the specified XlStruct name then the cell in the smallest region that contains the cell with the formula and contains a cell with the specified XlStruct name is selected as the referenced cell. In other words, XlStruct searches successive nested containers of the cell containing the formula beginning with the smallest (innermost) container until a container is found that contains a cell with the specified XlStruct name. If the found container contains more than one cell with the specified name then one of the candidates will be arbitrarily selected as the referenced cell. (Obviously, the user should avoid building spreadsheets where references are not uniquely resolved.) In XlStruct this way of referencing cells is referred to as localized referencing. Note that if a region is copied and pasted then the localized references in formulas naturally references the corresponding cells in the new locality. (This provides the same capability as Excel’s relative references but uses user-friendly names instead of relative cell references.) The following example below will make this clear.
To resolve the localized reference "DDD" in the formula "=3*DDD" contained in the cell B4 XIStruct first searches the region named "BBB" (B3:C4) which is the smallest container containing the cell B4 for a cell named "DDD" and does not find one. Then XIStruct searches the region named "AAA" (B2:D4) which is the next smallest container containing the cell B4 for a cell named "DDD" and finds the cell D2. Thus the localized reference "DDD" in the formula "=3*DDD" contained in cell B4 is resolved to the cell D2. The localized reference "DDD" in the formula "=2*DDD" contained in the cell F2 is resolved to the cell G5 and not to the cell H6 since the region named "YYY" (F2:G5) is smaller than the region named "XXX" (F2:H6).

### 4. THE **SUMNAMED()** FUNCTION

XIStruct provides a function called **SumNamed** for use in cell formulas. **SumNamed** takes two arguments: a pattern for matching names and the name of a container. **SumNamed** will sum the value of all cells that are contained in the specified container and have a name that matches the specified pattern. The specified container is the smallest container that has the specified name and contains the cell that contains the formula that contains the SumNamed expression. The pattern is a regular...
expression. Although a container may contain cells of many different types, SumNamed will sum the value of only those cells in the container with names that match the specified regular expression. For example detail items and subtotals may be contained in the same container but only the detail items may be selectively summed to obtain a grand total if the names of the detail items and the names of the subtotal items can be differentiated by a regular expression. Furthermore, when new rows are added to the bottom of a container the SumNamed function automatically includes the new rows since the container automatically expands to accommodate the new rows. The following example will make this clear.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td></td>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>60</td>
<td>660</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1001</td>
<td>1010</td>
<td>1020</td>
<td>1030</td>
<td>1100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>1002</td>
<td></td>
<td></td>
<td>1200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>1003</td>
<td></td>
<td></td>
<td>1300</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3006</td>
<td>3060</td>
<td>6660</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td>Name</td>
<td>Contents</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>---------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2:G6</td>
<td>AAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>BBB</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>BBB</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td>BBB</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>CCC1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>CCC2</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>CCC3</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G2</td>
<td>DDD1</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td>DDD2</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G4</td>
<td>DDD3</td>
<td>300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B6</td>
<td>EEE1</td>
<td>=SumNamed(&quot;BBB&quot;,&quot;AAA&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C6</td>
<td>EEE2</td>
<td>=SumNamed(&quot;CCC.+&quot;,&quot;AAA&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D6</td>
<td>EEE3</td>
<td>=SumNamed(&quot;(CCC.<em>)(DDD.</em>)&quot;,&quot;AAA&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B8:G12</td>
<td>AAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B6</td>
<td>BBB</td>
<td>1001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C9</td>
<td>BBB</td>
<td>1002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D10</td>
<td>BBB</td>
<td>1003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C8</td>
<td>CCC1</td>
<td>1010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D8</td>
<td>CCC2</td>
<td>1020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E8</td>
<td>CCC3</td>
<td>1030</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G8</td>
<td>DDD1</td>
<td>1100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>DDD2</td>
<td>1200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G10</td>
<td>DDD3</td>
<td>1300</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B12</td>
<td>EEE1</td>
<td>=SumNamed(&quot;BBB&quot;,&quot;AAA&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C12</td>
<td>EEE2</td>
<td>=SumNamed(&quot;CCC.+&quot;,&quot;AAA&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D12</td>
<td>EEE3</td>
<td>=SumNamed(&quot;(CCC.<em>)(DDD.</em>)&quot;,&quot;AAA&quot;)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The formula ";=SumNamed("BBB","AAA")" in cell B6 sums all the cells with name "BBB" in the container named "AAA" at B2:G6 (This is the smallest container named "AAA" that contains the cell B6.) which are the cells B2, C3 and D4. The formula ";=SumNamed("CCC.+","AAA")" in cell C6 sums all the cells that have names beginning with "CCC" in the container named "AAA" at B2:G6 which are the cells C2, D2 and E2. The formula ";=SumNamed("(CCC.*)(DDD.*)","AAA")" sums all the cells that have names beginning with "CCC" or "DDD" in the container named "AAA" at B2:G6 which are the cells C2, D2, E2, G2, F3 and G4. The formulas in the container named "AAA" at B8:G12 (This container is a clone of the container at B2:G6,) sum the respective cells in this container. Note that the formulas are the same in both containers but they automatically reference cells in their respective containers.
5. THE CLONE OPERATION

A region may be cloned. The original region is called the progenitor. The cells of the progenitor may be designated as propagateable or non-propagateable by the spreadsheet builder. A clone is said to be synchronized with its progenitor if the clone has exactly the same member regions at exactly the same relative locations as the progenitor and the contents of all propagateable cells of the progenitor are identical to the contents of the corresponding cells of the clone. The propagateability of a cell in a progenitor contained in nested containers is determined by the propagateability of the innermost container. In XIStruct a clone is permanently linked with its progenitor and a clone will always be synchronized with its progenitor not just at the time of creation but forever. XIStruct propagates to the clones of the progenitor only some of the modifications that have been done to the progenitor and at the same time preserving some of the original contents in the clone. Specifically, changes to the size and/or position of member regions contained in the progenitor are propagated to the clones of the progenitor. The size, position and contents of newly created member regions contained in the progenitor are propagated to the clones of the progenitor. Member regions of the clones that correspond to deleted existing member regions of the progenitor are deleted from the clones. The contents of propagateable cells of the progenitor are propagated to the corresponding cells of clones of the progenitor. Other cells in the clones of the progenitor will have their contents preserved. XIStruct guarantees that the propagateable cells in a clone always have the same contents as the corresponding cells in the progenitor. XIStruct will not allow the user to edit a propagateable cell in a clone. Hence, the user never needs to worry that a propagateable cell may have an inadvertent wrong value. In particular, the user only needs to check non-propagateable cells when verifying his spreadsheet. XIStruct can highlight the propagateable and non-propagateable cells to make the checking easier. This greatly reduces the effort needed to verify a spreadsheet. The following example will make this clear.
### BEFORE EDITING

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>111</td>
<td></td>
<td>2</td>
<td></td>
<td>707</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>333</td>
<td></td>
<td>222</td>
<td></td>
<td>2121</td>
<td></td>
<td>1414</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Range</th>
<th>Name</th>
<th>Contents</th>
<th>Propagateable</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2:D4</td>
<td>AAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2:C3</td>
<td>BBB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>CCC</td>
<td>111</td>
<td>no</td>
</tr>
<tr>
<td>D4</td>
<td>DDD</td>
<td>=EEE*CCC</td>
<td>yes</td>
</tr>
<tr>
<td>D2</td>
<td>EEE</td>
<td>2</td>
<td>yes</td>
</tr>
<tr>
<td>B4</td>
<td></td>
<td>=CCC+DDD</td>
<td>yes</td>
</tr>
<tr>
<td>F2:H4</td>
<td>AAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2:G3</td>
<td>BBB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>CCC</td>
<td>707</td>
<td></td>
</tr>
<tr>
<td>H4</td>
<td>DDD</td>
<td>=EEE*CCC</td>
<td></td>
</tr>
<tr>
<td>H2</td>
<td>EEE</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td></td>
<td>=CCC+DDD</td>
<td></td>
</tr>
</tbody>
</table>

The region F2:H4 is a clone of the progenitor at B2:D4. Since the cells D4, D2 and B4 of the progenitor are propagateable the corresponding cells H4, H2 and F4 of the clone have identical contents. Note that if a propagateable cell in the progenitor contains a formula the corresponding cell in the clone contains the identical formula but not necessarily the same value.
### AFTER SYNCHRONIZATION

<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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>333</td>
<td>3</td>
<td></td>
<td></td>
<td>2121</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10000</td>
<td></td>
<td></td>
<td></td>
<td>10000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10444</td>
<td>111</td>
<td></td>
<td></td>
<td>12828</td>
<td>707</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Range</th>
<th>Name</th>
<th>Contents</th>
<th>Propagateable</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2:D4</td>
<td>AAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2:B3</td>
<td>BBB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td>CCC</td>
<td>111</td>
<td>no</td>
</tr>
<tr>
<td>B2</td>
<td>DDD</td>
<td>=EEE*CCC</td>
<td>yes</td>
</tr>
<tr>
<td>D2</td>
<td>EEE</td>
<td>3</td>
<td>yes</td>
</tr>
<tr>
<td>C3</td>
<td>FFF</td>
<td>10000</td>
<td>no</td>
</tr>
<tr>
<td>B4</td>
<td></td>
<td>=CCC+DDD+FFF</td>
<td>yes</td>
</tr>
<tr>
<td>F2:H4</td>
<td>AAA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2:F3</td>
<td>BBB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H4</td>
<td>CCC</td>
<td>707</td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>DDD</td>
<td>=EEE*CCC</td>
<td></td>
</tr>
<tr>
<td>H2</td>
<td>EEE</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>G3</td>
<td>FFF</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td></td>
<td>=CCC+DDD+FFF</td>
<td></td>
</tr>
</tbody>
</table>

The progenitor "AAA" at B2:D4 has been edited as follows: The region "CCC" at B2 was moved to D4. The region "DDD" at D4 was moved to B2. The region "BBB" at B2:C3 was shrunk to B2:B3. The content of region D2 was changed from "2" to "3". The region "FFF" was created at C3 with contents "10000". The content of B4 was changed from "CCC+DDD" to "CCC+DDD+FFF". After synchronization the clone changes as follows: The region "CCC" at F2 moves to H4. The region "DDD" at H4 moves to F2. The region "BBB" at F2:G3 shrinks to F2:F3. The content of region H2 changes from "2" to "3". The region "FFF" is created at G3 with contents "10000". The content of F4 changes from "=CCC+DDD" to "=CCC+DDD+FFF". Note that the content of the region "CCC" in the clone is unchanged since the corresponding cell in the progenitor is not propagateable.
6. PLANES AND MULTIPLE HIERARCHIES

The same data may naturally be classified into multiple different hierarchies simultaneously. For example the data may be classified geographically by city, state and country and simultaneously classified by time by day, month, quarter and year. XIstruct supports multiple hierarchies on the same spreadsheet using planes. Each spreadsheet has four planes - main, auxiliary 1, auxiliary 2 and auxiliary 3. Each plane supports one hierarchy. When trying to resolve a localized reference XIstruct searches the containers in all the planes in order by size smallest first until the localized reference is resolved. The following example will make this clear.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>Spring</td>
<td>Summer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>North</td>
<td>11</td>
<td>10</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>South</td>
<td>130</td>
<td>120</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>141</td>
<td>130</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Range</th>
<th>Name</th>
<th>Contents</th>
<th>Plane</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3</td>
<td>AMOUNT</td>
<td>11</td>
<td>Main, Aux1</td>
</tr>
<tr>
<td>C4</td>
<td>AMOUNT</td>
<td>130</td>
<td>Main, Aux1</td>
</tr>
<tr>
<td>D3</td>
<td>AMOUNT</td>
<td>10</td>
<td>Main, Aux1</td>
</tr>
<tr>
<td>D4</td>
<td>AMOUNT</td>
<td>120</td>
<td>Main, Aux1</td>
</tr>
<tr>
<td>C2:C5</td>
<td>SEASON</td>
<td></td>
<td>Main</td>
</tr>
<tr>
<td>D2:D5</td>
<td>SEASON</td>
<td></td>
<td>Main</td>
</tr>
<tr>
<td>B3:E3</td>
<td>REGION</td>
<td></td>
<td>Aux1</td>
</tr>
<tr>
<td>B4:E4</td>
<td>REGION</td>
<td></td>
<td>Aux1</td>
</tr>
<tr>
<td>C5</td>
<td></td>
<td>=SumNamed(&quot;AMOUNT&quot;,&quot;SEASON&quot;)</td>
<td></td>
</tr>
<tr>
<td>D5</td>
<td></td>
<td>=SumNamed(&quot;AMOUNT&quot;,&quot;SEASON&quot;)</td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td></td>
<td>=SumNamed(&quot;AMOUNT&quot;,&quot;REGION&quot;)</td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td></td>
<td>=SumNamed(&quot;AMOUNT&quot;,&quot;REGION&quot;)</td>
<td></td>
</tr>
</tbody>
</table>

The "SEASON" regions (C2:C5 and D2:D5) and the "REGION" regions (B3:E3 and B4:E4) cannot exists in the same plane since they partially overlap.

7. References

A proof of concept demonstrator is available at www.xlstruct.com<http://www.xlstruct.com/Xlstruct.htm> 19:44 4/05/04 JST.
TellTable Spreadsheet Audit: from technical possibility to operating prototype

John Nash, Andy Adler and Neil Smith
School of Management, University of Ottawa, Ottawa, K1N 6N5, Canada
nashjc@uottawa.ca

ABSTRACT

At the 2003 EuSpRIG meeting, we presented a framework and software infrastructure to generate and analyse an audit trail for a spreadsheet file. This report describes the results of a pilot implementation of this software (now called TellTable; see www.telltable.com), along with developments in the server infrastructure and availability, extensions to other "Office Suite" files, integration of the audit tool into the server interface, and related developments, licensing and reports. We continue to seek collaborators and partners in what is primarily an open-source project with some shared-source components.

1. BACKGROUND

At the 2003 EuSpRIG Conference in Dublin, we presented our first, "proof of concept" version of a spreadsheet audit trail capability. This showed that it was possible to use the change-recording infrastructure of modern spreadsheet processors to keep track of changes in the underlying spreadsheet file. That is, we were able to record the time and authorship of all changes in each cell of the spreadsheet, and to analyse the contents before and after the change. Furthermore, by permitting filtering of the reported changes, we could focus on particular types of cell changes. For example, we have had a particular interest in changes from formulas to static values, since this type of change would be used to alter a course mark, a return on investment, or the result of an assignment calculation.

The audit trail reporting tool was one part of our work. To prevent users of a spreadsheet file from tampering with both the data and its audit trail, we needed to secure the file in some way. Our choice was to store the files on a server and to run the spreadsheet processor there. Software to do this was the second part of our reported work. At the time of the 2003 Conference, this was working, but we had constructed all the components "by hand". There was no simple installer nor any tools to facilitate many of the server administrative tasks. There had also not been a proof of concept of the whole system nor a pilot implementation.

In building the two complementary parts of the audit trail system, our efforts were considerably simplified by the existence of the OpenOffice spreadsheet processor calc, and the structure of the OpenOffice suite files in general. (Brauer, circa 2004; Sun Microsystems, 2002). These files are stored as compressed text with XML tags, for which there are freely available parsers (part of JVM for J2SE release 1.4; Apache Software Foundation, 1999, 2000; Free Software Foundation, 1999). Furthermore, calc maintains its user interface configuration in XML format. We are thus able to present the user with a spreadsheet processing menu that has the unwanted controls removed, such as those for change-recording. By turning on change recording and preventing the user from turning it off, we ensure the audit trail information is kept.

We believe it is likely that a similar strategy could be used to develop an audit trail for Microsoft Excel, Corel Quattro or Lotus 1-2-3. However, as far as we are aware, the internal file structures of these products are not published, making analysis of the files for audit very difficult. Some
workers appear to have been able to learn the file structure, possibly by reverse engineering. Since we do not wish to have to rely on continuing efforts in such a pursuit, to date we have decided to work only with OpenOffice.org files.

2. DEVELOPMENTS SINCE JULY 2003

The principal developments in our work since the 2003 Conference have been as follows.

2.1 Software Improvements

Significant improvements have been made to the TellTable software, both in terms of usability and security. A read-only mode has been implemented, which is primarily used for viewing of archived file versions. Improvements to the user interface have allowed easier access to archives, and the ability to view, download or audit previous file versions. Security enhancements ensure that the server runs over an encrypted HTTP communication protocol, and rolling session ID’s have been implemented to prevent replay and roll-back attacks.

2.2 A Pilot Study

During the September to December 2003 Trimester at the University of Ottawa, we conducted a pilot study to use TellTable instead of a spreadsheet on a local computer for recording student marks from two sets of courses involving multiple professors and teaching assistants (TAs), in both English and French (Adler and Nash, 2004).

The principal finding of this pilot study has been that the TAs found it relatively easy to use, even over a dial-up line. Since TellTable runs OpenOffice.org applications on a server, transmitting the “screen” via a web-browser using VNC (Virtual Network Computing) technology, the satisfactory operation over dial-up lines was a pleasant surprise. The main complaint, which we are addressing, is that we made the server “too secure”. When one user has a file open, other users are locked out. Should the first user not log out properly, the file remains locked. We are addressing this issue on a technical level with an automatic logout on non-activity or other events that indicate the session is over. However, there are decisions concerning the definition of an “abandoned” session and whether to implement time limits on sessions that are socio-political in nature. Our goal is to provide technical tools that enable managers to implement locally established policies.

There were also some complaints of failure that were the result of bugs in our code. At the time of writing these have been largely overcome. A final complaint was the lack of a cut-and-paste feature to allow external information to be copied into our files. This is discussed below.

2.3 GTEC Exhibition

In early October 2003, we were invited to participate in the Open Source Lab of the Government Technical Exhibition (GTEC) in Ottawa (MediaLive International, 2003) This was the impetus for choosing a new name for our system.

2.4 A Project Name

The working name for our project was SSScan, short for SpreadSheet Scanner. Given the extensions to the original ideas and the near unpronounceable short form, GTEC prompted us to think of a new name. Several candidates were considered, but TellTable fitted well with our feelings about the project. Also, it seemed a good fit in a bilingual environment where “tableur” is
the term for “spreadsheet” in French. The domain name was registered and a web-site was set up in quick succession. Figure 1 shows the logo. The web address is www.telltable.com (17:30 31/03/2004).

Figure 1. The TellTable logo.

2.5 Other Office-Suite Applications

During June of 2003, one of us (JN) had been working on some tools for Web Content Management. Another (AA) suggested talking to Dr. Sylvie Noël of the Communications Research Centre of Industry Canada, who was also working on tools for collaborative authoring of text-based documents. In mid-October 2003, recalling this discussion, Nash realized that the server structure could collaboratively operate non-spreadsheet functions of the OpenOffice.org suite. This led very rapidly to a joint paper submitted to a special issue of the IEEE Journal on Professional Communications (Adler, Nash, Noël, 2004). This paper, on software for collaborative authoring of documents, spreadsheets, presentations and drawings, was prepared using TellTable and thus a test of the ideas it discussed, in the process revealing some further strengths and weaknesses of the TellTable approach and implementation.

Again, dial-up access proved acceptable for viewing the file(s) or making small changes, though clearly less efficient than broadband access. The OpenOffice.org writer assigned different colours to the changes made by the different authors, making it very easy to see how we had edited the document.

On the negative side, the system does not properly allow cut-and-paste from existing files local to the user's machine into the server based objects, that is, text, spreadsheets, drawings or presentations. This problem is inherent in the use of a Java applet in an Internet browser-based client. The security model of the Java virtual machine does not allow cut and paste interaction with the local machine. This problem comes from the use of a web-browser interface, and is more general than our own software. However, it has particular implications for setting up documents or other files for use, for example, in providing the base class-list files for TAs to use for marks, or for resetting a file should an older version be needed. There are several possible approaches to solve this issue. One would be using a technology such as ActiveX to embed the applet into the Internet browser, which does not have the restrictions of the Java applet. Another approach, which we hope to implement shortly, allows files to be imported into a special paste window that is accessible to the browser screen. Although this is not quite the usual cut-and-paste protocol, it will allow material, such as bibliographic references, to be imported from existing files. Indeed it was the bibliographic reference task that brought home this deficiency.

So far, we have fairly extensive tests of the spreadsheet and word-processing functions, and have verified that slide presentations and drawings can be accessed. We will need “real” users to fully evaluate the latter two functions and any other features of OpenOffice that are or may become available.
2.6 The telltable-s Open Source Project

Early in January 2004, we established an open-source software project within the SourceForge framework to develop and enhance the server infrastructure. This project is called telltable-s (all lower case) and can be accessed via sourceforge.net, in particular at telltable-s.sf.net. The software can be downloaded using a CVS client (Cederqvist, 2002) and is licensed under the GNU Lesser General Public License. That is, anyone wishing to set up a TellTable server is at liberty to download the software and install it. There are many details in doing this that may make this tedious or difficult in some Linux/Unix distributions, and we anticipate some potential users will choose to engage us or other knowledgeable workers to help them. Furthermore, we hope that a growing community of users will encourage the development of tools to assist installation and maintenance of TellTable servers, as well as enlarge the pool of possible applications such servers can support.

In contrast, the TellTable Analyse spreadsheet audit trail analysis program is at this time considered proprietary software which we are distributing under a shared-source model. TellTable Analyse is not part of the telltable-s server project, though, as the next item describes, it can be conveniently run under the server infrastructure. Users who purchase a license for TellTable Analyse will get a copy of the source and run-time versions of the program, but may only use the program under agreed terms and conditions, and may not pass the program to others. We anticipate that there will be considerable interest in customized versions of TellTable Analyse, rather than a generic program. So far we have not begun developing audit trail tools for OpenOffice.org applications other than the spreadsheet processor.

While the primary feature of telltable-s is the server-based access to OpenOffice functions, there is no fundamental technical obstacle to running other software through the server, thereby allowing better control and management of important organizational files. While OpenOffice can read and write most files of type .doc, .xls, and .ppt, these are not the native format and we have some reservations about their suitability for audit. However, for situations where strict formatting of output is required, as in some desktop publishing requirements, it would be better to use the Microsoft Office tools. A technical objection to this – that Microsoft Office does not run under Linux – can be answered with the CodeWeavers CrossOver Office tools. We have performed a very cursory test that Microsoft Excel could be executed through the TellTable-s infrastructure, and could see no obvious failings. Whether Microsoft's EULA (End User License Agreement) permits or by legal action can be required to permit such a usage is still an open question. Nor are we certain how well we could control the change recording of Microsoft Office applications. Clearly it is of interest to us to resolve technical and legal issues so that TellTable can have the widest possible utility. We welcome an exchange of information with other workers, and EuSpRIG is clearly the premiere forum for such exchanges.

2.7 Workflow Tool Integration

Discussions with users and potential clients have reinforced our belief that TellTable capabilities should be well-integrated with workflow requirements. Recently we re-designed the user screens and incorporated a cleaner display where users can, according to their privileges, edit, view, audit or download the files to which they are permitted access. Some users may be granted only view or audit access, with no edit privileges. “View” presents the spreadsheet in the usual way, but does not allow changes to be made. “Audit” calls up the TellTable Analyse program that we described last year (Nash, Smith, and Adler, 2003). This provides ways to look at the detailed change record of the spreadsheet file, including filters to assist the user to focus on particular types of changes. As TellTable Analyse is written in Java, it can be easily run on the server or on the client machine. However, a single click implementation saves much of the work of downloading the file, launching TellTable Analyse, and finding and loading the correct version of the downloaded file.
We have also simplified and enhanced the version history interface, so that users with administrative privileges may download, view or audit any of the versions of a file. These privileged users are able to insert files into the repository, or update a file with a version edited outside TellTable. While this means that such a user could remove the version history held within a spreadsheet file, TellTable maintains a read-only repository of all previous versions. As such administrative users are able to make arbitrary updates to the current file version, but a complete history of the modifications is still maintained.

TellTable Analyse has also been tested to show that it properly reports the effects of move or copy operations. OpenOffice **calc** keeps this audit information at the cell level, and visualization mechanisms to reconstruct the block operations are a feature we will have to keep in mind.

The single-click functionality can be enhanced by adding status and approval tags to files so that their disposition can be tracked and managed. This is very much in the spirit of management of the improvement of quality and productivity where we already have interests (Nash and Nash, 1997, 1998) under the rubric “Visible Management”. The essential idea of Visible Management is to make visible the state of work processes. Clearly many workers are active in this area, as a Google search of “Business Processes” will quickly reveal. So far, our view is that it is easier to carry out office work processes than to make it clear that they have been done, and done correctly. We will be kept occupied for some time with incorporating such tools into TellTable.

2.8 Applications in Education

Two of us are academics who teach as well as carry out research. Therefore applications of a spreadsheet audit trail in educational environments were an obvious subject of interest, and we prepared a paper (Adler and Nash, 2004) for the electronic journal Spreadsheets in Education. The “how” in that paper is similar to the content of this paper, but the “what” and “why” are different.

3. TELLTABLE DESIGN AND FUNCTION

This section describes the design and architecture of TellTable, which has changed somewhat since 2003. TellTable can reside on one (or more) servers. Server functions are split into components running within the Apache web server and other processes running with limited privileges. For applications with many users and large spreadsheets, it is possible to spread the TellTable server components over several machines. Users connect to the server with a standard Internet browser which supports Java applets. Currently, TellTable only runs on Linux, but the architecture supports any UNIX platform. We have successfully tested TellTable user access with the Internet Explorer, Mozilla, Konqueror and Opera browsers running Windows 98/2000/XP, Linux, and Mac OS X, though we have noted that some versions do not give satisfactory operation.

The user types the URL of the server into the browser, or clicks on an appropriate pre-prepared link, and is presented with a standard login screen requiring a username and password. Once logged in, the user sees a list of files which he/she has permission to view or edit. At this point, the user is interacting with the web server components of TellTable; these manage the user login and the state of the user session, as well as spreadsheet file versions. Since web interactions themselves are stateless, user state is maintained using a randomly generated session ID in order to prevent out of sequence interactions with any TellTable application. For security the HTTPS web protocol is used.

Upon choosing to edit a file, the user sees a browser screen within which a Java applet presents a view of the spreadsheet running on the server. All keyboard and mouse activity in the browser is
sent to the server to be processed, and the resulting spreadsheet view is sent back to the browser to be displayed. As indicated, we use the Virtual Network Computing (VNC) protocol to handle the display and interaction (Richardson, 2004). Some details follow.

When the server is started, a pool of limited privilege users (suids) are created and initialized to export their display using the VNC protocol, and to wait for information to arrive in an input directory. The VNC protocol allows a desktop display and keyboard and mouse activity to be exported over the Internet, using a low bandwidth protocol. Various clients have been developed to connect to a VNC exported display, including the Java applet used in TellTable. When a TellTable user selects a file to edit, the web server first verifies that a suid is available, and then copies the selected file, along with the information about the username and a randomly generated one time password into the input directory of the suid. The suid will then move the spreadsheet file to a processing directory, modify its configuration to match that of the given username, and open the file with the Openoffice.org calc application into the desktop exported by VNC. In this way, we ensure that OpenOffice.org will record change information under the correct username. The web page sent to the user contains information to load the appropriate VNC viewer Java applet and to connect to the appropriate suid. When the user is finished editing the file and closes OpenOffice.org calc, the suid will move the edited file to its output directory, where it is further managed by the web server. For security, the VNC password is modified to a newly generated and encrypted password, while this password is also sent to the user's browser with the applet information so the file and user can be kept linked. We are aware that allowing network access to server user account creates potential security vulnerabilities. To address these issues, we limit the privileges of the suid in terms of its ability to access files on the server outside of its local directories.

After editing the file, the user has the option to save it or discard the changes, closes the Openoffice.org calc application, and then clicks on a web link in the browser page to return to the TellTable main menu. The edited file is tested for modifications, and is stored in the TellTable version repository. If a user wishes to view previous file versions, a version history page is available. Each previous version, including the name of the user who made the last edits and the time of saving is available to be viewed, audited, or downloaded. We do not permit editing of previous versions, as the semantics and workflow of such an edit are unclear. File versioning is managed using the CVS version control system. CVS software allows concurrent editing of files; however, such concurrent editing requires the ability to merge conflicting edits. As we are not aware of a robust approach to manage conflicts in spreadsheet files, we do not permit this within TellTable.

As indicated, the TellTable framework is flexible enough to allow diverse software to be run to perform different operations on spreadsheets and other office-suite files. For example, the TellTable Analyse spreadsheet audit software allows analysis of changes recorded in the spreadsheet files. We envisage other applications, both open-source and proprietary, being used within this framework as a way to enhance workflow management and efficiency.

4. PERFORMANCE TESTING

TellTable shifts much of the computational burden from the local machine to the server. In this section we report some tests to measure the relative loads of OpenOffice.org calc on the client in comparison to the TellTable server. Tests were performed using an Intel Pentium 4, 2.4 GHz server with 256 MB Ram running Mandrake Linux 9.1 within the VM Ware software environment. Client connections were made from an identical computer running windows XP using Internet Explorer with Sun Java 1.4.

4.1 Memory Test
The goal of this test was to measure the memory required by TellTable as a function of the number of users supported. The TellTable server was configured to support $N_{suid}$ processes. Initially, the memory requirements were measured without users editing spreadsheets. Subsequently, users were logged into all $suid$ processes to edit a simple spreadsheet. The total “committed” server memory was measured for $N=2, 4$ and $6$, using the UNIX command “top”. Measured memory consumption values were fitted to a line as:

$$151 + 5.8 \text{ (available } suid) + 17.0 \text{ (used } suid) \text{ MB}$$

This result indicates that the server memory requirements are relatively small compared to that required by the operating system and the spreadsheet data itself.

### 4.2 Timing Test

The goal of this test was to compare the time to carry out spreadsheet calculations within the TellTable framework versus similar calculations on a local machine. Tests were designed to perform a large computational load without excessive memory demands. The test spreadsheet is described in the appendix. For our tests we chose a spreadsheet size of 50x50. Results follow:

<table>
<thead>
<tr>
<th></th>
<th>TellTable</th>
<th>Local machine (Win XP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 user</td>
<td>81.5s</td>
<td>79.5s</td>
</tr>
<tr>
<td>2 users</td>
<td>165s</td>
<td>N/A</td>
</tr>
</tbody>
</table>

These results indicate a slight (2%) reduction in performance under TellTable, and another slight (1%) reduction in performance due to simultaneous use. However, these performance decreases appear to be acceptably small, especially given the benefit of an assured audit trail.

### 5. WORK IN PROGRESS

As the time of writing (end of March, 2004), we are testing the SourceForge version of telltable-s on a system running the Xandros Linux distribution (version 2.0) which is built upon the Debian distribution and package manager (www.debian.org 17:30 31/03/2004). Our pilot study ran on a system that used the Mandrake distribution (version 9.1), and there are a number of minor but annoying changes that need to be made to account for differences in file system layouts. We hope to build both documentation and scripted customization to allow the installation of telltable-s to be very simple. At the present time we do not have plans to port telltable-s to non-Linux servers, but we would provide what advice we can to anyone wishing to do so in exchange for learning what issues arise. In principle, such a port should not be difficult.

We are also working toward the development of a "live CD" demonstration tool based on the popular Knoppix version of Linux. Knoppix is also based upon Debian and there are a number of derivative systems. See www.knoppix.org (17:30 31/03/2004). The advantage of such CD-based tools is that they do not require installation to be tested and run, yet provide all the components of the operating system and supporting software that are needed, and provide them in a coherent and well-organized form. The user of most Intel or AMD PCs simply puts the CD in his/her computer and “reboots”. After the session is over, the CD is ejected and the user restarts the machine to
return to the installed operating system, which is not altered by the Knoppix software apart perhaps from a single “swapfile” to provide for virtual memory. (This file is only created with user permission, is well-identified and may be deleted afterwards.)

There are some limitations to the use of Java applets in the client browsers. While almost all modern browsers support Java applets, most do not install with the Java tools, but require a subsequent download. Examples include recent releases of Internet Explorer and Mozilla. Another difficulty is that the Java applet security model does not support access to the local machine clipboard, preventing the use of cut-and-paste from the local machine to TellTable. Since Internet Explorer has built-in support for ActiveX plugins, we are exploring the use of ActiveX applets for VNC. Fortunately, the VNC protocol is well supported, and others have built such support (e.g., http://www.veridicus.com/tummy/programming/vncx/ 23:20 08/04/2004). Modifications to the TellTable server are minor, involving modification to the web page that loads the VNC viewer applet. One remaining difficulty concerns testing (from the server) that the user's browser is capable of running various applet plugins.

6. CONCLUSION

What we now call the TellTable project only began in November 2002. Since that time we have managed a proof of concept of a spreadsheet audit trail system, run a pilot study, demonstrated and used a collaborative office suite and prepared a number of academic and professional papers.

This progress confirms our belief that the TellTable framework is a valuable approach to managing some of the risks in spreadsheet design. From the point of view of the enterprise, TellTable allows the server administrator to control the versions and audit history of spreadsheet files. This capability is critical to any organization because of the value of the information that is stored in spreadsheet form. In our own experience, the ability to quickly access previous file versions has become an indispensable tool. Furthermore, the TellTable approach avoids the problem of inflexible proprietary applications, usually built around database technology, by providing the flexibility of full spreadsheet capabilities. Anecdotal evidence suggests that rigid centralized applications are left largely unused because they don't match user requirements, while the “real” data is entered into spreadsheets, sometimes referred to as “black book” applications.

From both the individual and the organizational perspective, the capability of TellTable to provide an audit trail to spreadsheet changes and additions allows for the discovery and correction of errors or falsification of data, particularly when the audit information can be filtered and summarized by TellTable Analyse. This capability, which can be extended to other office-suite files, is believed to be unique at the time of writing.

7. ACKNOWLEDGEMENTS

We appreciate the support of Joseph Potvin and Marcel Boulianne of Public Works and Government Services Canada, and for inviting us to participate in GTEC 2003, and for help in correcting translations of TellTable information. Mary Nash helped edit this paper.

8. APPENDIX

This section describes the high-computational-load spreadsheet test. The tests in this paper are based on a spreadsheet that requires significant computations, without the need for very large spreadsheet files which would, instead, be a test of the physical and virtual memory systems.
Since we were unable to find such a test in the literature, a custom test spreadsheet was designed. We describe it here because it may constitute a useful benchmark. See the telltable-s distribution files

The test spreadsheets are built up with a border (left and top) that is two cells thick, and a main section which is cell C3 and all cells below and right of it. Cell B2 contains an arbitrary number, and recalculation is forced by changing its value. The "border" is built up of number increments and trigonometric functions such that the values are in a reasonable range to help ensure the matrix inverses described below are numerically stable. Each cell in the "main" section of the spreadsheet is the upper left or 1,1 element of the matrix inverse of the largest available square block of cells to the upper left of that cell. The test is then performed by adjusting the value of the arbitrary number in cell B2, and waiting for the spreadsheet to recalculate. The size of the spreadsheet is determined by the number of elements in column A and row 1 to the right and below cell C3. The spreadsheet structure is shown in the following table:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>( =B1+1 )</td>
<td>( =C1+1 )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Arbitrary number</td>
<td>( =\sin( B2+B1 ) )</td>
<td>( =\sin( C2+C1 ) )</td>
</tr>
<tr>
<td>3</td>
<td>( =A2+1 )</td>
<td>( =\cos( B2+A2 ) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( =A3+1 )</td>
<td></td>
<td>( =\cos( B3+A3 ) )</td>
<td></td>
</tr>
</tbody>
</table>

where element C3 in the **Main Section** is defined as

\[
=\text{INDEX(}
\text{MINVERSE(}
\text{OFFSET(}
$A$1;
1 + \text{MAX( 0; A3 - C$1 \) ;}
1 + \text{MAX( 0; C$1 - A3 \) ;}
\text{MIN( C$1;A3 \) ;}
\text{MIN( C$1;A3 \) ) ;}
1;
1)
\text{)}
\]

This cell formula is then replicated for all cells below and right of C3. This spreadsheet has been tested on the spreadsheets OpenOffice.org *calc*, Microsoft Excel, and Gnumeric. Note that all semicolons (;) must be replaced by commas (,) for Excel and Gnumeric. We have not been able to make this spreadsheet work properly in Quattro Pro (version 9 was tried).

This spreadsheet was tested on a Pentium 4, 2.4 GHz computer running Windows XP for spreadsheets of size 30x30, 35x35, 40x40, 45x45, and 50x50, using OpenOffice.org *calc* version 1.1. Recalculation times for each size were 3.5, 7.5, 21, 44 and 80s, respectively. The maximum
memory consumption of the spreadsheet software was 32.4, 33.2, 33.8, 34.4, and 35.3 MB, respectively. These data can be modelled as:

\[
\text{recalculation time} = \left(\frac{\text{size}}{24.7}\right)^{6.3} \text{ sec.}
\]

and

\[
\text{maximum memory} = 28.2 + 0.14 (\text{size}) \text{ MB}
\]

Thus, this spreadsheet shows strong exponential growth in calculation time, for linear growth in memory consumption.

We have also built a moderately computationally intensive spreadsheet that allows us to test the time/memory relationship for very large spreadsheets. With this we can explore situations where the spreadsheet file cannot be stored entirely in memory, but must be swapped out to disk (virtual memory). This test is based on one in Nash and Nash (1994, p. 67). It uses a first row (Row 3) where cell A3 is set to 1, and cell B3 is set =A3+1, and so on to increment each element of the third row by 1. Cell A4 is set =A3+2, so that the first column increments each row by 2. We can make this border have as many rows and columns as we like, then fill in the rectangle with top-left cell B4 do the maximum settings with cell i, j computing the function

\[
\exp(\sin(\cos(\text{rowdata} + \text{columndata})))
\]

That is, cell E7 is set =\(\exp(\sin(\cos(A7+E3)))\). We can make the core function more or less complicated as needed for our tests, but must note that this spreadsheet is less time-consuming than the one above. It is available as [http://cvs.sourceforge.net/viewcvs.py/telltable-s/telltable-server/repository/files/timetest.sxc](http://cvs.sourceforge.net/viewcvs.py/telltable-s/telltable-server/repository/files/timetest.sxc) (2004-6-4 15:30).

Our objective in building these tests was to see how badly the TellTable server would degrade when running more than one compute-bound spreadsheets. Very large spreadsheets may also run slowly due to the use of swap memory on disk, which is an artifact of virtual memory subsystems.

9. REFERENCES


Narrative Text: TellTable Spreadsheet Audit: from technical possibility to operating prototype

John Nash, Andy Adler and Neil Smith

Free Software Foundation, Inc. (1999) The GNU JAXP Project


Identification of logical errors through Monte Carlo simulation

Hilary L. Emmett, Decisioneering (UK) Ltd.
Broadgate Court, 199 Bishopsgate, London EC2M 3TY, United Kingdom
hemmett@crystalball.com

Lawrence I. Goldman, Decisioneering Inc.
1515 Arapahoe Street, Suite 1311, Denver, CO 80202, U.S.A.
lgoldman@crystalball.com

ABSTRACT

The primary focus of Monte Carlo simulation is to identify and quantify risk related to uncertainty and variability in spreadsheet model inputs. The stress of Monte Carlo simulation often reveals logical errors in the underlying spreadsheet that might be overlooked during day-to-day use or traditional “what if” testing. This secondary benefit of simulation requires a trained eye to recognize warning signs of poor model construction.

1 INTRODUCTION

1.1 Types of Spreadsheet Risk

Risk is commonly defined as the probability of loss, damage, or any other undesirable event. For spreadsheet users, risk is the probability of the spreadsheet model yielding an incorrect output. Here, we will divide the sources of risk into two major categories, those due to incorrect inputs, and those due to faults in the mathematical and logical formulas within the model.

When using spreadsheets, analysts traditionally input average or best-guess values for uncertain variables because programs like Microsoft® Excel only allow them to enter a single value or formula in a cell. These “deterministic,” or known, models provide a single outcome upon which a business or technical decision is made. To capture uncertainty, analysts can perform simple “what if analysis” or “scenario analysis” by manually changing model variables and analyzing their effect on the key outputs. This approach provides a range of possible outcomes but does not impart an understanding of the likelihood of any particular outcome. There are simply too many combinations of input values to calculate every possible result.

1.2 Monte Carlo Simulation

One established solution to the limitations of spreadsheet risk analysis is Monte Carlo simulation. Since Excel alone does not have the ability to run and analyze simulations, modelers must rely on third-party programs like Crystal Ball® that add in and expand the features of Excel. Crystal Ball adds two techniques to Excel: the replacement of single values with probability distributions and the random simulation of a model. The result is a probability-based spreadsheet with quantifiable outcomes, such as a 75% probability of staying under budget or a 90% certainty of 100 million barrels of oil within a geologic reservoir. This example focuses on the probability of success for a simple project, based on the uncertainty in future sales, cost of goods sold, and operating expenses. Excel’s NPV function is used to calculate the present value of the project.

As an auditing tool, Monte Carlo simulation tests data outside the limits of the normal range (distribution tails.) In a survey of Australian spreadsheet developers, 63% acknowledged this as something they should be doing, while only 33% actually reported using the technique. [Hall, 1996]
1.3 Garbage In, Garbage Out

One of the major criticisms of Monte Carlo simulation is that it can provide a false sense of confidence in cases where the underlying spreadsheet logic is incorrect. This problem is prevalent in Excel-based tools, where the stochastic results are only as good as the skill, accuracy, and discipline of the spreadsheet developer.

There are, however, several check points where the output of a simulation can reveal formula errors that had been overlooked in the process of creating the spreadsheet. These check points occur during and after the simulation and through the analysis of several different types of charts and statistical methods.

Following recent accounting scandals in the United States, the Securities and Exchange Commission has mandated some form of risk analysis for all publicly traded companies. Similarly, the Financial Accounting Standards Board now requires reporting on the value of employee stock options. A number of companies are now implementing Monte Carlo simulation in an attempt to comply with these standards.

In comparison, formal spreadsheet application review and audit mandates are estimated to exist in as few as 10% of companies. [Hall, 1996] Any attempt to formalize the spreadsheet development process can be seen as an improvement. Model verification through simulation is not intended as a substitute for a structured development and review of all model logic, but rather as a final reality check.

2 TOOLS FOR RELATING RISK INTO ACTION

2.1 Modeling Input Risk

The first step in the move from deterministic to stochastic modeling is to identify key input variables that are subject to uncertainty. We will refer to these inputs as assumptions. For each assumption, the user must select an appropriate distribution to describe the possible variation. Crystal Ball provides 16 standard distributions and a custom distribution to characterize the behavior of the assumption variable. Distribution selection should be based first on any available historical data. In the absence of historical data, one must rely on the underlying physics of the quantity to be described or select a simple distribution such as the triangular distribution and apply reasonable limits. Expert interviews and data gathering can provide critical information to assist in this process.

![Figure 1: Crystal Ball Distribution Gallery](image)

2.2 The Monte Carlo Technique

Monte Carlo simulation is a proven, efficient technique that only requires a random number table or a random number generator on a computer. A random number is a mathematically selected value that is generated to conform to a probability distribution. Monte Carlo simulation was named for Monte Carlo, Monaco, where the primary attractions are casinos containing games of chance, such as roulette wheels, dice, and slot machines, that exhibit random behavior.
The random behavior in games of chance is similar to how Monte Carlo simulation selects variable values at random to simulate a model. When you roll a die, you know that a 1, 2, 3, 4, 5, or 6 will come up, but you don't know which for any particular roll. It's the same with the variables that have a known range of values but an uncertain value for any particular time or event (e.g., interest rates, staffing needs, stock prices, inventory, phone calls per minute). The multiple scenarios created through simulation can be analyzed to give more insight into the risks and mechanisms of the spreadsheet model. When used correctly, Monte Carlo simulation can provide valuable insights not available through deterministic models.

2.3 Interpretation of Results

The Simulation Process

Spreadsheets provide a calculation engine for the model while Crystal Ball repeatedly samples from the input distributions. As the simulation progresses, key formulas identified by the user as forecasts are captured in memory for further analysis. For each of these forecast variables, there are two primary outputs from Crystal Ball, the forecast chart and the sensitivity chart. To understand how these charts can help identify potential faults in the model logic, it is necessary to first examine their relevance for risk quantification.

Forecast Charts

The forecast chart is a histogram that displays the range and frequency of different outcomes for an individual forecast. This information is used to calculate statistics about the probability or likelihood of different outcomes. For instance, one could calculate the probability of a financial loss occurring on a particular project where future revenue is uncertain. In addition to the graphical view, a full set of descriptive statistics is available for each simulation.

Sensitivity Charts

The second chart of importance is the sensitivity chart. Sensitivity charts display the relative relationship of an output forecast and each of the random assumption variables. This is represented by a correlation coefficient that measures the strength of the linear relationship between any two uncertain variables.

While the intent of sensitivity charts is to help identify key sources of variation, it is also an opportunity to perform a reality check. Seldom does a spreadsheet user attempt to model a problem without some
knowledge of what factors are mission critical. In much the same way, most modelers have an intuitive understanding of the sort of results to expect from a particular forecast, making these two charts valuable in the identification of formula errors.

![Sensitivity Chart](image)

**Figure 3: Typical sensitivity chart results**

### 3 WARNING SIGNS

#### 3.1 Errors During the Simulation Cycle

Before running the simulation, you can check for logic errors by clicking on the Single Step button on the tool bar (Figure 4). Each single step is a random trial in Crystal Ball. In one step, Crystal Ball generates a random number for each assumption, and Excel automatically recalculates the model. As the model recalculates for each trial, you may discover calculation errors due to unanticipated alternative scenarios.

![Tool Bar Buttons](image)

**Figure 4: Tool Bar Buttons: (1) Define Assumption (2) Define Forecast (3) Run Preferences (4) Run (5) Stop (6) Single Step (7) Forecast Windows (8) Sensitivity Analysis (9) Create Report (10) Help**

By default, a simulation stops whenever a calculation error occurs in a forecast cell. In this case a calculation error is defined as any formula which can not be resolved such as the square root of a negative number, division by zero, a bad look up table, or a financial function that does not converge due to a poor guess for an argument (for example IRR.) If this error occurs it identifies the location of the forecast cell and retains the underlying assumption values that lead to the calculation error.
Oftentimes, calculation errors are expected for certain financial calculations such as an Internal Rate of Return with a negative series of cash flows. In such cases, you can set the simulation to discard all trials that resulted in calculation errors. This adjustment, however, should only be done after careful consideration and ensuring that the calculation error is not due to some underlying flaw in the spreadsheet logic.

3.2 Irregular Forecast Results

No Variation in Target Forecast

One common cause of errors is the unintentional removal of formulas. Simulation add-ins often conflict with spreadsheet logic protection schemes. When cells are left unlocked, you are exposed to the risk of this innocent mistake. With complex logic spanning several sheets within the workbook, even experienced modelers can fall victim. Figure 6, which shows a forecast chart with a single value rather than a range of values, is a red flag that the logic chain may have been broken.

Out-of-Range Forecasts

The forecast chart can also point to potential logical flaws by identifying out-of-range forecasts. While the root cause of the problem may be obscured, the results should be fairly obvious. The warning sign here is
values that lie outside of theoretical limits. For example, some cash flow models must account for the fact that once a balance reaches zero, it must remain at zero for the remainder of the time horizon in the analysis. While the problem is relatively easy to fix from a formula standpoint, it is often overlooked because users seldom work with worst-case scenarios while building a deterministic model.

3.4 Scenario Analysis

The most effective way to understand forecasts outside the expected range is to find a set of assumption values that leads to the result in question. One method for accomplishing this is to apply the Scenario Analysis tool in Crystal Ball (Figure 7). First you specify a target range for the forecast, then Crystal Ball gathers all of the simulation trials that result in forecasts for that particular range. The tool also includes spreadsheet macros that paste the combination of inputs that led to a given output back into the original spreadsheet. This technique allows the user to investigate whether or not the forecast result is possible. On the other hand, if scenario analysis reveals a problem with the model logic, it is helpful to have a base case that replicates the error for auditing the formulas.

![Figure 7: Scenario Analysis Output](image)

3.5 Sensitivity Sanity Check

Results of the sensitivity chart should always be scrutinized to ensure that the results are consistent with the theoretical positive/negative relationship between an input assumption and the output forecast. Gut-feel instinct should not be ignored when the results of the sensitivity analysis conflict with prior experience. In Figure 8, the OPEX/Sales% appears to have the greatest effect on NPV, but experience suggests that Year 1 sales should have a much higher relative impact on the results of the simulation.
If the results of the sensitivity chart are counter intuitive, two actions should be taken. First, review the parameters for input distributions. Inconsistent entry of percentage values is a common cause of such errors. Next, if discrepancies remain following a review of the input parameters, a review of the logic will be necessary. The tornado chart and Excel’s auditing tools are useful for tracing formulas dependent on the input assumptions that appear to be under (or over) emphasized. In the case of the Year 1 sales, the discrepancy was traced to a formula that hard coded the Year 1 sales rather than referred to the input distribution (Figure 9).

At first glance, the positive correlation between Net Present Value and Cost of Goods Sold (COGS) growth rate shown in Figure 9 could be cause for concern. If this were interpreted strictly as a cause and effect relationship, that would suggest that increasing costs leads to a direct increase in profits. A savvy modeler would consider the possibility of a formula error in the calculation of Net Sales in Row 12. Looking carefully at the sensitivity chart, you will notice that the assumptions for Sales Growth Rate and COGS Growth Rate are marked as correlated assumptions. This underlying relationship between the variables is important, as a strong positive correlation could mask the underlying cause and effect created by the model logic.

3.6 Resolving Discrepancies Using the Tornado Chart

Tornado charts provide an alternative sensitivity technique to rank correlation when trying to determine whether the negative relationship between COGS Growth Rate and NPV is caused by the underlying correlated assumptions or by a formula error. Rather than simultaneous sampling of all assumption variables, tornado charts isolate and change each variable separately (Figure 10).
By isolating the variables we are able to see the direct impact of change and understand the underlying logic. In this example, we do not see any cause for doubting the model, an increase in COGS Growth rate causes a small decrease in Project NPV.

3.7 Self-fulfilling Prophecies

One must always be wary of using a gut-feel based approach to spreadsheet audits. Familiarity with the spreadsheet makes the developer a less than objective reviewer. In Ayalew Yirsaw’s Interval Based approach, this problem is addressed by the spreadsheet developer documenting expected intervals for output results to serve as a guide for an independent auditor. Another way to avoid such problems is to use a technique referred to as back-casting. Monte Carlo simulation can sample random historical data points to test the model performance with real data.

4 CONCLUSION

Performing simulation analysis for future outcomes can help spreadsheet modelers to reduce their risks and dramatically improve the quality of their decisions. By moving to a probabilistic approach to spreadsheet forecasting, analysts can better quantify the risks inherent in their models and gain insights not available through traditional deterministic approaches. The stress placed on the model in the simulation process exposes weaknesses in the formulas and provides clues for tracing breaks in the logical flow.

5 REFERENCES


End User Computer Applications -
Auditability and Other Benefits Derived from a Temporal Dimension

By Ralph Baxter, Copyright Cluster Seven Ltd 2004. All Rights Reserved.
ClusterSeven, Coppergate House, Brune St,
London EC1 7NJ
rbaxter@clusterseven.com

ABSTRACT

Locating changes to computer-based documents (as generated within Microsoft Office) is labour intensive. This is due to the lack of a permanent temporal record. Other problems include the lack of business context or data trend information or an ability to monitor document integrity. All lead to an absence of auditability and exposure to error or fraud. These problems have been solved by the creation of a structured history of all document/user interaction, creating a new approach to managing end user data. Built using Microsoft. Net and SQL Server database, the solution provides a complete monitoring and analysis service of changes occurring in end user documents.

The solution is a major step forward in reducing risk in the information supply chain (a pressing demand of investors and regulators alike), providing a wealth of previously inaccessible business intelligence in addition to full forensic auditability. The accompanying change in technology paradigm from control to monitoring opens the way for the future development of business systems to be lower cost and more agile.

1. INTRODUCTION

Section 2 of this paper summarizes the disadvantages of the current environment faced by Excel spreadsheet users with Section 3 exposing the comparative benefits of introducing a temporal history to sit behind Excel activity and Section 4 summarizing how this temporal history is created in the ClusterSeven solution. This leads to speculation in Section 5 as to the consequences for future systems development. Section 6 briefly describes the areas of spreadsheet risk not touched by the ClusterSeven solution.

2. CURRENT ENVIRONMENT

Most business users of Excel are familiar with the situation where the value of cell representing a key parameter (e.g. Profit and Loss or Value at Risk) moves outside of the anticipated range. The usual response is to initiate an enquiry into those data elements contributing to that parameter (i.e. in Excel language, the precedent causes to that cell) with a view to reconciling expectation against actual. This enquiry will be focused on locating the event that triggered the apparently anomalous result - with a view to determining whether it is valid, an error or occasionally something more serious. Achieving complete reconciliation can be extremely costly in terms of time and the seniority of personnel involved.
Similar challenges occur in the management of other desktop products (Word documents, Access databases).

The pressures for increasing speed and accuracy in corporate disclosure processes (from both investors and regulators) mean that better ways must be found to reduce the frequency of, and effort required to resolve, these enquiries. In seeking the answers during these enquiries few have time to stop and ask why they are so difficult they have become an expected cost of this form of end-user computing.

The underlying cause of the problem is not that complex - it is simply that these applications have no permanent memory of their development. Enter new data, change old data or change a formula and then save the result and it as if the past never existed. The only remotely accessible solution is by versioning the file periodically - a crude approach. Even where the organisation can depend on the diligence of the relevant authors (of whom there may be many, if linked documents exist) to follow strict version control procedures, identification of the relevant file and changes within it is extremely cumbersome.

Besides the difficulty of locating causal events the absence of temporal history leads to a suite of associated problems, some of which have been exploited to perpetrate fraud:

**Lack of business context:** such as change authorship or links to business workflow: desktop documents, such as spreadsheets, are inherently personal snapshots but their flexibility, low cost and ease of use has led to them becoming key elements of business critical processes. Business context can only be built externally (usually manually) to the document leading to gaps and inconsistency.

**Absence of trend information:** the time variation of parameters is a key tool for issue identification. Just the relative change from a previous value may be informative - but standard Excel only allows alerts to be linked to absolute limits. To date, more rigorous solutions could only be achieved by the manual rolling forward of historic values or the use of expensive pre-configured links into databases such as Hyperion. Derivatives of parameters such as variance and co-variance may well have provided further information but usually required too much effort to elicit. In all these cases the inability to refer easily to past content means that such analysis and monitoring is not possible.

**Changes in document integrity:** erroneous input, cutting and pasting or failed links to other documents can all break previously established integrity. The inability to compare today’s structure with the intact integrity of yesterday’s means that changes may be difficult to notice or locate.

**Absence of auditability:** all of the above lead to at best imperfect and, more usually, a complete lack of auditability. Even where an organisation is able to establish processes and controls to achieve some of the above these are likely to be highly dependent on the local integrity and diligence of those working with the relevant documents.

### 3. BENEFITS OF A TEMPORAL HISTORY

The introduction of a complete structured history for end user documents opens up a wide range of benefits to users and their managing organisation. These benefits have been confirmed by the responses from initial clients.

**Business context:** full history supports the identification and location of relevant changes and/or additions to data or functionality, showing which author made what amendments and with what consequences. Besides addressing changes in the content of spreadsheets it can also deliver understanding about the degree and nature of change activity in those documents, supporting workflow management and process integrity. For example one common fraud involves frequent minor
changes of static information. Though incremental changes may be insignificant, the frequency of change would be a clear warning.

**Document content:** the performance of content (e.g. cell values) over time is a critical starting point for the monitoring of relative changes, rates of change or other trend analysis, providing far more support for business analysis and monitoring than relying on absolute limits alone. In addition the existence of a full history of data and functionality, rather than a relying on a restricted, pre-determined selection means that the requirements of any analysis or audit are not compromised.

**Precedent causes:** the combination of full document histories plus precedent trees allows a complete forensic audit of any content change, with immediate access to the root cause, be it in the same document or a linked one. This allows any data change or addition to be interrogated immediately.

**Integrity of document infrastructure:** once the working structure of an end-user document oriented business environment is captured, changes to that structure that might corrupt output can be highlighted. For example in spreadsheets these may be the introduction of unauthorized links (as with Rusnak at Allied Irish Banks) or an erroneous cut and paste (as with Trans Alta in Canada).

**Auditability:** with all of the above in place, the requirements of both process and forensic audits are substantially supported - considerably mitigating the high risks of documents generated from end-user computer activity: manual entries, either for raw data input or top-level adjustments can be immediately identified (and linked to appropriate alerts as necessary); the time to compile data can be massively reduced, leaving valuable management time for analysis; and, with all relevant changes captured, previous versions can be recalled to meet any document retention requirements.

## 4. CREATION OF A TEMPORAL HISTORY

Through the development of XiGence, ClusterSeven have created a new approach to managing unstructured end user data. We have built a data discovery, observation, analysis, audit and reporting framework that addresses the major challenges associated with the use of Microsoft Office documents in the business.

The XiGence framework comprises a number of key components: XScan is a non-intrusive file scanner that is configured to scour corporate networks for instances of a chosen type of data file. XWatch is a passive file watching processor that is targeted by data from the XScan database to observe file events. These events trigger a data analysis and deconstruction process, XStore. XStore compares the internal data with the previous data set in the time series for that file. The changed data is stored and associated with the triggering event.

The XiGence framework is built using Microsoft’s .Net framework and SQL Server database. It has a hybrid service- and event-oriented architecture.

The combination of the two architectural models results in a very sophisticated data management and processing framework that is both fast and flexible. The framework makes use of the web services paradigm. The choice of web services was natural given the deployment, management and scalability characteristics of web services.
5. FUTURE PERSPECTIVE

Today’s spreadsheet development environment is commonly underpinned by the belief that any new spreadsheet is a ‘temporary’ solution. This culture means that the quality assurance processes deemed appropriate in other development environments are often missing. Now that the integrity of the mature spreadsheet application can be managed – and hence seen as a long-term solution – this opens up the opportunity to create more formal assurance processes e.g. the creation of a "Trusted Zone" for models / applications, akin to the "authorized move to a production environment" in mainframe computing.

Further out, the adoption of change monitoring and analysis service as part of the end user computing environment creates a powerful shift in the paradigm for enterprise technology. It is a major step in linking the innovation, speed and flexibility of end user computing activities with the enterprise requirements of transparency, corporate ownership and auditability.

Such a shift achieves an immediate primary aim of reducing risk in the information supply chain. However, the switch in technology emphasis from control to monitoring will enable a lower cost, but also more agile and evolutionary approach to the development of many future business systems.

6. LIMITATIONS

The ClusterSeven solution does not address the quality of the spreadsheet logic created by the user – the target of most other spreadsheet management applications. However, the real time monitoring of all cell events, whether against value limits or Excel error events means that user activity triggering these alerts can be rapidly identified. This allows immediate focus on the location of potential logic errors, thus supporting early correction.
A novel approach to formulae production
and overconfidence measurement
to reduce risk in spreadsheet modelling

S. Thorne, D. Ball, Z. Lawson
University of Wales Institute Cardiff

sthorne@uw.ac.uk, dball@uw.ac.uk, lawsonzf@cardiff.ac.uk

ABSTRACT

Research on formulae production in spreadsheets has established the practice as high risk yet unrecognised as such by industry. There are numerous software applications that are designed to audit formulae and find errors. However these are all post creation, designed to catch errors before the spreadsheet is deployed. As a general conclusion from EuSpRIG 2003 conference it was decided that the time has come to attempt novel solutions based on an understanding of human factors. Hence in this paper we examine one such possibility namely a novel example driven modelling approach. We discuss a control experiment that compares example driven modelling against traditional approaches over several progressively more difficult tests. The results are very interesting and certainly point to the value of further investigation of the example driven potential. Lastly we propose a method for statistically analysing the problem of overconfidence in spreadsheet modellers.

1. INTRODUCTION

In this paper we discuss one possible novel approach to formulae production and an overconfidence measurement to reduce risks in spreadsheet modelling. Our novel approach was the result of an inter-university collaborative project between UWIC and Cardiff Universities.

1.1 Spreadsheet errors

Since the widespread availability of the office PC there has been a rapid rise in End User Computing (EUC). EUC is a fairly broad term which covers database, presentation, word processing, spreadsheet and any other application that end users have access to. The activity of EUC was identified as presenting particular risks to organisations (Davis, 1987) (Alavi and Weiss, 1985) (Brown and Bostrom, 1989) (Munro et al, 1987) (Benson, 1983) (Brown and Bostrom, 1994) and (Alavi et al, 1987). Consequently, management strategies were formulated, designed to control EUC according to an organisation’s needs and culture (Brown and Bostrom, 1989) (Munro et al, 1987) and (Alavi et al, 1987).
Although spreadsheet modelling is incorporated into EUC management strategies, it requires special consideration due to the high flexibility, usability and availability that spreadsheets offer. For these reasons spreadsheets have become an indispensable tool to organisations for personal, strategic decision-making and even mission critical modelling. However, spreadsheet ease of use becomes problematic when organisations relying on spreadsheets in decision-making processes miss simple errors, which can then have serious repercussions. The EuSpRIG website (www.eusprig.org) has some such examples: ‘Cut and Paste error’ cost Trans Atlanta Corp. $24 million (June, 2003), Seattle self storage company shares fell 7.1% after it was revealed they had unintentionally overpaid two chief executives $700,000 each (October, 2003) and Florida education executives duplicated the cost of an Elementary school by $12 million, seriously effecting the projects budget (September, 2003).

1.2 Human factors

Research has suggested that human factors influence the quality of spreadsheet models. Panko (1998) suggested that error rates in spreadsheets are similar to other areas of complex human cognition. Kruck (1998) proved that spreadsheet model quality could be improved by equipping modellers with the cognitive skills that they require to produce complex spreadsheets.

All of this research establishes spreadsheet use coupled with human factors as a high-risk activity for strategic decision-making in organisations. In order to lower the risk, spreadsheets and modeller interactions need to be examined in great detail and new alternative methods of interaction developed. In this regard, Panko at the 2003 EuSpRIG conference strongly encouraged the need for new approaches based on human factors research.

1.3 Example driven modelling (EDM)

Considering the existing research into spreadsheet error production, it seems that most of the problems arise from human cognitive errors. Panko (1998) discusses a basic error rate even for simple tasks and an increased rate for more challenging tasks. This begs further research into the precise nature of the relationships involved. As discussed below our research resulted in a negative exponential function using Halstead’s ‘complexity’ measure (Halstead, 1977).

One interesting way to possibly improve the quality of spreadsheet models, taking account of the need for new approaches based upon human factor research, may be to eliminate the need for humans to produce formulae. Producing formulas is not what we are naturally best at (Michie, 1979). Human neural processes are mainly example driven and pattern matching. In contrast, computers are naturally good at arithmetic and logic (the Arithmetic Logic Unit is at the heart of a computer), but computers are not naturally good at generating examples or pattern matching. So perhaps at present we have things the wrong way round when producing spreadsheets, or at least not optimal.

For example, children learn arithmetic by example (Jon has four apples, Mary takes two. How many does Jon have left?), rather than solving: \(4 - 2 = 2\). When shopping at the supermarket, we know from previous examples when we are in danger of overspending. We rarely generate some formulae, which includes the effect of tomorrows interest rate rise on our credit card account, etc., and then use that formula to decide whether or not to buy those Garibaldi biscuits. Or again, we learn to catch a ball by example (i.e. practice) rather than dreaming up and solving the trajectory formulae. In this vein, we were interested in investigating whether it
may be easier to provide simple examples, which satisfy the problem rather than derive the requisite formulae.

### 1.4 Overconfidence

Panko (2003) extended his human factors research to measure overconfidence in modellers and the subsequent effect it has on the spreadsheet data integrity. Panko found that overconfidence in modellers ranged from 80% to 100%. These results show that overconfidence is a significant issue in spreadsheet modelling. Overconfident modellers may fail to apply any methodology or testing strategy and will not question the validity of their model. This practice is clearly counter-productive and contributes to the current poor integrity of spreadsheet models, hence our interest in establishing a satisfactory metric. Our results extend research on overconfidence measurement.

### 2. EXPLORING FORMULAE PRODUCTION

#### 2.1 Introduction

To examine how pervasive spreadsheet errors are in formulae production, in the face of increasing formula complexity, experiments were designed to establish error rates in both traditional formula production and example driven modelling (EDM). Traditional modelling was used as a control for the EDM experiments. The results of these different paradigms are then compared and analysed accordingly.

#### 2.2 Aim

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

1. The relationship between spreadsheet error rate and formula complexity using a) Traditional modelling, b) EDM
2. The (hypothesised) superiority of EDM over traditional modelling.
3. A more satisfactory statistical measure of overconfidence.

#### 2.3 Experiment design

##### 2.3.1 Introduction

This experiment was designed in accord with Campbell and Stanley (1963), which is considered a seminal text in Quasi and Experimental research design. Our experiment randomly selected a large target group (57 students) from a universe (of 2000 university students). The necessary experimental details were established using a pilot experiment using a smaller sample (12 students).

##### 2.3.2 Sampling

Participant selection is critical to the credibility of the experiment. In order to minimise bias of inexperience, certain courses were targeted. Considering similar studies: (Panko and Sprague, 1998); (Panko and Halveson, 1996); (Galletta et al, 1997); (Galletta et al, 1993); (Teo and Tan, 1997); (Panko and Halverson, 1997) and (Irons, 2003) these all used either undergraduate students or Masters level students as participants. For comparable results several different courses were targeted, which also maximised the number of results. The courses targeted were: Final year Undergraduate Business Information Systems and MSc Information Systems. These groups have been selected for their academic and industrial
The participant’s previous experience in spreadsheet development varies. All participants have, at some point, undertaken a module that focuses on EUC development packages (including spreadsheets with a specific assignment). Students are exposed to spreadsheets all through their university life, they are used in business type modules but also in statistics modules. It is a fair assumption that the participants also use spreadsheets outside of university life in some capacity. All participants would have at least a basic working knowledge of spreadsheets and would have been exposed to creating spreadsheet formulae. It is likely that the participants would have tackled similar problems. To further ground the experience level, all participants were given a brief lecture and document detailing the construct of various spreadsheet formulas.

Equal importance should be given to sample size, considering the studies mentioned above, the average number of participants was 52. In our case the number of participants was 57.

2.3.3 The Experiment

Students from the relevant groups were given a series of 5 tasks to complete using two different approaches. The tasks involved a ‘traditional’ approach of manually constructing spreadsheet formulae (serving as a control) and an EDM approach (where the participants were required to give example data for various attribute classifications).

The experiment was conducted in three stages. The first was to deliver a brief reminder lecture on how to use formulae in excel and some practical demonstrations. The second stage was a series of five tests involving formulae production. This was followed by a self-evaluation of perceived accuracy that was compared to the actual accuracy to measure over/under confidence. The third stage tested EDM by repeating the first five tests, but instead of creating formulae, the participants were required to give correct example data in accord with the model. This was again followed by their self-evaluation of perceived accuracy that was compared to the actual accuracy to measure over/under confidence. The tests in both cases were progressively more challenging, each building on the difficulty of the last.

2.3.4 Confidence questionnaire

As stated, once the questions were completed the participants were required to fill in a short self-evaluation of perceived accuracy questionnaire on ‘task completion’ and ‘confidence measure’. This measured how confident the participants were and was later compared with the actual result.

2.3.5 Measuring the dependent variable and error production metrics

For a spreadsheet formulae the dependant variable’s success or otherwise is determined when the formula that is produced has the correct syntax and has the correct cell referencing. To put it simply, where mistakes were not obvious, the formulae were inserted into a known populated spreadsheet to check their validity by inspection. EDM success was determined by inserting the answer values into known working formulae to see if the data was valid. The resulting formulae were examined and the ‘percentage of models with errors’ (Panko, 1998), ‘percentage accuracy’ and ‘task complexity’ deduced.

The questions were measured in terms of Halstead’s complexity (Halstead, 1977). Halstead was used because it is a generally accepted metric and uniquely not only considers software complexity but also within formulae production. Using Halstead’s complexity test (Halstead,
1977), the questions were given a value allowing relative comparison. Halstead’s test was originally developed as a means of determining the relative complexity of software and algorithms within software and formed part of a larger set of tests to determine volume, difficulty, effort and complexity. Halstead’s complexity formula, which is commonly referred to as Halstead’s complexity, is shown below. In spreadsheet formulae, the operands and operators are determined across the whole formulae. Operators include: IF; AND; NOT; AVERAGE; SUM; ; = etc. Operands are the cell references and numbers used in the formulae. The resulting complexity ranges from 0, being the most complex, to 2 being the least complex.

\[
\text{Complexity} = \frac{2 \times n1}{n2 \times N2}
\]

where

- \(n1\) = the number of distinct operators
- \(n2\) = the number of distinct operands
- \(N1\) = the total number of operators
- \(N2\) = the total number of operands

The percentage accuracy for the sample group was then plotted against the Halstead complexity.

### 2.3.6 Overconfidence measures

As there are no existing satisfactory statistical measures of overconfidence, mainly attributed to its relative newness in the field, a method has been created to achieve this goal. The ‘overconfidence ratio’ is a coefficient value ranging from 1 (worst) to 5 (perfect). This value matches the participants ‘combined overconfidence’ value to the actual result \(F(x)\) by questions 1 to 5.

The combined overconfidence is created by using the questionnaire results for confidence (1 to 5) and perceived difficulty (1 to 5). Hence:

\[
\text{Combined overconfidence} = (A \times \text{Confidence}) + (B \times \text{Difficulty})
\]

where \(A = B = 0.5\)

Define \(X\) as the number of errors, \(F(X) = \text{mark/actual result}\)

\[
F(x) = \begin{cases} 
5 & \text{if } x = 0 \\
4 & \text{if } x = 1 \\
3 & \text{if } x = 2 \\
2 & \text{if } x = 3 \\
1 & \text{if } x \geq 4 
\end{cases}
\]

\((F(x) = 0 \text{ if the question was not attempted})\)

\[
\text{Confidence ratio} = \frac{\text{Ratio perceived error rate}}{\text{Actual error rate}}
\]

In real terms this measures the expected outcome of the participants to the actual outcome and ranks it according to how accurate their prediction was, resulting in a value of 1, indicating a perfect match between expected and actual, to 5 indicating the worst match between expected...
and actual. It may also range below 1 to 1 fifth where greater than 1 implies over confidence and less than 1 implies under confidence.

2.3.7 Participant tasks

The participants were given 5 descriptive problems based around creating formulae to produce grades for a hypothetical set of marks (no actual data was included) for a university. It was then up to the participants to produce a formula that would solve the problem. The questions got progressively more complicated, requiring the participants to account for different factors in the formulae. For example, the first question required a formula that would output “pass” or “fail”. Later questions required the formula to output “Fail”, “Compensate”, “Pass”, “Merit” and “Distinction”. The participants were also required to use indirect referencing and variable grade boundaries as the questions progressed.

For example one possible solution to “Ensure both exam and coursework are above 40 and can have the classification: Fail (<40); Pass (>=40, <55); Merit (>=55, <70) or Distinction (>=70)” formulae is:

=IF(MIN(C5:D5)<40,"Fail",IF(AVERAGE(C5:D5)>=70,"Dist",IF(AVERAGE(C5:D5)>=55,"Merit",IF(AVERAGE(C5:D5)>=40,"Pass"))))

The traditional formulae success or otherwise was determined on whether the formula that was produced had the correct syntax and the correct cell referencing. To put it simply, where mistakes were not obvious, the formulae were inserted into a known populated spreadsheet to check their validity by inspection. EDM success was determined by inserting the answer values into known working formulae to see if the data was valid. The confidence questionnaire then followed this.

As the second part of the experiment, the participants were given the same descriptive problems, as questions 1 to 5, but instead of producing formulae, they were required to give example data for each classification. The second confidence questionnaire then followed.

2.3.8 Experimental details

Described below are the conditions and details for the experiment conducted in an academic arena.

Conditions

There was no conferring allowed between by participants; each test was unique to a participant. There was no time limit imposed on the test, the test did have to be completed in the presence of an examiner. The participants were not told the nature of the experiment.

Details

The participants were given two documents for the test. The first document was designed to accompany the brief reminder lecture given to the participants. The second document was the question, answer and confidence papers. The participants were talked through the documentation they had been given and were told where to write the answers and what to do with the papers when finished. They were then allowed to start the test.

3. ANALYSIS
The results that this experiment provided are broken down into two sections. The first deals with the accuracy of manually modelling formulae against the example driven approach and the second deals with overconfidence in modellers.

### 3.1 Traditional approach

The results from the traditional modelling show a high error rate. The average percentage of incorrect answers for questions 1 to 5 was 80%. The percentage of models with errors, Panko (1998) is therefore 80%. This high error rate is typical of similar studies such as: 100% (Hicks, 1995); 91% (Coopers and Lybrand, 1997); 91% (KPMG, 1997); 86% (Butler, 2000); 84 and 95% (Janvrin and Morrison, 1996 and 2000) and 80% (Panko and Halverson, 1997).

Figure 1 shows the percentage accuracy drops with increasing Halstead complexity, as the questions progress. The traditional method also yielded an average of 4 mistakes per question. EDM, in contrast gave an average of 0.3 mistakes per question.

![Figure 1](image1.png)

The curve in figure 1 appears to be exponentially decreasing; it does however demonstrate some strange behaviour at either extremity. We believe that the equation is trying to cater for the boundary conditions at either extremity, see figure 2.

![Figure 2](image2.png)
In the case of very low complexity, there are good reasons for believing that the percentage accuracy will never reach 100%. This upper boundary is due to the base error rate (Panko, 1998) and therefore a limitation occurs and the curve plateaus. See figure 3.

In the case of very high complexity, we believe that a similar limitation occurs and the curve becomes a vertical line. The reason for this kink is due to the well-known constraints of human working memory (Miller, 1956) which states that human memory (of the order of a few minutes) is restricted to handling ‘seven plus or minus two concepts simultaneously’. For example nested IF statements that include AND, averaged cells with indirect referencing is starting to get quite complex. When the number of concepts being handled simultaneously exceeds 9 (the Miller threshold), then unless there is some kind of spreadsheet engineering technique used, errors are almost certain. This point is crucially important when as far as mission critical spreadsheets are concerned. It would be very interesting to know what percentage of mission critical spreadsheets have formula beyond the Miller threshold, where also such organisations have no technique for coping beyond the Miller threshold, i.e. a spreadsheet modelling methodology or equivalent.

Figure 3

3.2 EDM

Figure 4 displays the results for EDM. As can be seen from the graph, the level of accuracy is very high and is easily able to cope with the most complex question. On average, 98% of all answers given were correct.
3.3 EDM and Traditional comparison

Figure 5 compares the difference between EDM and traditional methods. These results show a very significant improvement in accuracy when using an EDM approach. However, further research on EDM may well reveal new sources of error (Fraser and Smith, 1992).

![EDM versus Traditional](image)

**Figure 5**

3.4 Overconfidence

Figure 6 shows the results for the confidence metric. Clearly the traditional approach leads to overconfidence. Our method for measuring confidence is based on how well the participants match their actual and perceived accuracy in assessing the difficulty of the task. A confidence ratio of 1 means a perfect match. It would appear that, within experimental error, this is true for EDM responses because the line starts just below and ends just above the confidence ratio of 1.
A novel approach to formulae production and overconfidence measurement to reduce risk in spreadsheet modelling
S. Thorne, D. Ball, Z. Lawson

Figure 6

The questionnaire used in the experiment also allows relative comparison of perceived difficulty in the traditional and EDM approaches see figure 7. Note that, 1 corresponds to impossibly hard, 5 corresponds to trivially easy. The interesting point here is not the slow increase in perceived difficulty as the questions increase in complexity, since this was expected. Rather, the interesting point is the gap between the traditional and the EDM approaches. The gap corresponds to a significant usability advantage for EDM.

Figure 7

4 CONCLUSIONS

4.1 Validation of aims

The first aim was to establish the relationship between spreadsheet errors and formula complexity using traditional modelling and EDM. The relationship is a negative exponential but the model breaks down at both very high (Miller threshold) and very low (Base error rate) complexities.
The second aim was to consider whether EDM was superior to traditional modelling. As discussed above, the evidence certainly suggests this.

The third aim was to establish a more satisfactory measure of overconfidence. We feel our overconfidence measure has more transparency than existing methods. Further, relative comparisons between the traditional and EDM were more easily highlighted.

4.3 Limitations to the experiment

The experiment has been conducted in an academic environment; other environments also need to be considered. The EDM experiment suggests that there may be considerable merit in EDM being much more thoroughly investigated.

4.4 Further research

It would be very useful to verify the above findings within a real-life business-modelling environment. As discussed above it would be very interesting to assess the percentage of mission critical spreadsheets which contain formulae beyond the Miller threshold which do not make use of a rigorous spreadsheet engineering methodology or equivalent. Further research on EDM might include a machine to read in the examples and convert them to formulae. Many questions might be asked concerning the validity and application of such a machine, but this research points to possible advantages of such a novel advance.

References


Davis.B, 1987, ‘Commentary on Information systems’, Accounting horizons, 43


A novel approach to formulae production and overconfidence measurement to reduce risk in spreadsheet modelling

S. Thorne, D. Ball, Z. Lawson


Complexity Metrics for Spreadsheet Models

Andrej Bregar
University of Maribor
Faculty of Electrical Engineering and Computer Science
Smetanova 17, SI-2000 Maribor, Slovenia
Email: andrej.bregar@uni-mb.si

ABSTRACT

Several complexity metrics are described which are related to logic structure, data structure and size of spreadsheet models. They primarily concentrate on the dispersion of cell references and cell paths. Most metrics are newly defined, while some are adapted from traditional software engineering. Their purpose is the identification of cells which are liable to errors. In addition, they can be used to estimate the values of dependent process metrics, such as the development duration and effort, and especially to adjust the cell error rate in accordance with the contents of each individual cell, in order to accurately assess the reliability of a model. Finally, two conceptual constructs – the reference branching condition cell and the condition block – are discussed, aiming at improving the reliability, modifiability, auditability and comprehensibility of logical tests.

1 INTRODUCTION

Many spreadsheet models are large and complex. Because they are rarely built according to formal software analysis and design strategies, and because cell formulas are hidden, most of these models contain serious errors. Several research studies of fault rates have been performed in the past [Panko, 2000; Panko and Sprague, 1998], indicating that 20 to 60 percent of spreadsheets produce wrong outputs. For this reason, minimization of errors through proper design, quality assurance and systematic cell inspection is crucial. Some structured development/modelling methods have already been applied in order to reduce risks [Conway and Ragsdale, 1997; Janvrin and Morrison, 2000; Kreie et al., 2000; Read and Batson, 1999; Ronen et al., 1989]. Moreover, useful auditing techniques and tools exist, which assist developers in testing and maintaining spreadsheet models, as well as in understanding their structure [Clermont and Mittermeir, 2003; Mittermeir and Clermont, 2002; Nixon and O'Hara, 2001].

In this paper, several metrics are defined, which aim at estimating the complexity of spreadsheet models. It has been proven that the complexity of a model or a particular formula represents an important factor to be considered in the process of spreadsheet development, because a complex spreadsheet makes error finding difficult [Teo and Lee-Partridge, 2001] and because errors come in relation with cells that have a high potential for faults [Panko, 2000]. Various product and process metrics have been extensively used in the software engineering field for a couple of decades [Conte et al., 1986]. Some of them may be adopted for the purpose of spreadsheet development, yet new ones should be introduced in order to detect cells that are especially prone to errors.

It should be stressed that this paper presents work in progress. Hence, the metrics defined here are not exhaustive, and they have neither been validated in practice. The paper is organized as follows. In Section 2, the complexity metrics are discussed. Section 3 gives an example to show why it is sensible to adjust the cell error rate in accordance with the estimated degree of complexity, in order to accurately assess the level of bottom-line reliability. Since the definition of logic structure metrics indicates that spreadsheets do not permit efficient modelling of complex logical tests, two
conceptual constructs are proposed in Section 4 – the reference branching condition cell and the condition block. Directions for further work are stated in the concluding Section 5.

## 2 COMPLEXITY METRICS

Figure 1 gives a summary of metrics. The formula size and structure metrics are mostly adapted from traditional software engineering. They comprise measurements of logic structure, because tests and branches are possible only within individual formulas. Since there can be no loops, a sensible metric is the *decision count*. It is the number of simple conditions – conjunction and disjunction predicates – within one formula.

### Table 1: Summary of metrics with regard to usage

<table>
<thead>
<tr>
<th>Formula size</th>
<th>Cell range</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Number of operators</td>
<td>- Range width (height)</td>
</tr>
<tr>
<td>- Number of operands</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formula structure</th>
<th>Cell cascade</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Nesting level of a token</td>
<td>- Cell fan-in</td>
</tr>
<tr>
<td>- Average nesting level</td>
<td>- Cell fan-out</td>
</tr>
<tr>
<td>- Depth of nesting</td>
<td>- Reachability of a cell</td>
</tr>
<tr>
<td>- Decision count</td>
<td>- Average reachability</td>
</tr>
<tr>
<td></td>
<td>- Average path length</td>
</tr>
<tr>
<td></td>
<td>- Maximal path length</td>
</tr>
<tr>
<td></td>
<td>- Total number of paths</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cell references of a formula</th>
<th>Modular structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Dispersion of references</td>
<td>- Number of data binding triples</td>
</tr>
<tr>
<td>- Single column (row) reference</td>
<td>- Percentage of unreferenced data</td>
</tr>
<tr>
<td>delta</td>
<td></td>
</tr>
<tr>
<td>- Maximal positive (negative) delta</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Summary of metrics with regard to usage

An important complexity factor is represented by the nesting levels of formula operators and operands. Nesting occurs, because each function operand can be a result of another function. Two metrics can be defined: the *depth of nesting* and the *average nesting level*. The latter is computed in the following way:

\[
NL_{Avg} = \frac{\sum_{i=1}^{NL_i} NL_i}{N_1 + N_2}.
\]

Here, \(N_1\) and \(N_2\) are the numbers of total occurrences of formula operators and operands, and \(NL_i\) is the nesting level of \(i\)-th token.Literal values and variables (cell references) are considered as operands. Operators include function names and special symbols. It should be stressed that the depth of nesting in a formula can be reduced by assigning primitive functions to other cells in a worksheet and making references to those cells. In this case, however, the lengths of cell cascades increase. Consequently, unnecessary computational details are exposed, and it becomes more difficult to put input data in close physical proximity to dependent results of calculations. It therefore remains an interesting task to prove experimentally whether reducing nesting improves auditability and modifiability of a spreadsheet. Another point to notice is that the number of all cell references increases. Yet, these references are rather uniformly distributed among several formulas, implying that the average number of references per single cell lessens. Thus, a correlated research question worth investigating is whether there exists a higher probability of linking errors when there are \(N\) references altogether and averagely \(n\) references in a single cell, or when there are \(M\) respectively \(m\) references, so that \(N < M\) and \(n > m\). According to some studies [Janvrin and Morrison, 2000; Panko, 2000; Teo and Lee-Partridge, 2001], linking errors are classified as a
special case of mechanical spreadsheet errors. They represent wrong cell references which are destroying the integrity of subsequent modules, variables and algorithms.

Because there can be many ways to reach each formula cell in a cascade, a useful metric is the reachability of a cell. To define it, the number of direct links that lead in a cell and the number of direct links out of a cell have to be measured. Fan-in is thus the count of references to other cells (precedents), while fan-out is the count of references from other cells (dependents). It can now be said that a bottom-line cell is a cell with fan-out equal to zero. It has no dependents, and it is considered to be an output variable. It terminates a computational cascade which can have several independent zero-fan-in starting cells with no precedents. These cells store data, and are regarded as input variables.

One approach to determine reachability is shown on Figure 2. It describes each path as a sequence of arcs from one of the start cells to the relevant terminal cell.

Figure 2: Paths through a cell cascade

The first number written within each symbol on Figure 2 represents the fan-in of a cell, while the second number denotes its reachability. It can be seen from the example that there are four unique paths to reach the cell marked with an asterisk. Its reachability is hence equal to four. Similarly, the reachability of the bottom-line result cell is seven. It is fairly evident that the reachability of a cell is computed by summing the reachabilities of preceding cells that are directly linked by references, unless a cell has no precedents, in which case its reachability equals to one:

\[ R(C) = \begin{cases} 1 & , \text{if } \sum_{i=1}^{FI(C)} R(C_i) = 0, \\ \sum_{i=1}^{FI(C)} R(C_i) & , \text{otherwise.} \end{cases} \]

In this formula, \( R(C) \) is the reachability and \( FI(C) \) fan-in of the observed cell \( C \). Cells, which are referenced by \( C \), are denoted with \( C_i, i = 1, \ldots, FI(C) \); their reachabilities are marked with \( R(C) \).

Conceptually, each computational cascade has only one terminal zero-fan-out cell. The total number of paths in a cascade equals the reachability of this bottom-line cell. The average reachability of a cascade is easily calculated: \( R_{avg} \) for the given example is 17/6. Also, the number of cells in each path can be counted, the path with the maximum length can be found, and the average length can be computed.

Cell ranges should be given special treatment. Firstly, a reference to a range has to be transformed into as many single-cell references as there are cells in a range. This is required because each cell is accessed and processed separately. Thus, each cell in a range is on its own independent path. Secondly, a series of calculations in a range with copied formulas could be regarded as a loop with a fixed number of iterations. It tends to be more auditable than a group of cells with different formulas, but it exhibits a risk potential because of possible invalid absolute and relative references. Two size metrics – the width \( (SW) \) and the height \( (SH) \) of a range – are therefore defined. Suppose that there are two single-column ranges, \( A \) and \( B \). Cell formulas in the range \( B \)
reference cells in the range $A$, so that $s$ successive cells are accessed at a time. The equation $S_{HA} = s$
must hold true in the case of absolute referencing, regardless of the size $S_{HB}$, and $S_{HA} = S_{HB} + s - 1$
must hold true when linking $A$ and $B$ via relative references. Otherwise, there exists a great
probability of error.

It has been shown that mechanical error rates rise dramatically when equations contain references
to cells that are in both different columns and different rows than a cell containing the formula
[Panko, 2000]. For this reason, the dispersion of references is measured. The most simple way to
define it is to use the exponential utility function:

$$DR = 1 - \exp(-\alpha \cdot \Delta),$$

$$\Delta = \sum_{i=1,N} |DX_i \cdot DY_i|.$$

Here, $N$ is the number of formula operands representing cell references, $DX_i$ and $DY_i$ are the
column and row reference deltas for the $i$-th cell reference and $\alpha$ is a small positive constant. If a
referenced cell is in the same column or row as the formula cell, then either $DX_i$ or $DY_i$ equals to
zero, and the reference does not contribute to the dispersion. The exponential function is applied
for the computation of $DR$ because cell distances do not linearly influence the model readability;
however, the sigmoid function form could be used as well. The constant $\alpha$ sets the slope of the
dispersion function and determines which distance sums are “still good enough” in terms of nearly
all quality characteristics, including the communicative effectiveness, comprehensibility and
design. The value of $\alpha$ has an order of magnitude of $10^{-2}$ and has been set experimentally. Figure 3
demonstrates that pretty realistic degrees of dispersion are obtained for $\alpha = 0.01$.

<table>
<thead>
<tr>
<th>$\Delta$</th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.0952</td>
</tr>
<tr>
<td>20</td>
<td>0.1813</td>
</tr>
<tr>
<td>50</td>
<td>0.3935</td>
</tr>
<tr>
<td>100</td>
<td>0.6321</td>
</tr>
<tr>
<td>150</td>
<td>0.7769</td>
</tr>
<tr>
<td>200</td>
<td>0.8647</td>
</tr>
<tr>
<td>300</td>
<td>0.9502</td>
</tr>
</tbody>
</table>

Figure 3: Dispersions of references obtained for $\alpha = 0.01$

Other equations to determine the dispersion degree will have to be defined and evaluated within
the scope of further research work, as the proposed formula has several drawbacks.

1. Even if the referenced cell is in the same column or row (either $DX_i = 0$ or $DY_i = 0$) as the
observed cell formula, the non-zero reference delta has to influence the calculated level of
dispersion. Especially, if a co-ordinate delta is large (for example, when the Manhattan
distance exceeds 20 cells), such a reference hinders the ability to audit, test, modify and
comprehend a spreadsheet model. To take into account this kind of reference deltas, the $|DX_i \cdot
DY_i|$ product will be replaced by $L_1$ (Manhattan) and $L_2$ (Euclidean) distances.

2. If, simultaneously, cells in the same column and cells in the same row are referenced, so that $\exists
i, j: DX_i = 0 \land DY_i = 0 \land DX_j = 0 \land DY_j = 0$, the auditability, modifiability, comprehensibility
and other quality characteristics additionally deteriorate.

3. References may be balanced or unbalanced. In the first case, they are all oriented to the
common direction, which means that they point to cells which are either close to a single co-
ordinate axis or which are in the same quadrant. For this reason, it would be sensible to
consider references as two-dimensional vectors, and to determine angles between these
vectors. Clearly, this would be quite a restrictive approach, since most modellers obey good
spreadsheet design practice which advises us to make references solely to the left or upwards [Read and Batson, 1999].

In order to partially overcome the three stated problems, the dispersion of references is supplemented with the column span and row span. Spans are computed by subtracting maximal negative reference deltas from maximal positive deltas. These measures are necessary, because a matrix of approximately twenty times twenty cells is visible to the spreadsheet modeller, auditor, user or maintainer. Large spans therefore noticeably reduce the auditability and reliability. The dispersion does not carry this information, as it varies according to the number of references, and may distort the complexity picture, if some referenced cells are in the same column and some other in the same row as the treated formula.

A way to improve spreadsheet design is the structuring approach [Janvrin and Morrison, 2000]. Large models are modularized by creating worksheets. Each individual worksheet becomes a structured module that takes a defined set of inputs and translates them into a prescribed set of outputs. Values are linked as outflows and inflows from one module to another by cell references. Each major component of a spreadsheet is thereby represented as a different module, and all these modules are integrated into a coherent whole.

If a spreadsheet has such a modular design, the sharing of data among modules can be measured. One approach, which is adopted from traditional software engineering [Conte et al. 1986], is to count the number of data binding triples (P, Q, R). The variable Q is set by the module P and read by the module R. The more triples there are for the modules P and R, the more complex is their relationship. All variables in a spreadsheet model are global; they can be accessed from anywhere, but they do not necessarily pass from one module to another. The lower fan-in and fan-out the module has, the more independent it is. This certainly has an important influence on reliability, modifiability and auditability.

3. ESTIMATION OF BOTTOM-LINE ERROR RATE

In programming, one of the basic metrics is the number of faults per thousand lines of non-comment code. A similar metric for spreadsheet models is the cell error rate (CER) – the percentage of non-label cells containing errors [Panko and Sprague, 1998]. CER is estimated to be between 1% and 2%. However, according to conducted research studies, between 20% and 60% of spreadsheet models contain at least one serious error [Panko, 2000]. The difference is due to the fact that bottom-line values are computed through long cascades of formula cells. Because in tasks that contain many sequential or subordinate operations error rates multiply along cascades of subtasks, the fundamental equation for computing the bottom-line error rate is based on a memoryless geometric distribution over cell errors:

\[ E = 1 - (1 - e)^n. \]

Here, E is the bottom-line error rate, e is the cell error rate and n is the number of cells in the cascade. E indicates the probability of an incorrect result in the last cascade cell, given the probability of an error in each cascade cell is equal to the cell error rate.

Figures 4 and 5, respectively, provide an example to show that the cell error rate should necessarily be adjusted in accordance with the complexity of each individual cell. This is a prerequisite for the bottom-line error rate to be accurately estimated.
Figure 4: Spreadsheet flow diagram [Ronen et al., 1989] for the weighted additive composition of single-attribute utility functions

Figure 5 gives two logically equivalent, yet physically different, implementations of the multiple-attribute choice problem, which is modelled on Figure 4. The first cascade consists of only five cells and has substantially lower bottom-line error rate than the second cascade, which consists of nine cells. However, all formulas in the latter are very simple and easily understandable, while the former contains a fairly complex formula in the cell labeled 'Utility'. This formula includes six cell references (two cells – ‘CPU’ and ‘RAM’ – are referenced twice), several operands and operators, and nests tokens on three levels. Such formulas have the potential to be considerably more liable to errors than simple formulas or cells containing data. Consequently, the cell error rate $e$ should be adjusted in accordance with the complexity of each individual cell. Only then could the reliability of different cascades linking together many cells with simple formulas, or few cells with complex formulas, be reasonably estimated. Hence, it is essential to define and to apply some appropriate measures of complexity.

![Spreadsheet Flow Diagram](image_url)

$e = 0.02, n = 5 \rightarrow E = 0.0961, n = 9 \rightarrow E = 0.1663$

Figure 5: Cell cascades which implement the weighted additive composition of single-attribute utility functions
4. CONDITIONAL CONSTRUCTS

It is sometimes useful that a model is allowed to perform different operations dependent upon different values of input variables or intermediate calculations. Yet, logic structure is bound to logical tests within individual cells of the electronic spreadsheet. Conditional functions may thus quickly become overly complex, because conditions have to be nested on many levels within formulas. The complexity additionally increases when sequences of several computational operations are required to be calculated as a consequence of all, or at least some, logical expressions. In the latter case, each operation is usually included on a separate level of a common conditional formula. It may therefore be concluded that auditability, modifiability, reliability and comprehensibility are hindered.

Several approaches to applying complex branches exist. Firstly, conditional statements can be written in the integrated structured and/or object-oriented programming language, as is for example Visual Basic (in the case of Microsoft Excel). More efficient branching of multiple subordinate IF statements is thus enabled, in the sense of better modifiability and comprehensibility. It also allows for the inclusion of an arbitrary number of ELSE IF blocks on the same nesting level. The drawbacks of this approach are that a single cell might potentially contain a lot of references to other cells, and that modellers, auditors and maintainers have to deal with hybrid spreadsheet contents. Hence, it could be argued that neither the auditability nor reliability improve. The second solution is the use of the LOOKUP function [Read and Batson, 1999] which handles arbitrarily many unnested conditions. However, this construct requires comparisons based on logical equivalence. The reason is the origin of the LOOKUP function which is a form of an SQL query. The condition block is therefore proposed. It is a slight modification of LOOKUP. The first column (row) of the block declares conditions, and the second column (row) specifies bottom-line operations of computational cell paths. The result of the evaluated condition block is obtained by the CONDITIONB(range) function. No lookup value is required, since the value of a cascade belonging to the positively assessed condition is returned.

Finally, the classical IF function may be used in the sense of the reference branching condition cell. Proper use of IF would declare two “forward” references corresponding to the positively and negatively assessed condition, respectively. The function thus evaluates the expression and forces one of two cell cascades to execute. It returns a result which is computed by a cascade of operations belonging to the executed branch. In this way, the obtained result can be referenced in other cells regardless of a branch that generated it. Because a “forward reference” cell, or any cell that is directly/indirectly referenced by it, is allowed to contain an analogous logical test, sequencing and nesting of conditions is enabled. The problem with this approach is that only one of two referenced cell cascades returns a relevant, logically reasonable result, while the other, which does not satisfy the imposed expression, evaluates itself to a nonsensical value and might even cause a fault. A possible solution to controlling the execution stream of complex branches is to hide paths. Only paths belonging to the positively evaluated conditions should be normally visible to spreadsheet users. Cells in other paths should be shaded in a predefined way, so that branches are indicated, but not calculated. A similar problem exists for the condition block. It might be solved in the same way.

In order to determine the complexity of logical structure, possible logical paths through a model are considered. Conditional constructs without dependent conditionals are found, because all sub-branches join in cells of these constructs. The complexity of a conditional construct is then recursively computed by summing complexities of all precedent/nested conditional functions, and by adding the number of conditionless computational cascades:
Here, $O(S)$ is the complexity of the observed conditional construct $S$, $M$ is the number of nested or precedent conditionals, which are denoted with $S$, and are directly or indirectly accessed by the construct $S$, $O(S_i)$ are the complexities of these conditionals, and $N$ is the number of computational cascades not containing any logical tests. When conditional expressions or cascades of operations in an IF function, a reference branching condition cell or a condition block directly/indirectly depend on other conditional constructs, these constructs determine the number of possible logical branches, out of which only some are executed. Otherwise, each computational cascade, not containing any logical tests, adds a single possible branch. If the constant $\beta$ is set to 0, the $O(S)$ measure returns the number of logically disjunctive branches leading to the same bottom-line cell. If $\beta$ has an order of magnitude of $10^{-1}$, two other complexity factors are dealt with.

1. Perceived complexity noticeably increases when conditional functions are nested on many levels of computational cascades, because auditability, modifiability, reliability and comprehensibility tend to decrease. So, the more conditionals there are between any two conditional constructs – the observed and the final one, the more should the observed construct intensify the overall complexity.

2. Different types of conditional constructs – the classical IF function, the reference branching condition cell, and the condition block – should not equally influence the overall complexity. Therefore, a specific value of $\beta$ will have to be experimentally chosen for each individual type of conditionals. In parallel, the degree of risk that is associated with the three different types of constructs will be determined.

Complexity $O(S)$ is measured for each computational cascade separately. It considers all logically disjunctive branches through a cascade. The complexity of an individual branch is computed by applying metrics that were defined in Section 2.

5. CONCLUSION

Several spreadsheet complexity metrics were discussed in the paper. These metrics have the purpose of identifying cells which are specially liable to errors, and of adjusting error rates of individual cells in order to properly estimate the probability of wrong bottom-line results. It has to be stressed, though, that the paper presents work in progress, so much effort remains to be invested into further research activities.

The proposed metrics will be validated. It will have to be determined which among them are useful and which must be modified or substituted with more appropriate measurement instruments. A very important task is to apply metrics to actual spreadsheet models in order to determine to what extent the identified complexity factors contribute to obtained error rates. Hence, a correlation between complexity metrics and CER must be specified. Additional complexity factors, which have been overlooked throughout the first research stage, will have to be defined as well. Last, but not least, the proposed metrics should be implemented as a part of an automated analysis tool.

In the paper, only product metrics were dealt with. Traditional software engineering also applies process metrics, such as the development duration and effort. It would make sense to define this kind of quantitative measurements for the purpose of spreadsheet development, and to formally correlate them to the discussed cell complexity factors.

6. REFERENCES


A Toolkit for Scalable Spreadsheet Visualization

Markus Clermont
Software Quality Research Laboratory
Dept. of Computer Science and Information Systems
University of Limerick
IRELAND
E-Mail: markus.clermont@ul.ie

ABSTRACT

This paper presents a toolkit for spreadsheet visualization based on logical areas, semantic classes and data modules. Logical areas, semantic classes and data modules are abstract representations of spreadsheet programs that are meant to reduce the auditing and comprehension effort, especially for large and regular spreadsheets.

The toolkit is integrated as a plug-in in the Gnumeric spreadsheet system for Linux. It can process large, industry scale spreadsheet programs in reasonable time and is tightly integrated with its host spreadsheet system.

Users can generate hierarchical and graph-based representations of their spreadsheets. This allows them to spot conceptual similarities in different regions of the spreadsheet, that would otherwise not fit on a screen. As it is assumed that the learning effort for effective use of such a tool should be kept low, we aim for intuitive handling of most of the tool's functions.

1 INTRODUCTION

In today’s business life many important decisions are based on the results of spreadsheet programs. For a software engineer a spreadsheet program is obviously software and thus should be developed obeying some systematic approach and then be carefully tested. For the typical spreadsheet user, who is not a software engineer, but rather an expert in the application domain, a spreadsheet program is not considered as software but as a tool for performing calculations and formatting their results. Spreadsheets are often considered as a word-processor for numbers, and not as the highly complex data flow program that they really are.

Another issue that we often observe is overconfidence. Even if the spreadsheet users are aware that their spreadsheet is complex and the occurrence of errors is likely, they do not believe that there could possibly be an error in any spreadsheet that they have written. In field audits we have observed a case were the author of a spreadsheet denied the presence of errors in his spreadsheets even when they were pointed out to him, although they had a severe impact on the spreadsheets’ results.

So, despite the importance of spreadsheets there is still only little effort made to assure the correctness of spreadsheet programs. Although there are already a couple of possible methodologies to either enforce a systematic development of spreadsheets, according to software engineering principles (see [Chadwick 1999, Isakowitz 1995, Ronen 1989, Knight 2000] or to reduce the error rate of already existing spreadsheets, by testing (see [Ayalew 2001, Rothermel 2000, Burnett 2000]) or auditing (see [Butler 2000, Sajaniemi 2000], and other commercially distributed tools), they are still not widely accepted. One reason for the poor acceptance of
approaches that require a systematic development of spreadsheet programs is the nature of the spreadsheet itself. As they are meant to be a modelling tool for end users (see Nardi 1990, Brown 1987) it seems exaggerated to force the users into a distinct design phase before implementing the actual spreadsheet. Additionally, spreadsheet programmers are end users and would need extra training in order to apply a systematic approach.

Unfortunately, the lack of systematic development is not counter-balanced by rigorous checking of the finished spreadsheet. As it was found out earlier by Panko (see [Panko 1998]) checking a spreadsheet is a time consuming and expensive task. Thus, it should be limited to the crucial parts of spreadsheets that are most likely to be subject to errors. However, it is not trivial to identify these parts in a quick and efficient way. There are some methodologies that operate on user assessment of the risk of an error in a certain region of the spreadsheet (see [Butler 2000]), but they are subject to the auditors attitude and might not map to the actual erroneous areas of spreadsheet programs. Although there are approaches to direct the auditors’ attention to certain parts of the spreadsheet, mainly by means of visualizing the distribution of equal formulas throughout the spreadsheet, the approaches existing so far work fine for small examples, but are not efficient to handle large, ‘industry-sized’ spreadsheets of some 10000 cells or more.

In this paper we will introduce a toolkit that supports auditors to identify the hot-spots in spreadsheet programs based on the concepts of semantic classes and data modules (see [Mittermeir 2002, Clermont 2003.2, Clermont 2003.3]). Both, conceptually and technically it is a further development of an auditing toolkit that was used in a case study also presented at this conference (see [Clermont 2002]). Therefore, we will briefly summarize these concepts in the next section. Section 3 will outline the usage of our prototype and in section 5 we will demonstrate its capabilities by means of an example.

2 SEMANTIC CLASSES AND DATA MODULES

Our auditing toolkit is mainly based on two orthogonal methodologies to identify related cells in a spreadsheet. The first methodology tries to identify semantic classes, i.e. blocks of similar cells, where the criterion for similarity is based on the structure of a cell’s formula. The second concept, data modules, is based on identifying blocks of cells, that are tightly linked by cell references.

2.1 Semantic Classes

As mentioned above, a semantic class can be considered as a set of blocks of similar cells (a more formal definition is given by [Mittermeir 2002]). These blocks, in the rest of the paper called semantic units, have to satisfy certain geometric conditions that can restrict their horizontal and vertical extension as well as the size of gaps in these blocks. In the current toolkit the geometrical conditions have to be supplied by the users by means of three parameters: \(d_h\), \(d_v\) and \(d_{man}\).

The first two specify the maximal size of gaps in the semantic unit, either horizontally \((d_h)\) or vertically \((d_v)\). Thus, by setting \(d_h\) to 1 and \(d_v\) to 0 users can require semantic units to consist of horizontally adjacent cells. Setting \(d_h\) to 2 and \(d_v\) to 0 allows semantic units to consist of horizontally adjacent cells, with gaps spanning at most one cell.

If both \(d_h\) and \(d_v\) are set to values greater than zero, semantic units are can be of a rectangular shape. As users will sometimes want to restrict the overall size of gaps in these rectangles, the parameter \(d_{man}\) allows to specify the maximum manhattan distance\(^1\) that is spanned by a gap. Setting \(d_h\) and \(d_v\) both to 2 would result in rectangular semantic units, with horizontal and vertical

\(^1\)The Manhattan distance between two cells with the coordinates \((x_1, y_1)\) and \((x_2, y_2)\) is \(|x_1-x_2| + |y_1-y_2|\).
gaps (see Figure 1). However, in order to forbid the maybe unwanted case shown in Figure 1 in rows 11 to 12, $d_{\text{Man}}$ can be set to 1.

In order to group a couple of semantic units into a semantic class, they are required to be similar. Two semantic units are considered similar, if they have an identical geometrical shape and extent, and all the cells on same relative positions in the semantic units are similar. Two cells are considered similar, if their formulas are either

- copy-equivalent, i.e. they are absolutely identical, maybe created by copying and pasting a cell, or
- logical-equivalent, i.e. they differ only in absolute cell references\(^2\) and constant values, or
- structural-equivalent, i.e. they differ in relative\(^3\) and absolute cell references and in constant values. Structural equivalence means there are the same operators in the same order.

These similarity criteria comply with the so-called node-equivalence-classes that have been introduced in [Clermont 2003] in the context of logical areas. It is shown that there exists an ordering relation between the degrees of similarities, because two copy-equivalent formulas are also logical equivalent, and two logical equivalent formulas have to be structural equivalent, too. Thus, it is said that structural equivalence is weaker than logical equivalence, and logical equivalence, again, is weaker than copy equivalence.

Thus, a semantic unit is a set of cells that satisfy the imposed geometrical restrictions and a semantic class is a set of semantic units with cells on the same position relative to the semantic units’ upper left cell are in the same logical area, i.e. are either copy-, logical- or structural equivalent. Users can control the requested degree of similarity between the semantic units between two further parameters, $e_{\text{Start}}$ and $e_{\text{Rest}}$. Both specify the requested similarity between the cells in semantic units in the same semantic class. $e_{\text{Start}}$ specifies only the similarity for the top-left cells, whereas $e_{\text{Rest}}$ is applied to all the other cells.

\(^2\)An absolute cell reference is a cell reference that specifies the coordinates of the referenced cell relative to the upper left corner of the spreadsheet (see [Clermont 2003.2]). In all major spreadsheet systems an absolute cell reference is marked by the $-sign in front of the cell coordinates.

\(^3\)A relative cell reference gives the coordinates of the referenced cell relative to the referencing cell.
This separation was originally introduced to decrease the number of automatically defined semantic classes, by allowing a higher degree of similarity between the starting cells than between the rest of the cells.

2.1.1 Auditing Strategies

Semantic classes are meant to generate a compact abstract representation of the spreadsheet. Each semantic class represents a set of similar semantic units, and again each semantic unit represent a set of geometrically and because of the recurring pattern, also conceptually related cells. Hence, a straight-forward, but effective auditing strategy is:

1. Identify the semantic units and semantic classes
2. Check whether semantic units in the same semantic class are distributed according to a certain geometrical pattern on the spreadsheet
3. Find outliers or irregularities in the pattern and add them to the list of hot-spots.
4. Repeat 2 and 3 for all semantic classes
5. Check the hot-spots in detail, using any effective auditing or testing technique.

Obviously, testing or auditing is still necessary, but the amount of cells that has to be checked is efficiently reduced. Furthermore, most of the irregularities are very easy to spot, because semantic units correspond very often to whole rows or columns of the spreadsheet to examine. In contrast to many existing approaches that are considered inefficient, the irregularities can be spotted on a higher level of abstraction and are thus, easier to find.

Besides searching patterns, there are two more effective auditing strategies. The first one relies on the assumption, that errors do very often materialize themselves by means of slight deviations, i.e. a misreference or a constant instead of a cell reference. Thus, comparing the result of the analysis of the spreadsheet with varying similarity criteria effectively points out hot-spots. If the two results are not identical, auditors see themselves confronted with the question: Why are specific semantic units logical equivalent, but not copy-equivalent? Of course, sometimes this is just what the spreadsheet users wanted, but sometimes it is a strong clue for an error.

The third auditing strategy is based on the partly data-flow nature of a spreadsheet program. Therefore, the way a given spreadsheet works can be reconstructed by examining the cell-dependencies. Unfortunately, for large spreadsheet programs, the data-dependency graph (DDG) becomes too complex to be comprehended with reasonable effort. Thus, a more compact, but still meaningful representation has to be offered. We create an abstraction of the DDG, the so-called SRG, that is a directed graph, where each semantic unit is represented by a vertex, and there is an edge between two vertices \( v_1 \) and \( v_2 \), if any cell in \( v_2 \) references any cell in \( v_1 \). A detailed inspection of the SRG can help to give insight into the way the spreadsheet works, and direct the auditors’ focus on details that might be hidden in the mass of information that is shown in the DDG.

A more detailed discussion of auditing strategies that rely on semantic classes would be beyond the scope of this paper and is given in ([Clermont 2003.1] and [Clermont 2003.3]).

2.2 Data modules

In this section we give only a brief summary of the concept of data modules. For a more detailed discussion, we refer to [Clermont 2003.3]. Spreadsheet programs have some basic characteristics

---

4The DDG is a directed, acyclic graph, very edge vertex corresponds to a cell in the spreadsheet. There is an edge between two vertices \((v_1, v_2)\) if the formula in \(v_2\) references the value displayed in \(v_1\). Empty cells that are not referenced by any other formula are not represented in the DDG.

5SRG is the acronym for set relation graph, because each node in the SRG represents a set of cells.
of data flow programs and of graph-reduction programs, too (see [Mittermeir 2004]). Thus, the
DDG of a spreadsheet program has an important role for its execution. As the DDG is a directed,
acyclic graph, there are some nodes, that are not sources of further edges, i.e. sink nodes.

To grasp the idea, one can assume that a data module is a set of cells that has a distinguished result
cell, that is transitively dependent on all cells in the data module. Cells that are outside the data
module may only reference its result cell. Broadly speaking, a data module is a subgraph of the
DDG, that has only a single sink node, namely its result cell. The result cell of such a data module
is either a sink node of the DDG, i.e. a result cell of the spreadsheet program, or a node that is
connected to more than one data module. Cells that are not part of a specific data module may
reference only its result cell. Obviously, this definition is recursive, but because of the hierarchical
organisation of a DDG and its finiteness, this is not a problem.

As the data modules are not a-priori known, we have developed a way to recover them out of the
spreadsheet’s DDG. The recovery of data modules will start assuming the spreadsheet’s result
cells to be data modules and adds all cells that are only referenced by one data module to this data
module. A cell that transitively contributes to more than one data module is assumed to be the
starting point of a new data module and will be treated in the same way.

However, before the DDG can be partitioned into such data modules, the result cells have to be
identified. Obviously, not all sink nodes of the DDG have the semantics of a result of the
spreadsheet program, e.g. check-sums. In contrast to conventional programming where
intermediate results are not displayed and each subroutine has a well defined result, in a
spreadsheet each intermediate result is visible to the user and all the other formulas. Sometimes,
calculations are deliberately formulated in a more complicated way in order to obtain some desired
intermediate results.

Obviously, most of the cells of a spreadsheet program can be considered as well as a
computational auxiliary, intermediate result or result cells. For sure it can only be said that cells
that are not further referenced by other cells are result cells, because we know that users place
them on the spreadsheet with the single purpose to see their contents.If they would not like to see
the displayed value, they had not introduced this cell.

Therefore, it seems legitimate to consider DDG sink nodes, i.e. cells, that are not referenced by
other cells as result cells, and search those cells, that influence a specific result. As a matter of fact,
it is often the case, that the sink nodes in the DDG are not the real results, but check-sums. In this
case, the check-sums have to be removed manually, and the remaining DDG is then analyzed.

### 2.2.1 Auditing Strategies

Data modules are particularly useful to identify errors due to misreferences. If a intended cell
reference is not part of a formula due to an error, the data module will split up in two different
modules. In the opposite case, that a cell reference that should not be part of a formula, might lead
to the merge of two unrelated data modules.

Hence, auditors have to watch out for superfluous data modules. Subsequently, the cell where the
result of the superfluous data module should have been referenced has to be identified and
corrected. The opposite case is more difficult. If an expected data module is not part of the
visualization, auditors have to look for the cell where the missing data module is erroneously
referenced.
Although fault tracing is more troublesome, the presence of an error can be easily detected. In contrast, certain kinds of errors that are easily discovered by other techniques do not influence the resulting data modules at all. E.g., wrong operators or mis-references to cells in the same data modules, will influence the result of a data module, but not the assignment of cells to a data module, as only the data dependencies are taken into account.

A different auditing strategies makes again use of the fact that we can generate a compressed, but semantically equivalent, representation of the DDG. In the so generated SRG, each data module is a node and there is an edge between data modules, if one references the result cell of the other one. Assuming that original DDG is acyclic, the SRG will be acyclic, too.

The SRG can be used to generate a fish-eye view of the spreadsheets, if we replace one of the data modules by the subgraph of the DDG that it corresponds to. Thus, we can have a very detailed look at a certain part of the spreadsheet, without being bothered by unnecessary details, but still having an eye on the context of the part we are currently examining.

2.2.2 Fault Tracing

Fault tracing is a very common problem in spreadsheet programs, as the symptoms of errors often do not occur at the same place as the faults that cause the wrong results. Hence, most testing techniques also involve techniques for fault tracing that are usually based on the calculation of error probabilities for the predecessors of the faulty cell in the spreadsheet programs DDG (see e.g. [Ayalew 2001] or [Rothermel 2000]).

The generation of data modules and the usage of the SRG are powerful helps for fault tracing. If an error is detected in the result cell of a data module, it is not necessary to check all the predecessors in the DDG until the error is found. If the spreadsheet auditor is aware of the data module where the symptom of the error occurred, there are only two possibilities:

1. The error occurred inside the data module where it is detected, or
2. the error occurred in a predecessor module in the SRG.

It is not difficult to decide on which case applies: the spreadsheet auditor has to check the result cells of the predecessor data modules in the SRG. If they are correct, the error is buried in the module where the failure occurred. Else it is assumed that the error is propagated from the erroneous module.

For the first case, the DDG of the data module where the failure occurred has to be checked by one of the techniques that are suggested in [Ayalew 2001] and [Rothermel 2000]. Nevertheless, as a piece of extra information, the auditors are aware that the error must be in the currently examined subgraph of the DDG, and the bug tracing can stop at the module boundaries.

In the second case, the same process is repeated: it has to be checked, whether the fault occurred inside the data module, or in one of its predecessor modules. Depending on the error source, either the module is checked, or the search continues upward in the SRG.

Obviously, also a combination of error sources is possible, as errors can be hidden inside the module as well as in several predecessor modules. Nevertheless, an iteration of several testing and correction phases will finally find all the errors. In this case, the SRG is helpful because it can decrease the number of entities that have to be examined in order to find all the errors.
Next we will introduce the analysis functionality of our auditing toolkit using the concepts of data modules and semantic classes. A brief introduction into the analysis of spreadsheet programs using logical areas is given in [Mittermeir 2000] and [Ayalew 2000].

3 A SPREADSHEET AUDITING TOOLKIT

Although Excel is currently the most popular spreadsheet system, Gnumeric was chosen as target system for the prototype of the spreadsheet auditing toolkit. Due to the fact that Gnumeric is subject to the GNU-public license (see [GNU 2003] for the details), the source code is public domain, and can thus be modified and used freely.

In contrast, the Excel source code is not available at all. Visual Basic for Applications (VBA) offers extensive functionality to increase the functionality of Excel. It turned out that a first prototype of the auditing toolkit that was written with VBA had massive performance problems, because in order to assign cells to logical areas, that is a necessary task for building semantic classes later, the abstract syntax trees of the attached formulas have to be compared. As the Excel formula parser is not accessible from VBA, each formula had to be parsed again by a parser that used to be part of the prototype initially.

3.1 Advantages of Gnumeric

Although Excel has the advantage to be the most wide-spread spreadsheet system, there are some technical advantages that favored Gnumeric, as the here introduced auditing toolkit was developed merely as a research prototype. In order to make it accessible to a large market, it is necessary to revise the decision about the target-system. However, this discussion is out of scope for this paper.

Gnumeric plug-ins can be developed in any programming language that seems appropriate. They just have to register themselves to the spreadsheet systems by means of a specific plug-in API. As Gnumeric loads plug-ins dynamically into its address space at runtime, plug-ins can access all internal functions and data structures of the spreadsheet system1. Thus, not only the integration of a parser into the auditing toolkit becomes superfluous, but also the parsing of individual cells for comparing abstract syntax trees needs not be done by the add-on, because each formula is parsed as soon as it is entered and the abstract syntax tree is stored in an internal data structure.

Hence, the already stored abstract syntax tree can be used for comparing formulas. The performance of the prototype had been heavily improved by reimplementing it in the Linux and Gnumeric environment, because

1. runtime performance of C is superior to VBA and
2. formulas do not have to be parsed at analysis time.

Further advantages concerning the accessible internal functions of the Gnumeric spreadsheet system have supported the decision. These internal functions, e.g. parsing cell references of the A1 style and converting them to the R1C1 style, managing ranges and evaluating specific cells, saved a lot of development time and eliminated several potential sources of errors.

3.2 Usage

The prototype is invoked via an entry in the spreadsheet system’s main menu. Once opened, users see a dialog that offers them currently three possibilities:

1. Analysis by Logical Areas
2. Analysis by Semantic Classes
3. Analysis by Data Modules
As the first issue is already discussed at another location, we will concentrate on the latter two.

3.2.1 Using the Prototype for Semantic Class Analysis

In order to analyse a spreadsheet for semantic classes, users have to enter the required parameters, i.e. \( d_h \), \( d_v \), \( d_{Man} \), \( eq_{Rest} \) and \( eq_{Start} \) in the dialog (see the left dialog in Figure 2). After pressing the analysis button the abstraction of the spreadsheet is generated, using the algorithms discussed in [Clermont 2003.3]. The result is represented in a tree-control, where each entry corresponds to a semantic class. Attached to each semantic class are its member semantic units.

Additionally, a second dialog opens that displays a visualization of the SRG. The SRG is visualized by the graph visualization package LEDA and AGD (see [Ganser 1999]).

In each of the offered visualizations, users can select an entry (i.e., a semantic class or a semantic unit in the tree control, or a node in the SRG), and the corresponding cells in the spreadsheet are selected. This feature can be used for instant coloring of cells, or for the identification of patterns.

3.2.2 Using the Prototype for Data Modules Analysis

In order to reconstruct the data modules of a spreadsheet we must, at first hand, identify the result nodes of the spreadsheet. Therefore, users are presented a list of sink nodes in the spreadsheet’s DDG as well as a graphical display of the DDG, with sink nodes colored red. However, the later is sometimes no useful help, as DDG with more than 1000 nodes and 10000 edges do occur. The dialogue pictured in Figure 2 on the right hand side shows the results for the analysis of a spreadsheet that has obviously two ‘result’ cells (see Figure 3, on the left). If the user decides to remove one of the sink nodes from further analysis, maybe because it represents a check-sum, it is removed from the list of sink nodes and replaced by its predecessors in the DDG.

Again, the result of the analysis is shown in the structure of a tree, with each data module and its associated cells forming one entry in the tree. Users can also examine the SRG (see Figure 3, on the right) and select a data module in the tree view and zoom into it (see the resulting SRG in Figure 4). In this case the displayed SRG is modified, such that the node that corresponds to the data module zoomed into is replaced by the subgraph of the DDG that contains its member cells and their references.
4 DISCUSSION

The main motivation behind the development of this toolkit was the lack of scalable auditing tools at the moment. The viability of most auditing tools is demonstrated only on small scale laboratory examples and most of them do not hold in the large what they promise in the small. Most of the existing approaches perform very well on small spreadsheets, or by working on one part after the other, but, unfortunately, the effort seems to grow exponentially with the size of the analyzed spreadsheet.

Obviously, as there are different kinds of spreadsheet, different approaches have to be selected for each type of spreadsheets. As a matter of fact, we found out, that semantic classes deliver very good abstractions of large spreadsheets that are usually created by copying, pasting and modifying cells. One of the example spreadsheets that we analyzed consisted of some 1200 formula cells, but boiled down to 23 semantic classes. There are other spreadsheets that do not partition well into semantic classes, usually consisting of a high number of semantic classes with only a few members - but some of them are well suitable for the analysis by data modules, e.g. if they consist of more or less independent parts that are aggregated into the calculation of some business figures.
Figure 3: On the left hand side, the DDG of a small example spreadsheet (taken from ...) with 68 nodes and 86 edges. On the right hand side the SRG representing data modules as nodes of the same spreadsheet.

Figure 4: The same SRG as in Figure 3, right hand side, but with an expansion of the node originally labeled K10.

Each of the so discovered data modules or semantic classes can then be checked independently, using any other test methodologies. For semantic classes, the effort is reduced by checking only one semantic unit in a class - if this one is correct, we have to verify the places where it turns up only. Data modules allow auditors to check one part of the spreadsheet at a time without losing information about the context.

Unfortunately, we cannot make any claim on the correctness of a spreadsheet. Even if the analysis with semantic classes or data modules does not show any irregularities, and every formula is correct, the spreadsheet also depends on the input-values, that are not considered by this approach.
Nevertheless, the here introduced tool offers functionality to easily discover constants in the spreadsheet and in case that an erroneous result is discovered, data modules will support the actual fault tracing.

5 FURTHER WORK

Although the here introduced toolkit offers an efficient way to discover irregularities, even in large spreadsheets, there are, of course, many points that can be improved. To start with, the auditing toolkit is a prototype and, thus, the user interface needs still improvement. Additionally, the fact that the auditing toolkit requires a Linux-box running the gnumeric spreadsheet system is a drawback if you try to open to a wide market. Hence, integration with Excel - and re-implementation of the prototype in a Windows environment - seems necessary.

Additionally, the underlying concepts could be extended in manifold ways. One of the most promising that is currently dealt with in a upcoming master-thesis (see [Hipfl 2004]) includes layout information into the analysis and can automatically identify the parameters for the semantic class analysis. Furthermore, it is no longer necessary to have the same parameters applied to the whole spreadsheet. This approach has already been integrated into the here presented prototype (see [Hipfl 2004] for more details).

6 REFERENCES


A Toolkit for Scalable Spreadsheet Visualization: Markus Clermont


Using Layout Information for Spreadsheet Visualization

Sabine Hipfl
Institut für Informatik-Systeme
Universität Klagenfurt
shipfl@edu.uni-klu.ac.at

ABSTRACT
This paper extends the spreadsheet visualization technique reported in [Clermont, 2003] by using layout information. The original approach [MC, 2002] identifies logically or semantically related cells by relying exclusively on the content of cells for identifying semantic classes. A disadvantage of semantic classes is that users have to supply parameters which describe the possible shapes of these blocks. The correct parametrization requires a certain degree of experience and is thus not suitable for untrained users. To avoid this constraint, the approach reported in this paper uses row/column-labels as well as common format information for locating areas with common, recurring semantics. Heuristics are provided to distinguish between cell groups with intended common semantics and cell groups related in an ad-hoc manner.

1 INTRODUCTION
Systematic research and field audits [Panko, 1998] have shown that erroneous spreadsheet programs are wide spread, even as base for important decisions. This is due to lack of systematic development and testing. To overcome the poor quality of spreadsheets, in recent years, some testing (see [RLDB, 1998], [RRB, 1999] or [AC, 2002]) and visualization methods (see [Clermont, 2003], [Sajaniemi, 2000] or [Butler, 2000]) for spreadsheets have been developed.

Model visualization, as developed by Clermont [Clermont, 2003], consists of three different methods to aggregate semantically related cells to groups in order to get an abstraction of the spreadsheet and to build a comprehensive representation as basis for detecting errors. One of these methods serves to detect repeating components of spreadsheets, called semantic classes [MC, 2002]. The approach reported in this paper extends this approach by including layout information.

Semantic classes are mainly based on a notion of similarity between cells, so called logical equivalence classes [CHM, 2002]. These equivalence classes were initially used as a base for grouping cells with the same content into logical areas [ACM, 2000]. This approach was extended to assign similar groups of cells, so called semantic units, into semantic classes [MC, 2002]. By this, semantic classes are an abstraction of a spreadsheet, based on the similarity of formulas. The user has to specify, whether semantically related cells are spread out column-wise, row-wise or block-wise. Thus the content of cells and spatial situations are taken into account and it is checked, whether the semantic content of these areas is of repetitive nature.

The spatial parametrization can be tricky in certain cases, as the orientation can be indicated only for the whole spreadsheet. Sometimes a spreadsheet consists of different blocks of cells more or less related. As the orientation of one block could be row-wise, that of the other block column-wise. The parametrization must be done by the spreadsheet user.
user itself. As this demands a high knowledge of the structure of the spreadsheet it requires a certain degree of experience and might not be suitable for the untrained user.

Moreover, semantic classes only identify semantically related blocks of cells having the same extent. This requires the blocks to be of the same size. But blocks of cells that are semantically related but do have a different size will not be identified in their full extent.

The approach presented in this paper overcomes these hurdles, as explicit parametrization is avoided. This is done by using layout information to identify and group semantically related cells, yielding so called layout areas. These can then be checked whether they contain semantic classes.

Layout information in spreadsheets is, for example, bands of labels or frames. The spreadsheet-programmer uses layout information to structure the spreadsheet and to increase the understandability and readability of the sheet. The main idea behind using layout information for spreadsheet visualization is based on the general area of application of layout information: the spreadsheet-programmer uses layout information to describe semantically related areas on the spreadsheet. For the reconstruction of these related areas layout information is used again. By this

- explicit parametrization is avoided as
- implicit identification of the orientation of cells is possible,
- the orientation is identified for each block of cells individually and
- it is possible to identify semantically related blocks of cells with different extent.

Including formulas, spatial situations and layout information of cells can help

- to get a better view of the spreadsheet and
- to better recognize related areas.

This can lead to two major advantages:

- The comprehension of the whole spreadsheet improves and
- the reconstructed model of the spreadsheet is more meaningful.

Briefly, the technique that is subsequently presented assigns cells to labels and then detects similarities between cells that are assigned to similar labels. The notion of similarity between groups of cells that is used throughout this paper is based on logical equivalence [MC, 2002]. The generated abstraction consists of semantic classes where each semantic unit bears a label. Thus, the auditing techniques that can be applied to this abstraction are the same as [CM, 2003] suggest for semantic classes.

In contrast to existing techniques spreadsheet-programmers are not forced to introduce any specific layout characteristics into a spreadsheet. However, the approach is able to use the present information to create a meaningful abstraction of the spreadsheet. The analysis for similarity between the groups of cells goes also much more into detail as many of the commercial available toolkits do. Therefore, a high number of semantically equivalent blocks can result and color coding of similar cells of the spreadsheet is not directly supported. This is not a disadvantage at all because users are enabled to audit the
spreadsheet on the base of the high level abstraction and not on the cell level, making the approach suitable for large spreadsheets.

The following section introduces different kinds of layout information and their relevance for the method of layout areas. In the third section some basic formal definitions are given, which are important for the implementation of the toolkit. In the fourth section the four heuristics on which layout areas are built are presented and illustrated by examples.

2 LAYOUT INFORMATION IN SPREADSHEETS

Layout information can be classified into three main groups:

- Labels
- Formatting of the content of cells
- Lines and Frames

In general, layout information is used to structure the spreadsheet. This makes the spreadsheet more readable as well as more understandable.

2.1 Labels

Labels are used to describe the semantics of cells and to help the spreadsheet-user to understand the usage of the whole spreadsheet, of blocks of cells as well as of single cells. Labels have the highest impact for the here introduced method of layout areas.

Extent of Labels

In general, labels can run horizontally to the right or vertically downwards. If labels are used to describe columns, they are called horizontal labels as they are running horizontally to the right. If labels are used to describe rows, they are called vertical labels as they are running vertically downwards.

The extent of the labels within a spreadsheet defines an implicit identification of the orientation of the spreadsheet or of a particularly labelled portion of the sheet. As horizontal labels are describing columns, such parts of a spreadsheet are column-oriented and as vertical labels are describing rows, such parts of a spreadsheet are row-oriented.

Kinds of Labels

The main idea of using labels for layout areas is the assumption that similar labels are an indicator for semantically related cells. As there are different degrees of similarity between labels, five kinds of labels can be defined. A formal definition of each kind of label is given in the next section.

- **Running numbers**

  In general, labels are cells containing a string. Running numbers are a special form of labels as they contain numbers. They are a set of cells which are running horizontally to the right or vertically downwards, containing values which are increasing in relation to the preceding cell at a given step size, usually one.

  Using running numbers in the same way as conventional labels makes sense, because in many spreadsheets running numbers are not used as numbers but as
labels to describe blocks of cells. For example, the years 2001 to 2004 can be used to describe blocks of cells in which the cash flow is calculated for four years.

- **Labels interpreted as ordinal numbers**
  These kinds of labels are a special form of running numbers. They are not similar to each other. Hence a function or a dictionary is needed to map these labels to numbers or help from the user to identify this mapping.

  For example, the twelve months January, February, March … December can be mapped to the numbers 1 … 12.

- **Counters**
  These kinds of labels are made up of one word with an attached running number in front or back of the word. It doesn’t matter if there is a blank between the word and the running number.

  For example, the labels Year 1, Year 2, Year 3 and so on are counters.

- **Labels with complete identity**
  Labels containing exactly the same word/s are called labels with complete identity.

- **Labels with partial identity**
  A set of labels that are made up of \( n \) words \((n \geq 2)\) are called labels with partial identity, if the first \( m \) \((m \leq n-1)\) words or the last \( m \) \((m \leq n-1)\) words are completely identical.

  For example, the labels Cash flow to Present Value and Cash flow increment are partially identical as they contain the completely identical words Cash flow.

2.2 Formatting of the Content of Cells

In spreadsheets different formatting of the content of cells is used to highlight related cells or to separate important issues from less important ones. The basic idea behind using different formatting for the identification of layout areas is the assumption that cells with the same format are assumed to be semantically related. Hence, format information is also used for layout areas.

Usually, there are five different kinds of formatting elements applicable for rendering the content of a cell: the font itself, the font size, the style of the font, the font color and finally the background color of a cell. In this work only the first four elements are used, the background color is left out.

2.3 Lines and Frames

In spreadsheets, lines and frames are used to separate cells, or more common, blocks of cells. This kind of layout information is not used for the identification of layout areas because the information it provides is ambiguous due to the problem of proper attribution. E.g., a horizontal line can belong to either one of the two cells that it separates. Because of this ambiguity, lines and frames are not considered. It might be worth consideration for further work.
3 FORMAL DEFINITIONS

As a base for implementation of a toolkit, some formal definitions have to be introduced. These definitions are based on the formal definitions in [Clermont, 2003].

In the following, $C$ denotes the set of all non empty cells of a spreadsheet, with $c \in C$: $(x, y)$ is the address of a cell, $v$ is the value, $f$ is the formula of cell (which can be empty), $fi$ is the formatting-information of a cell and $ri$ is the frame-information of a cell. The term label denotes a cell that contains a string and does not have a formula.

The definitions are stated in $Z$-syntax [Diller, 1996].

**Definition 1: Running Numbers**

Two cells $c_1, c_2 \in C$ are running numbers, if

- $c_1 = ((x_1, y_1), v_1, f_1, fi_1, ri_1) \land c_2 = ((x_2, y_2), v_1 + 1, f_2, fi_2, ri_2) \land$
- $(x_1 = x_2 \land y_1 < y_2 \land \exists c_3 = ((x_3, y_3), v_3, f_3, fi_3, r_3) \in C \mid x_3 = x_1 \land y_1 < y_2) \lor$
- $(y_1 = y_2 \land x_1 < x_2 \land \exists c_3 = ((x_3, y_3), v_3, f_3, fi_3, r_3) \in C \mid y_3 = y_1 \land x_1 < x_3 < x_2) )$

These properties guarantee that

- the value in cell $c_2$ is the value in $c_1$ plus one,
- if the running numbers are arranged vertically no other cell is between them or
- if the running numbers are arranged horizontally no other cell is between them.

**Definition 2: Labels interpreted as ordinal numbers**

A label $l \in C$, with $l = ((x, y), v, f, fi, ri)$, is interpretable as ordinal number, if there is an injective function $f: Strings' \rightarrow \{0, 1, \ldots, n\}$, with Strings’ $\subseteq Strings$ and $v \in Strings'$.

**Definition 3: Counters**

Two labels $l_1, l_2 \in C$ are counters, if they fulfill the following requirements:

- $l_1 = ((x_1, y_1), v_1, f_1, fi_1, ri_1) \land l_2 = ((x_2, y_2), v_2, f_2, fi_2, ri_2) \land$
- $\exists r \in Strings \mid (v_1 = r \circ n_1 \land v_2 = r \circ n_2) \lor (v_1 = n_1 \circ r \land v_2 = n_2 \circ r) \land$
- $c_1 = ((x_1, y_1), n_1, f_1, fi_1, ri_1) \land c_2 = ((x_2, y_2), n_2, f_2, fi_2, ri_2)$ are running numbers.

These properties guarantee that

- the values $v_1$ and $v_2$ are strings consisting of two parts:
- a word which is exactly the same in both labels and
- a running number in front or in back of the word.

**Definition 4: Complete Identity**

Two labels $l_1, l_2$ with $l_1 = ((x_1, y_1), v_1, f_1, fi_1, ri_1)$ and $l_2 = ((x_2, y_2), v_2, f_2, fi_2, ri_2)$ are completely identical if $v_1 = v_2$.

**Definition 5: Partial Identity**
Two labels $l_1, l_2$ with $l_1 = ((x_1, y_1), v_1, f_1, f_{i_1}, r_{i_1})$ and $l_2 = ((x_2, y_2), v_2, f_2, f_{i_2}, r_{i_2})$ are partial identical if

$$\exists u, v, w \in \text{Strings} \mid (v_1 = u \circ w \land v_2 = v \circ w) \lor (v_1 = w \circ u \land v_2 = w \circ v).$$

4 LAYOUT AREAS

The basic idea behind layout areas is to assign cells to labels. There is a distinction between geometrical and semantical assignments. Definitions 6 and 7 introduce these two concepts.

4.1 Definitions

In the sequel, $L$ denotes the set of all labels of a spreadsheet. The term \textit{label} denotes one of the five kinds of labels mentioned above and the term \textit{cell} denotes a cell containing a formula or a numeric value that is used in computations.

Geometrical assignment is used to assign cells to labels based on specific geometrical restrictions.

\textbf{Definition 6: Geometrical Assignment (GA)}

The set of geometrical assignable cells, with $c_i \in C$, of a label $l_i \in L$ is defined as:

$$GA_{l_i} = \{ c \in C \mid c = ((x_2, y_2), v_2, f_2, f_{i_2}, r_{i_2}) \land l_i = ((x_1, y_1), v_1, f_1, f_{i_1}, r_{i_1}) \land (x_1 = x_2 \land y_1 < y_2 \land \forall l_2 \in L \mid l_2 = ((x_2, y_2), v_2, f_2, f_{i_2}, r_{i_2}) \land x_1 = x_2 \land y_1 < y_2 < y_2) \lor (y_1 = y_2 \land x_1 < x_2 \land \forall l_2 \in L \mid l_2 = ((x_2, y_2), v_2, f_2, f_{i_2}, r_{i_2}) \land y_1 = y_2 \land x_1 < x_2 < x_2) \}) \land v_2 \notin \text{Strings} \}$$

if:

$$\forall c_1, c_2 \in GA_{l_i} \mid \text{dense}(GA_{l_i}, c_1, c_2, d)$$

These properties guarantee that a cell is geometrically assigned to a label only if

- the label is above or on the left side of the cell,
- there is no other label between the cell and the label and
- the cell itself is not a label.

The predicate \textit{dense} is true, if either $c_i$ is separated only by $d$ cells from $c_2$, or there exists another cell $c_3 \in GA_{l_i}$, such that \textit{dense}(GA_{l_i}, c_1, c_3, d) and \textit{dense}(GA_{l_i}, c_3, c_2, d). A more formal definition of \textit{dense} is given in [MC, 2002].

It is important to mention that a cell can only be assigned to one label. If there are two labels that a cell could be assigned to, a horizontal and a vertical one, two different cases are possible:

- If the two labels are of a different kind the problem is solved by heuristics 1 and 2.
- If both labels are of the same kind the labels are equivalent and it doesn’t matter to which label the cells are assigned.
In contrast to other approaches the user has only to indicate the maximum allowed number of empty cells within a block of cells.

The semantical assignment is used to assign cells to labels based on the formatting information of the content of the cells and labels.

**Definition 7: Semantical Assignment (SA)**
The set of semantically assignable cells, with \( c_i \in C \), of a label \( l \in L \) is defined as:

\[
SA_l = \{ c \in C \mid c = ((x, y), v, f, fi, ri) \wedge \\
( (x, y, v, f, fi, ri) \wedge \\
( (x, y, v, f, fi, ri) \wedge \\
( x \leq x' \wedge y \leq y' \wedge fi = fi' ) ) \\
( x \leq x' \wedge y \leq y' \wedge fi = fi' ) ) \\
( x \leq x' \wedge y \leq y' \wedge fi = fi' ) ) \\
( x \leq x' \wedge y \leq y' \wedge fi = fi' ) ) \\
( x \leq x' \wedge y \leq y' \wedge fi = fi' ) ) \\
( x \leq x' \wedge y \leq y' \wedge fi = fi' ) ) \\
\}
\]

These properties guarantee that the semantical assignment of a cell to a label is only done if

- the label and the content of the cell are having the same formatting information or the value of the cell itself is not defined (which means empty),
- there is no other cell between the cell and the label, containing another kind of formatting information and
- the cell must be geometrically assignable to the label, as expressed by \( c \in GA_l \).

As the same in geometrical assignment, the assignment is only done, if the cell is not assignable to another label.

**Definition 8: Layout Area**
A layout area is set of cells \( L \) and a label \( l \), if \( c \in L \Rightarrow (c \in GA_l \wedge c = l) \)

### 4.2 Four heuristics
Effective identification of layout areas is based on four heuristics. The main steps of these heuristics are always the same:

1. Identify labels.
2. Assign cells to the identified labels.
3. Group them into equivalence classes.

The result of step one and step two are layout areas, step three makes a subsequent auditing possible, as the layout areas are grouped into equivalence classes [MC, 2002].
Every cell in the spreadsheet is assigned to either one label or to none. After the identification of one kind of label cells are immediately assigned. So in each step the number of assignable cells is reduced. The first two heuristics are based on the different kinds of labels, introduced in section 2.1. The third heuristic is based on the semantical assignment and the last heuristic is based on the geometrical assignment (see Definitions 6 and 7).

The four heuristics are processed in a specific order, namely in the order in which they are presented in the sequel. This accounts from the fact that each heuristic has a different significance for denoting semantic relatedness. So the first one has highest significance, the last one the lowest.

There is also an special order within the identification of labels. This is necessary if there are, for example, running numbers and counters existing in a way, that they enclose a block of cells. In such a case the question is, to what kind of label should the enclosed cells be assigned – to the running numbers or to the counters? Because of this problem a hierarchy is introduced between the five kinds of labels. This means, the identification of labels is done in the order in which the labels are introduced in section 2.1. After the identification of one kind of label, cells are immediately assigned and the co-occurrence of two different kinds of cells will not lead to non-determination.

The hierarchy, as proposed in this paper, was introduced after analysing some 40 spreadsheets. It reflects the significance of the different kinds of labels for denoting semantical relatedness. Most of the spreadsheets studied are freely available at [Decisioneering, 2004].

It is also possible that labels of the same kind describe a rectangular area, as they enclose a group of cells. In such a case it doesn’t matter to which labels the cells are assigned, either to the vertical ones or the horizontal ones, as the labels are equivalent.

To provide auditing for the resulting layout areas, they are grouped into equivalence classes (see [MC, 2002]). These equivalence classes can be checked for irregularities, which might be symptoms of errors. The layout areas have also been integrated in the existing prototype (see [Clermont, 2003]).

**Assignment Based on Running Numbers and Counters (Heuristic 1)**

The individual steps of this heuristic are

- the identification of running numbers, labels interpretable as ordinal numbers and counters (in this order),
- the geometrical assignment of cells to the identified labels. As mentioned above, this is done immediately after one of the three groups of labels is identified.

The result of these two steps are layout areas. Subsequently

- the layout areas are grouped into semantic classes and
- heuristic 2 is applied.
Figure 1 shows an example. In this example two different kinds of labels can be identified: running numbers (vertical running) in the cells A19:A22 and horizontal running counters in the cells B18:E18. In this example the two kinds of labels are arranged in a way that they enclose a block of cells. Now the importance of the stated hierarchy can be shown. If no hierarchy is made an assignment would not be possible, because the cells B19:E22 can be assigned to the running numbers as well as to the counters. However, the running numbers have higher priority.

As stated in our introduction, layout areas make an implicit identification of the orientation of the spreadsheet possible. The orientation of the example spreadsheet is row-wise as the running numbers are horizontal labels. In figure 1, four layout areas can be identified: A19:E19, A20:E20, A21:E21 and A22:E22. To highlight these layout areas, each layout area is marked with another greyscale.

To support auditing, these four layout areas are subsequently grouped into semantic classes. As this would go beyond the scope of this paper readers are referred to [Hipfl, 2004].

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Year</td>
<td>Quarter 1</td>
<td>Quarter 2</td>
<td>Quarter 3</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Production Forecast</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>2001</td>
<td>200,000</td>
<td>250,000</td>
<td>100,000</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>2002</td>
<td>150,000</td>
<td>220,000</td>
<td>120,000</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>2003</td>
<td>133,441</td>
<td>195,714</td>
<td>106,753</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>2004</td>
<td>119,711</td>
<td>174,109</td>
<td>94,969</td>
</tr>
<tr>
<td>23</td>
<td>Total</td>
<td>602,152</td>
<td>839,823</td>
<td>421,722</td>
<td>538,747</td>
</tr>
</tbody>
</table>

Figure 1: Four layout areas build by means of running numbers.

**Assignment Based on Complete and Partial Identity (Heuristic 2)**

The individual steps of this heuristic are:

- the identification of labels with complete identity and partial identity (in this order),
- the geometrical assignment of cells to the identified labels. As mentioned above, this is done immediately after one of the three groups of labels are identified.

The result of these two steps are layout areas. Subsequently:

- the layout areas are grouped into semantic classes and
- heuristic 3 is applied.

Figure 2 shows an example. In this example complete identical labels can be identified. So each of the pairs of cells D7, H7 and E7, I7 and F7, J7 and G7, K7 contains complete
identical labels. As these labels are horizontal labels, the orientation of the example-sheet is column-wise.

Eight layout areas are identified: D7:D10, E7:E10, F7:F10, G7:G10, H7:H10, I7:I10, J7:J10 and K7:K10. In the example, each layout area built upon an identical label, has the same greyscale. These eight layout areas form four groups as areas with the same label belong together.

Figure 2: Eight layout areas build by means of complete identical labels

In a further step each of the four pairs of layout areas can be grouped into semantic classes. Further details are given in [Hipfl, 2004].

Assignment Based on Semantics (Heuristic 3)
The individual steps of this heuristic are:

For each label, not yet grouped to layout areas

- identify semantically assignable cells (see Definition 7) and
- geometrically assign the cells to the semantically related labels.

The result of these two steps are layout areas. Subsequently

- the cells within each layout area are checked for logical areas and
- heuristic 4 is applied.

Figure 3 shows an example. The formatting of the cells is described in the order font, font size, font style, and font color. The formatting of the block of cells B37:B40 is: Arial, 10 pt., normal, black. The formatting of the labels A37:A40 is Arial, 12 pt., normal, black. As the labels and the cells consist of different formatting a semantical assignment is not possible. Lets have a look at the other cells. The label A41 has the following formatting: Arial, 12 pt., bold, black. The cells B41:D41 have the same formatting. So an assignment is possible and the result is a layout area, highlighted in figure 3.
The processing of the result of this heuristic is different to those introduced before as the resulting layout areas are not grouped into semantic classes. In the first two heuristics the grouping to layout areas is based on the assumption, that similar labels are a sign for semantically related cells. But this heuristic is not based on the similarity of labels, so the resulting layout areas contain labels which might not be similar to each other. So the assumption made in heuristics 1 and 2, and consequently the grouping into semantic classes is not founded.

However, another abstraction technique can be applied. In [ACM, 2000] the method of logical areas is introduced. Cells within each layout area, resulting from this heuristic, can be checked for logical areas.

**Spatial Assignment (Heuristic 4)**

This last heuristic is in a way different from the others as the assignment is not based on any assumption of semantical relatedness. It

- identifies cells, which are not yet assigned and
- makes merely a geometrical assignment (see Definition 6).

The results of these two steps are layout areas that are not further processed.

The labels, as well as the cells, do not need to have any special characteristics. It also doesn’t matter whether the labels to which the cells should be assigned, are already in another layout area. As this heuristic does not provide any statement on semantical relatedness, there is no benefit to group the result to semantic classes or logical areas.

This heuristic assigns cells which are not yet processed by a preceding step but described by a label to layout areas. After applying this heuristic only those cells are left that can not be unambiguously assigned to a label or which are not described by any label. Such cells can be an indication of errors or of bad design.

**5 CONCLUSION**

In this paper the spreadsheet visualization approach introduced by [Clermont, 2003] is extended by using layout information. The main assumption for the method of layout areas is that layout information is used by the spreadsheet-programmer to show the semantical relatedness of cells. On these grounds semantically related blocks of cells can be identified.

The aim of the extension is to relieve users from the need to be highly experienced in a specific spreadsheet visualization technique and it is reached as follows:

1. Explicit parametrization of the semantic classes’ discovery algorithm is avoided.

   This is reached because semantic classes are identified basing on the layout areas. So the user does not need to specify spatial restrictions for blocks of cells with recurring semantics. In the extended approach labels and formatting information is
used to identify semantically related cells. Layout areas are built and subsequently out of them the semantic classes can be identified.

2. Implicit identification of the orientation of blocks is possible.

In this approach the extent of the identified labels is used to indicate the orientation of cells. If labels are running horizontal, they describe columns – thus, the orientation is column-wise. If labels are running vertical, they describe rows – so, the orientation is row-wise. The orientation of the cells does not need to be explicitly stated.

3. The orientation is identified for each block of cells individually, not for the whole spreadsheet.

Each layout area is build out of a set of labels which describe specific blocks of cells. These labels indicate the orientation of the blocks of cells they describe. Hence, the orientation is given only for blocks of cells.

4. It is possible to identify semantically related blocks of cells with different sizes.

The size of a block of cells is not taken into consideration for the construction of layout areas. The cells are assigned to labels until the termination condition is reached. So, it is possible that two belonging layout areas consist of a different number of cells.

6 REFERENCES


Spreadsheet Structure Discovery with Logic Programming

Jocelyn Paine,

http://www.j-paine.org/

ABSTRACT

Our term "structure discovery" denotes the recovery of structure, such as the grouping of cells, that was intended by a spreadsheet's author but is not explicit in the spreadsheet. We are implementing structure-discovery tools in the logic-programming language Prolog for our spreadsheet analysis program Model Master, by writing grammars for spreadsheet structures. The objective is an "intelligent structure monitor" to run beside Excel, allowing users to reconfigure spreadsheets to the representational needs of the task at hand. This could revolutionise spreadsheet "best practice".

We also describe a formulation of spreadsheet reverse-engineering based on "arrows".

1 INTRODUCTION

In our first paper, presented at EuSpRIG 2001 [Paine, 2001], we introduced the Model Master (MM) spreadsheet-description language, and showed how we could build spreadsheets by compiling them from MM programs, with advantages for readability, modularity, and code-reuse. In an MM version of Thomas Grossman's queuing simulation, where each column represented a server, we could change the number of servers simply by altering one constant.

We also introduced decompilation - translating spreadsheets to MM programs – and its benefits for reverse-engineering and error-detection. An MM program is a summary of a spreadsheet's calculations, useful in many ways. For example, Graham Macdonald has suggested that the commercial world would find it valuable in determining whether a spreadsheet accurately reflects a legal contract.

Our second paper, available on the Web [Paine, 2004], took decompilation further, making it rigorous via "spreadsheet algebra" (the phrase should be understood in the same way as, for example, "vector algebra"), which treats spreadsheets as mathematical entities. We implemented functions for operating on these, and an interface resembling those found in computer-algebra systems, which enabled the results to be displayed as MM programs or recompiled into spreadsheets.

One example decompiled one of Ray Panko's spreadsheets and put it through a series of transformations aimed at making it more intelligible. The freshly decompiled version listed cells individually, giving them their original, uninformative names - D1, D3, and so on. As we proceeded with our transformations, we were able to batch related cells together into arrays and give them names derived from neighbouring labels. Our final listing was easier to read and revealed several intentional errors in the spreadsheet.

Our motivation was that there is no "best" form for such displays. To illustrate: we recently posted on the EuSpRIG discussion list a reference to John Raffensperger's spreadsheet style guidelines [Raffensperger]. Louise Pryor replied that these are indeed good from the point of view of a user who is reading the results and changing some inputs. However, they ignore the needs of other types of users, especially those maintaining and updating the spreadsheet. To us, it is obvious that a spreadsheet must be available in different forms, depending on its user's needs. Blueprints, architectural drawings, circuit diagrams and maps respect this: even library catalogues are searchable both by author and by title. Yet spreadsheets lag behind, forcing users to adapt to one and only one way of ordering their information. Spreadsheet algebra banishes such rigidity.
At the end of that paper, we introduced the term "structure discovery" - discovering logically related cells and redescribing the spreadsheet so that they can be seen as a single data structure. In this paper, we explain structure discovery and describe an implementation which allows the user to define and apply heuristics for finding structure in spreadsheets. Our goal is an "intelligent structure monitor" for Excel, allowing spreadsheets to be reconfigured to the representational needs of the task at hand.

The structure-discovery heuristics are logical specifications of the spreadsheet, and should be coded as close to logic as possible. We are doing so using the logic-programming language Prolog. In the rest of this paper, we introduce structure discovery, explain how common structures can be described by patterns or grammars, and discuss our Prolog implementation. We also formulate reverse-engineering of spreadsheets in terms of "arrows".

We believe Prolog has many advantages for spreadsheet research, and so in our original draft, we included a tutorial on Prolog. On the advice of a referee, this is omitted from the final version appearing in the EuSpRIG 2004 proceedings. However, it can be found in the fuller Web version of our paper, at http://www.j-paine.org/spreadsheet_structure_discovery.html.

It should be noted that although this paper is concerned with one particular piece of software, namely MM, it will still interest other researchers. Structure discovery is necessary whenever one wants to try and describe the spreadsheet so as to tell more about the author's intentions - and hence have more chance of detecting errors - than the spreadsheet itself does. Much of our Prolog analysis code is independent of MM, as is the notion of translating logical specifications directly into Prolog. And our reverse-engineering formalism applies to any attempt to redescribe a spreadsheet as a collection of tables or arrays that, while mapping onto many (perhaps non-neighbouring) cells on the spreadsheet, are nevertheless to be regarded as single data structures.

2 STRUCTURE DISCOVERY

We begin with an analogy from the world of conventional programming languages. When a compiler compiles an IF statement, it does so using labels and jumps. The statement:

```
IF Condition THEN
  ThenPart
ELSE
  ElsePart
END IF
```

gets converted into code similar to that below, its exact form depending on the instruction set of the machine:

```
Condition
  If false, GOTO L1
ThenPart
GOTO L2
L1: ElsePart
L2:
```

Suppose now that we were to lose the program source. To reconstruct it, we would need a "decompiler", namely a program that reconstructs, from machine code, the source that generated it. (As [Decompilation] describes, many exist.) This IF statement was explicit in the source file, but implicit in the machine code. Spreadsheets too have high-level implicit structure. The difference is that they contain no explicit listing of this structure - it resides only in the author's brain, a location not readily open to public inspection.

Consider as example a three-column Income/Outgoings/Profit spreadsheet, where each cell in the third column computes the difference of its row-mates in the first two:

```
Income  Outgoings  Profit
=A2-B2
=A3-B3
=A4-B4
```

Logically, the columns are single entities, in the same sense in which the block of machine

---

Spreadsheet Structure Discovery with Logic Programming: Jocelyn Paine,
instructions above is a single high-level statement. Anybody trying to read or maintain
the spreadsheet would be greatly helped by knowing both this and what all the third-column
calculations have in common. Here, it is obvious; it may not be in larger spreadsheets.

Such analysis is even more vital when spreadsheets complicate their layout for visual effect.
Consider a buy-to-rent spreadsheet which performs the same calculation for a number of
properties, arraying them in blocks across the sheet, where each block calculates the profit \( P \) from
rental income \( R \), monthly mortgage payment \( M \), and other costs \( O \).

\[
\begin{array}{ccc}
\text{Property 1} & \text{Property 2} & \text{Property 3} \\
M & O & M \\
R & R & R \\
P & O & P \\
\end{array}
\]

Large spreadsheets of this kind can be confusing, especially when the blocks are split across
worksheets.

2.1 Representing structure in MM.

How can MM make structure explicit? MM programs contain "attributes" - arrays of elements -
related by equations to other attributes. In a freshly decompiled spreadsheet, each attribute is a
single cell. Structure discovery entails working out which cells can be grouped together into
arrays. Thus, our Income/Outgoings/Profit spreadsheet would look like this at first:

\[
\langle A1, A2, A3, B1, B2, B3, C1, C2, C3 \rangle \\
\text{where } A1=\text{"Income"}, B1=\text{"Outgoings"}, C1=\text{"Profit"} \\
C2=A2-B2, C3=A3-B3, C4=A4-B4
\]

but could have its attributes grouped and renamed to become more informative:

\[
\langle \text{Income\{1..3\}}, \text{Outgoings\{1..3\}}, \text{Profit\{1..3\}} \rangle \\
\text{where } \text{Profit\{all t\}} = \text{Income\{t\}} - \text{Outgoings\{t\}}
\]

Similarly, the Property one could become

\[
\langle \text{Mortgage\{Property1,Property2,Property3\}}, \text{OtherCosts\{Property1,Property2,Property3\}}, \text{Rent\{Property1,Property2,Property3\}}, \text{Profit\{Property1,Property2,Property3\}} \rangle \\
\text{where } \text{Profit\{all p\}} = \text{Rent\{p\}} - (\text{OtherCosts\{p\}} + \text{Mortgage\{p\}})
\]

Of course, the cells do not actually have names. However, one can try guessing plausible names
from neighbouring labels. Our spreadsheet algebra interface has functions for this, renaming
attributes and redisplaying the resulting program. The user can call these as many times as needed,
without committing to any set of names.

2.2 How can we discover implicit structure?

We are building up a library of structure-recognition heuristics (or "patterns" or "grammars") that
describe common structures and can be automatically matched against spreadsheets. We also allow
users to define new patterns, perhaps ones so specific that they apply only to one spreadsheet. This
work is described in the following two sections.

Before proceeding, we ought to say that we do not believe this to be the only approach to structure
discovery. More powerful methods exist, the ultimate surely being to learn patterns via inductive
logic programming [ILP]. We made some other suggestions near the end of our spreadsheet
algebra paper [Paine, 2004].

Markus Clermont has written an impressive program for discovering semantically-related regions
within spreadsheets [Clermont, 2003]. It searches for evidence of relatedness from a variety of
hints; for example, it might propose that cells mentioned together in a cell range or the argument to
a SUM should be grouped. Our approach does not pretend to be as comprehensive, but probably
shades into Clermont’s. Our heuristics are coded in Prolog, which can perform any computation
that any other language can, so our patterns could actually invoke any kind of search, including
those performed by his program.
3 PATTERNS, PATTERN LANGUAGES, AND GRAMMARS

In this section, we explain what we mean by spreadsheet grammars. To develop intuitions, we start with examples from other domains.

3.1 Pattern examples

Filename patterns

Unix and DOS, naturally enough, both allow filenames to be written in full, e.g. del scratch.tmp, copy ssi.xls ssi2.xls. But as a shorthand, they also allow patterns which match sets of filenames. Thus del *.tmp deletes all files with the .tmp extension, the asterisk standing for any sequence of characters. So we have symbols (the letters) that stand for themselves, but also symbols that stand for sets of strings.

Regular expressions

Regular expressions were invented by mathematician Stephen Cole Kleene, as a notation to manipulate "regular sets", formal descriptions of the behaviour of finite state machines. Today, they form an indispensable part of Unix commands such as "grep", which searches for a string in a set of files. The following examples demonstrate some features:

- `a` The letter a
- `a|b` An a or b
- `[a-z]` Any letter between a and z
- `([a-z])*` Any number of such letters
- `[a-z] ([0-9])*` One letter followed by any number of digits between 0 and 9

Once again, we have symbols that stand for sets of strings. We also have operators that combine patterns into bigger patterns: `|` makes the "or" of two patterns, and `*` following a pattern repeats it indefinitely.

Snobol patterns

Snobol [Griswold et. al., 1971] is a string-manipulation language written by David Farber, Ralph Griswold, and I. Polonsky of Bell Labs - and, incidentally, with a silly derivation for its name: StriNg Oriented and symBOlic Language. As these examples, which do the same as the regular expressions, demonstrate, there is more than one way to write patterns:

- `"a"`  
- `"a" | "b"`  
- `any(\"abcdefghijklmnopqrstuvwxyz\")`  
- `arbno( any(\"abcdefghijklmnopqrstuvwxyz\") )`  
- `any(\"abcdefghijklmnopqrstuvwxyz\") arbno( any(\"0123456789\") )`

3.2 Grammars

All the above patterns are grammars. A grammar is a formal definition of the syntactic structure of a language, normally written as rules which specify the structure of "phrases" in the language. Each rule has a left-hand side naming a syntactic category (for example, "noun phrase" or "verb" below) and a right-hand side defining it. The right-hand side can contain "terminal symbols" which stand for themselves, like the letters in the filename patterns, and "non-terminal symbols", which name other rules. There are no examples of these above. It may also contain operators for combining patterns, such as the regular expression "|" and "*", or the Snobol ",", "any" and "arbno". The example below is a grammar for a fragment of English:

- `sentence --> noun_phrase verb noun_phrase?`
- `noun_phrase --> proper_noun | determiner adjective* noun`
- `proper_noun --> "Mary" | "John" | "Fido"
- `noun --> "apple" | "ball" | "cat" | "fish"
- `verb --> "bites" | "eats" | "kicks" | "loves" | "sees"
- `adjective --> "big" | "brown" | "small"`
3.3 Spreadsheet grammars

From these examples, it seems reasonable that a grammar for describing parts of spreadsheets would let us name and invoke grammatical rules, and would have operators for combining elements within these rules. It might not need terminal symbols, since, at least for general-purpose rules intended to apply to many spreadsheets, it's unlikely that we would want references to specific labels or formulae.

A big difference is dimensionality. Spreadsheets can go down a column and across worksheets as well as along a line; the grammar will need operators to state where one element lies relative to the next.

What about content? Suppose we have a spreadsheet laid out as follows (this one is from a project on modelling house prices). Each row consists of a label followed by a cell:

Household Income  cell
Interest Rate     cell
Population        cell

Using the same notation as before, we could say the rows follow the grammar:

row --> label cell

Now, suppose the spreadsheet had been in columns instead of rows:

Household Income   Interest Rate   Population
cell               cell            cell

We could now think in terms of columns, describing each by the rule:

column --> label DOWN cell

We have introduced a "DOWN" operator to indicate the need to go down rather than along.

Somewhat more complicated is our Income/Outgoings/Profit spreadsheet, where each column contains three cells headed by a label. We could describe its columns as:

column --> label DOWN cell DOWN cell DOWN cell

or, since programmers loath writing anything more than once if they can devise a way to automatically repeat it, we might introduce a "repeat N times" operator:

column --> label (DOWN cell)*3

This rule is actually a special case of a more general rule which applies to many spreadsheets and says that a column is a label with any number of cells beneath:

column --> label (DOWN cell)*

Once more, we use "*" without a right-hand argument to denote indefinite repetition.

Finally, how about the Property spreadsheet, the one that's split into blocks, with attributes splattered across the worksheet? There are several possible descriptions. One is to start with the blocks as structural units:

spreadsheet     --> (block ALONG(12))*3
block           --> label DOWN
mortgage         --> cell
other_costs      --> cell
rent             --> cell
profit           --> cell

Here, we write an explicit ALONG operator, analogous to DOWN, rather than using implicit concatenation as we did in our first example. One advantage of this is that we can give the operator
an argument, so that ALONG(12) means "go along 12 cells". Another equally valid description would be to take the structural units to be attributes:

- spreadsheet → mortgage_parts AND other_costs_parts AND rent_parts AND profit_parts
- mortgage_parts → DOWN (cell ALONG(12))*3
- other_costs_parts → DOWN ALONG(3) (cell ALONG(12))*3
- rent_parts → DOWN(2) (cell ALONG(12))*3
- profit_parts → DOWN(3) ALONG(6) (cell ALONG(12))*3

DOWN now has an argument too, for when we want to move down more than one row. We have also introduced an AND operator, which superimposes elements without moving.

What have we achieved in this section? We have adapted ideas from pattern-matching languages and Prolog DCG grammars to describe structure in spreadsheets. Some descriptions will apply to many spreadsheets; others, such as the Property ones, will be more specific. In the following section, we indicate how we implement this in Prolog.

4 LOGIC PROGRAMMING AND PROLOG

4.1 What is logic programming?

Logic programming languages are quite different from languages like C and Fortran in which one gives the computer a sequence of instructions - read data, assign to a variable, print variable - which it has to follow. In logic programming, by contrast, the programmer writes statements in logic that describe the properties that the solution must have.

By analogy, the C or Fortran programmer writing a program to build a house would code instructions telling the computer to pick up a brick, lay it on another brick, put a window next to it, and so on. The logic programmer, however, would write logical statements describing what it means for something to be a house - there are walls, which are parallel to one another and perpendicular to the ground, and composed of bricks in one of several patterns, and may contain a window, and other things. Compare how, later in this section, we use Prolog to find the labels in a spreadsheet. Our program is not a set of instructions, but a specification of what it means for a cell to be a label.

Logic programs consist of "facts" or logical statements. Some are unconditionally true; some depend on the truth of others. To run a logic program, one gives it a "goal": a logical statement which it must prove from the facts. We start with one of the simplest possible examples of Prolog, namely these two facts:

```prolog
mortal(X) :- man(X).
man(socrates).
```

The first says that X is mortal if X is a man; the symbol ":-" means "if". In everyday language, all men are mortal. The second says Socrates is a man. So the first fact is conditionally true - it depends on whether its subject is a man - while the second is unconditionally true.

To run this program, we ask Prolog the goal:

```
?- mortal(socrates).
```

Prolog answers by finding the first fact, binding X to 'socrates', inferring that Socrates can be proven mortal if he can be proven a man, finding the second fact, thus proving him a man, and finally replying 'yes' to the goal.

More interesting are goals that don't merely check the truth of some statement, but find values for variables. If we ask

```
?- mortal(Y).
```

Prolog replies that Y=socrates.

4.2 Prolog
Prolog was the first widely used logic-programming language, and although others exist, it is still by far the most popular. There are some very good implementations around: we recommend SWI Prolog [SWI], which is efficient, free, and runs on Unix and Windows. This is what we are using in this project.

### 4.3 Analysing spreadsheets in Prolog

How can we apply logic programming to spreadsheets? In logic programming, we think in terms of predicates. “X is mortal” is a predicate we used above, as is “X is a man”. Other predicates from everyday life include “X is inside Y”, “X has colour C”, “X costs P pence”, “X is between Y and Z”, “X is employed by B in department D at salary S”, “the number I is less than the number J”, “the set S contains element E”. We see that many predicates express properties of a thing, or relations between things.

With spreadsheets, these things will include cells and cell ranges. We shall want to talk about properties of cells – “cell C contains formula F”, “cell C is empty” – and relations between cells – “cell C depends on cell D”, “cell C is in the same column as cell D”, “cell C appears to be a copy of cell D”, “cell C appears to be a label”. We might also want predicates on bigger pieces of the spreadsheet – “column C appears to consist of cells which are all copies of one another”, “the cells between (X1,Y1) and (X2,Y2) appear to be part of the same table”. Note the relevance to structure discovery.

Some of the spreadsheet predicates will be derived directly from the spreadsheet. For example, the facts defining “cell C contains formula F” are just an enumeration of each cell’s address and contents. As part of this project, we have written a Visual Basic program that dumps a spreadsheet in this form, so that it can be read into Prolog as a list of facts.

Other predicates will be defined in terms of these. The predicate “cell C depends on cell D” can be defined, roughly, like this:

```prolog
cell C depends on cell D if
  cell C contains formula F and
  F contains a subexpression SE and
  SE is a reference to D.
```

The predicate “cell C appears to be a label” can be defined as:

```prolog
cell C appears to be a label if
  cell C contains formula F and
  F is a string and
  there is no cell C' that depends on C.
```

where the final qualification allows us to exclude strings that are part of some calculation.

Given such definitions, we can ask Prolog to find values of variables that satisfy such predicates, thus finding (with the above examples) cells that depend on one another or are labels. We said that we might also want predicates on bigger pieces of the spreadsheet, such as “column C appears to consist of cells which are all copies of one another”; and we could equally well define and query these. We leave further discussion of Prolog to the Web version of our paper, which extends this section with a tutorial on its use for spreadsheet analysis. For a general introduction to Prolog, we recommend [Bratko, 1990] and [Sterling and Shapiro, 1994].

As far as structure discovery via grammars is concerned, we merely note that grammars are very closely related to predicates, and that logic programming has developed various techniques for implementing grammars by translating them into predicate definitions. These are well known amongst practitioners, and are introduced in the books just cited. We used them in the current project. Before we could do so however, we needed to invent data structures suitable for
representing spreadsheets and their structure. That is the topic of the next section.

5 ARROWS AND REVERSE ENGINEERING

Before we could implement the previous section’s structure-matching predicates, we needed a way to represent spreadsheets and the implicit structure we find. As Section 2.1 shows, MM gives us a way to do so. However, we still had to implement MM programs in Prolog, in a way that makes it easy to transform to and from spreadsheets, search for structure, and retain information about spreadsheet layouts. This section, which requires slightly more maths than the rest of the paper, describes how we did this. We believe it is an elegant and concise way of formulating reverse engineering for spreadsheets.

Let us begin by imagining one worksheet of a spreadsheet in value view. This is, in effect, an array whose indices are cell addresses and whose elements are values, i.e. numbers, strings, and other data. Mathematically, we can consider it a function from cell addresses to values.

Value view is boring, so let's switch to formula view. Now, each cell maps to a formula or expression, and the worksheet becomes a function from cell addresses to expressions.

MM views the world as a collection of attributes, each attribute being an array. These too can be considered functions, from the attribute’s indices to MM expressions.

An MM program, then, is modelled as a collection of functions, one for each attribute. Mathematically, we regard it as an indexed function, the indices being the attributes.

Compiling an MM program translates it into a spreadsheet; i.e., translates this indexed function into a single function from cell addresses to Excel expressions.

Now, before we can compile an MM program, we need to know how each attribute maps onto the spreadsheet. In our Income/Outgoings/Profit spreadsheet, each attribute mapped onto a column; in the Property spreadsheet, the attributes (Rent, Profit and so on) mapped onto cells staggered across the worksheet. In general, the compiler will assume a default mapping for each attribute (it allocates attributes to columns headed by their name), but this can be overridden by the user. In any case, MM clearly needs to know what the mapping from attribute to spreadsheet will be.

We now see that compiling an MM program becomes a matter of taking the attribute-to-expression functions, pre-composing with the attribute-to-spreadsheet mappings, and forming the union of the results. To generate a spreadsheet output file, we iterate over the domain of this union arrow, pumping out each element and its image - i.e. the cell address and its contents.

Decompilation goes in the opposite direction, from the spreadsheet-to-expression function to a set of attribute-to-expression functions. The trial and error of spreadsheet algebra entails finding a decomposition optimal for intelligibility; structure discovery infers appropriate domains for the attribute functions. In general, there is no unique decomposition, hence the need for trial, error, and human assistance.

Our latest version of MM takes this model literally, and represents the functions as Prolog data structures. We note for Prolog programmers that we found this good for manipulating functions in a representation-independent way, without needing to worry about whether they are stored as facts (clauses) or as association lists, trees, and so on.

Finally, many branches of mathematics like to consider functions as "arrows". Amongst other things, this gives us, via "commutative diagrams", a handy graphical way to depict them, and enables some proofs to be done largely by drawing. We found this very useful, and think likewise of our functions as arrows.

6 IMPLEMENTATION
As mentioned in Section 4, the current MM is coded in SWI Prolog. Advantages over Kawa, in which we coded the previous version [Paine, 2004], include: easier to generate .exe files for; probably less memory-heavy; the syntax is more readable, allowing one to define new operators, so making it easy to prototype notations for patterns without writing a parser; the TLI’s query interface also helps rapid prototyping. And logic programming is wonderful. SWI Prolog may not be as portable as the Java-based Kawa, but it does run on all the machines we’ve needed so far.

6.1 The user interface

Ideally, our system would just ask the user for a spreadsheet, and would then completely automatically match it against a library of structure-discovery rules, producing that MM program which best reveals the structure implicit in the spreadsheet. However, such complete automation is unlikely to be feasible. The user will often be able to contribute additional information about structure, or may want to guide the system toward a particular form for the program. This is why we developed the trial-and-error computer-algebra style interface of [Paine, 2004].

The present system will implement the same interface, together with a means of entering structure-discovery patterns which can be matched against a spreadsheet to suggest how cells may be grouped into attributes, and what names these attributes might have. The suggestions will be data structures which can be passed as arguments to the spreadsheet algebra functions. So, for example, if we are working on a particular spreadsheet, we might match it against our patterns and be told that the non-empty cells in column C all seem to be part of one attribute, named “Profit”. We could then pass that information to a function that merges these cells into the designated attribute, renames all references, and displays the result.

We are starting with a text-only interface, based on Prolog’s top-level interpreter. This is the part of a Prolog implementation to which the user gives Prolog goals to prove, getting back their truth or falsity, plus the values Prolog finds for variables. We are augmenting this with our Grips pre-processor [Paine, 2003], which allows goals to be written as function calls rather than predicate invocations. This makes certain kinds of goal more compact and easier to write.

Once we have the text-only interface working, we shall prototype a graphical user interface. We shall do this as a Web application where MM will run on a Web server, the user operating it via a browser. This is less interactive than a windowing interface, but easier to prototype, since the browser takes over the grunt work of rendering. We have taken advantage of this before in our Spreadsheet Autopublisher [Paine; Paine and Ramsden, 2002], a Web-based application which automatically converted simple spreadsheets or MM programs submitted to it into other applications runnable over the Web.

6.2 Specificity versus generality

We have to decide how much the user is allowed to do. Can they invoke any Prolog predicate on the spreadsheet, even defining their own if they know how? Or do we restrict them in some way?

We shall certainly provide access directly to Prolog, if only because we shall need it ourselves when debugging. However, although Prolog is closer to logic than many other languages, it is not a complete implementation, and Prolog programmers need to learn a number of non-obvious tricks before they can write predicates that are guaranteed not to loop indefinitely, generate alternative answers that are never needed, or handle negation incorrectly. To avoid users having to become skilled Prolog programmers, we will therefore provide a library of elementary predicates together with operations for combining these, and perhaps some limited facilities for defining new ones. The combining operations will be built on top of Prolog’s built-in logical operators, in much the same way that the AND, DOWN and ALONG operators in our spreadsheet grammars could be. This will enable us to hide some of the messy details of Prolog from the user, and will also give us the freedom to reorder patterns to make them more efficient.

6.3 Efficiency
This brings us to efficiency. In general, our approach is “make the program correct first, then
elegant, then efficient”. There are too many languages and systems around – perhaps Java is one –
designed with no thought for fundamentals. And computers are still getting faster. I can now carry
more computing power in my rucksack – complete with SWI Prolog compiler – than Oxford
University had to share between all my students when I first started teaching Prolog.

Having said this, we do need to decide what to do if our pure-Prolog approach is infeasibly slow.
One point is that many Prologs contain features to speed up the search for appropriate facts –
“clause indexing”, as Prolog programmers know it. For example, Dennis Merrit of Amzi! [Amzi!]
has told us that Amzi! Prolog implements indexed dynamic database predicates using B-trees, and
that this gives them very good response times with the large WordNet libraries (WordNet is an
online dictionary which models some features of human linguistic memory), which have 170,000
clauses in the larger predicates.

If such features turn out to be inadequate, we may be able to implement our own. All decent
Prologs have a “foreign language” interface, allowing them to call code written in C and other
languages. By using this, we could define our elementary predicates in C (say), optimising them to
the hilt but still making them look to the user like Prolog, so that the non-logic-programming
aspects are safely hidden. We have done this before when teaching Prolog. One of our students
wanted to write a limerick generator, whose output would rhyme, scan, and even be grammatically
correct. We had a machine-readable dictionary of about 70,000 words, whose files gave the
pronunciation, scansion, and parts of speech for each word. We converted this into an indexed file,
wrote some indexed-file-search routines in Fortran, and then interfaced them to our Prolog system
(which in those days was part of Sussex Poplog) using Pop-11. The result was that the student
could ask Prolog goals such as “find all nouns which rhyme with ‘chimney’”, or “find all verbs
which are 8 syllables long”. The goals behaved as if defined in Prolog, but were actually doing a
fast search implemented by the operating system’s indexed sequential files.

6.4 Progress so far

The most important part of our work was getting the representations described in Section 5
(Arrows and Reverse Engineering) right, as this gives us a proper mathematical foundation for
everything else. That is now done, and we have implemented the representation in Prolog and
tested it with a variety of structure-discovery patterns and spreadsheet algebra functions, using a
Visual Basic program to dump spreadsheets as Prolog facts.

Given that we have the Prolog top-level interpreter and our Grips pre-processor (Section 6.1), most
of the text-only interface exists, although we are still deciding on the most appropriate elementary
predicates to give the user. We have also started work on the Web interface, which we are
implementing as PHP [PHP] scripts that call SWI-Prolog.

We have not yet hit any efficiency problems – in our tests, the largest spreadsheet has had about
100 occupied cells out of 200. However, as mentioned above, we do have some ideas on how to
attack such problems should they arise.

7 FUTURE WORK

We need to experiment with alternative ways to write patterns. Once the notation is good enough
to be fixed, we would like to see researchers collaborate in building up a shared library of structure
patterns.

7.1 Intelligent structure monitoring and flexible best practice

Our goal is to run MM as an “intelligent structure monitor” alongside Excel. This could
revolutionise spreadsheet "best practice”. We have said already that different spreadsheet tasks
require different layouts. A layout convenient to a user entering data and reading off results may
be much less convenient to the maintenance programmer. The problem is that all guidelines for
best-practice demand a single layout. But no layout can be simultaneously optimum for tasks as
different as data entry and maintenance. The answer is to move from a view of spreadsheets as
single unchanging entities to a relativistic approach where they can freely adapt to the user's needs.
8 ACKNOWLEDGEMENTS

We would like to thank one of the anonymous referees for comments, and the Excelsior café in Cowley for coffee.

9 REFERENCES

Spreadsheet good practice:
is there any such thing?

David Colver
Operis, 8 City Road,
London EC1Y 2AA, United Kingdom
dcolver@operis.com

ABSTRACT

Various techniques for developing spreadsheet models greatly improve the chance that the end result will not contain basic mechanical errors. However, for every discipline in which a given technique is useful, there is likely to be another in which the same technique works badly. As a result, the author urges that EuSpRIG does not succumb to internal or external pressures to champion a particular set of "best practices", because no such set is optimal in all spreadsheet applications.

1. APPROACH

In this paper, I am going to show some spreadsheet development ideas, which I find valuable. They are ideas that I believe in: they are methods that my company, Operis [Operis, 2010], uses when it develops financial models, and they are ideas that we teach to fee-paying customers when they come on the courses we provide. (Operis is by a long way the largest provider of financial modelling training, with over 10,000 man-days of courses attended in the last seven years.)

After I show you each idea, I am going to rubbish it.

My aim is to show you that what works well in one set of circumstances works badly in others. My goal is to discredit, once and for all, any temptation by insiders or calls from outsiders for EuSpRIG to set down some best practice guidelines for spreadsheets, despite the fact that such advice already exists [Read & Batson, 1999][Bewig, 2005][Raffensperger, 2001]. With very few exceptions, there is in my opinion no such thing as universal spreadsheet good practice. This opinion is supported by Grossman “Best practices are situation-dependent” [Grossman, 2002].

2. WHERE TO BEGIN

The first principle that Operis teaches is where to start when building a model. "If you fall asleep for the rest of the day, or if you don't absorb anything else while you are here, please try to remember the next sentence. Design the output first."

Designing the output means laying out the reports that your model is going to produce. In many financial models, this means setting out a cash flow, a profit and loss statement, a balance sheet, and a host of supporting schedules. But such models are by no means the only applications for spreadsheets, and users of spreadsheets in other disciplines will have different sorts of reports to produce.

These mocked up reports don't have any numbers in them at this stage. But they do have all the headings, neatly set out; and they do have the margins set properly, and the headers and footers, and are laid out so they fit on the page properly.
Setting these reports up is a good first step:

- Its work you are going to have to do eventually anyway; you might as well do it right away.
- It allows you to show your client, your boss and your colleagues what you intend to produce; if you have misunderstood the brief, they can probably tell you right away.
- It gives visible evidence of progress within a couple of hours of starting the project.
- If the client turns up unexpectedly, you can print out a neat version of the model in moments, and have something professional looking to show.

Most of all, the reports serve as an informal specification for the project. Completing the model becomes a simple matter of filling the lines in, one at a time, so the output serves as a route map for the assignment.

Counterargument

Designing the outputs first will indeed act as an informal specification for the project, but not all projects can live with such informality. Some projects are so important that they need quite formal specification to be negotiated with the project sponsors. In fact, a mock-up is not a specification as a traditional systems developer would have it. It is a non-functioning prototype, which is not the same thing at all.

Prototyping is itself a valuable tool in systems development, but banging together a mock up and seeing if the client likes the look of it is not the way we would build the systems that control a nuclear power station or fly an aeroplane. Where is the test plan? What are the acceptance criteria? How will we know that the spreadsheet gives the right answer when we see it? Many development methodologies teach that these are the areas to start with, not a rush to knock out a non-functional piece of Excel.

Even in circumstances where a formal specification is unnecessary, it is open to question whether designing the output first is a good thing to do. If your model produces standard financial statements, you can visualise what those will look like with a high degree of accuracy. But someone using a spreadsheet to develop some kind of mathematical or scientific theorem will probably go down many blind alleys, and be forced to tidy up his work so that it is in an acceptable state presentationally when he eventually does reach some kind of conclusion.

3. MODEL DESIGN: THE WORKINGS

The outputs we have mocked up don't do anything useful on their own because we haven't yet put any figures in them. By the time we have finished building a model, there will be figures, generated by spreadsheet formulae. But where should we put these spreadsheet formulae?

In deciding this, the principle we follow is that we wage war on long formulae. Any formula that is not so trivially short that it can be understood at a glance should be broken up and laid out over several lines. If you do that, though, to formulae on the output worksheets we have already prepared, you will tend to ruin their carefully prepared layout, and wind up with a mess of unwanted intermediate workings.

The method that Operis teaches is that the calculations of a model should be on a separate workings page. That worksheet can take as many lines as it likes to calculate arbitrarily complicated results in easy steps. The conclusions from that work are then reported on the output sheet.
Besides providing the means to break up long calculations into short formulae without compromising the reports, separating the workings from the outputs has a number of practical advantages:

- Related items are calculated next to each other. The reporting of the expenditure on equipment, depreciation of that equipment and the resulting net book value of that equipment are reported on the cash flow, the profit and loss statement and the balance sheet, but the calculation of these intimately related items can be set out right next to each other. Formulae mention cells that are close at hand, which makes them easy to follow.

- If we placed the calculations on the reports, the formulae would have to operate in three dimensions. Besides being harder to read, such formulae are dramatically slower to calculate. On real world models, having the calculations on a single large workings sheet is about 5-10 times faster than dispersing them over many sheets.

It also in my view has a theoretical advantage: the way to calculate some results, and the way to display them, have nothing to do with each other. There is no other programming environment that I can think of that would contemplate mixing these items. Good system designs emphasise encapsulating design details, not jumbling them up.

Once we have our model written, we will start putting it to good use. One of the things we will want to explore is what happens if we take a different approach to the project being modelled. The way many people do this is to make copies of the workbook. And then overtype the numbers in it.

Quickly, you have many workbooks with many scenarios in them. Those scenarios are quite likely to exercise the model rather more vigorously than it has been exercised before, and expose various small bugs in it. Repairing them is difficult to do reliably, because you have many copies of the model that need identical repairs.

Our solution to this is to avoid having lots of models. We have just one model, with lots of scenarios in it. Each scenario has an input sheet of its own. We can switch between the scenarios very easily, by arranging for the workings sheet to take its data from one or other of the sheets. We normally provide a little menu option to do this, but in practice it activates some macro code of trivial proportions.

To sum up:

- we separate the inputs from the workings so that we can practice easy scenario management
- we separate the workings from the outputs to wage war on long formulae.
Counterargument

Separating the inputs, workings and outputs of a model has a nice, theoretical elegance to it, and it maps neatly on to how large scale data processing systems have worked for several decades. But spreadsheet models aren't large data processing systems: they are decision tools that work on devices that are not called personal computers for nothing.

By separating the inputs, workings and outputs you cause some lines in the model to appear three times over, once in each place. For someone who is not a proficient modeller, experienced spreadsheet user or even a fast typist, that is quite an overhead.

As well as making the model time consuming to develop, your method makes it harder to use. If someone doesn't like the figure reported by the model on one of the output sheets, they have to fathom out which inputs are responsible for it, and go and alter them appropriately. Those inputs are on a different worksheet, and quite hard to find. When naive spreadsheet users see a number they don't like, what they want to do is just type over it with a figure they prefer. With the inputs, workings and outputs all jumbled together, you have some chance of letting them do that. With these elements separated rigidly as you do, users of your model have to do something much less intuitive.

There is a particular circumstance where this issue really matters. A decade ago, models were used to structure a deal, and then thrown away when the paperwork was signed. Now the banks expect to see them updated every few months. Incorporating the historic actuals as they unfold is easy in a model that doesn't separate inputs from workings or workings from outputs: you just overtype formulae with real numbers. With the structure proposed in this paper, you are going to have to do something much more cumbersome to get historic values through the model from inputs to outputs.

There is one class of forecaster for whom this is a real pain. A broking analyst, paid to follow companies and express opinions about whether to buy or sell their shares, will have a spreadsheet for each firm that he is tracking. It will have historic figures, which then give way to projections. Every so often, perhaps once a quarter, the firm will announce some new results. He needs to update his spreadsheet with the new figures. In many cases, he will get two sets of figures: a press release with headline numbers, followed by the full accounts some weeks later.

For a while, therefore, his model has columns that are wholly historic; columns that are wholly projected; and one column that is a hybrid - see Figure 2. It has some actual numbers provided by the client, and others that have yet to be published and so must be derived. That's easy to do with a single worksheet in which inputs, workings and outputs are all the same thing; it is rather hard if the inputs, workings and outputs are rigidly separated.

![Figure 2 – Historic, Hybrid & Projected Columns](image)
The models Operis prepares concern themselves with large national infrastructure projects: roads, railways, hospitals, prisons. Financing them can take months, and during that time you do want to evaluate alternative scenarios. But the functionality of the model doesn't change much; all it is doing is applying the rules of double entry bookkeeping and calculating a few financial ratios. We can say that the logic of the model is stable, but the data is shifting.

Not all models are like that. A young associate in a bank's mergers and acquisitions department will often be asked to study the outcome of a merger between two companies. He will get the data from the companies' accounts. In the US, quite decent accounts appear every quarter, but in Europe, half-yearly accounts are more typical, and the mid-year report is very thin. The figures the banker has to work with therefore don't change at all for a whole year. With those figures he will prepare an analysis of what happens if company A buys company B. If that does not look sensible, he will wonder what happens if company B does a reverse takeover on company A; or if the two genuinely merge, or a new company is set up to buy the pair of them. Each of these transactions is fundamentally different and leaves him tearing his model down and rebuilding it.

In his case, it is the data of the model that is stable, but the logic that is shifting, quite the opposite situation from the Operis model. Separating the inputs from the workings to allow neat scenario management is a hindrance, not a help, in his circumstances.

4. AUDIT TESTS

Separating the workings from the outputs is a key tool in waging war on long formulae, but it does bring with it one notable drawback. It is possible to complete the most perfect of computations in the workings area, only to misreport it among the outputs. An example of such a mistake would be to leave a line out of the balance sheet.

My fix to this is to have a fourth kind of sheet, an audit sheet. Anything we can test on that, we do. There are obvious tests, like whether each subtotal adds up correctly, and—whether the balance sheet balances. And there are less obvious tests, such as an exact reconciliation of the total of the inputs sheet to the principal outputs.

A typical model shipped by Operis will have a 25% audit overhead, that is, an audit sheet that is a quarter of the size of the workings sheet. It's amazing how frequently the tests save our bacon. When asked to add some new capability to a model, we will do it diligently and methodically and with much skill and care. When we think we are done, we look at the audit sheet, and almost always find that there are three or four audit tests that are now failing. Without the audit sheet, we would have been more likely to ship the spreadsheet with those faults in it.

Counterargument

Putting copious self-testing into a financial model is such an obviously good thing that it is hard to imagine anyone not liking it. But there are people who object to them: they are professional spreadsheet auditors.

From his point of view, all the audit sheet does is introduce code to a spreadsheet that adds nothing to the outputs. It's code he has to make a choice about. Either, he can check that the tests do what they purport to do, which will take time and duplicate much of his own testing. Or, he can mention in his report that he is excluding them from his scope of work, which looks rather peculiar. In our experience, about half the model auditing practices implore us to remove the audit sheet from the versions of the model that they are asked to look at.
5. NAMES

We think of spreadsheet formulae as using a coordinate notation to involve other spreadsheet cells in a calculation. But spreadsheet formulae don't have to be expressed in terms of formulae; they can be expressed in terms of more meaningful names. It is entirely possible to build substantial models without a single coordinate being present in any of the formulae.

There are all sorts of practical advantages to using names.

- The names make the model easier to read. This is particularly true if your organisation follows a standard naming convention, as Operis does.
- The column matching is automatic. So long as you line the different worksheets up, so that column Z refers to the same time period throughout the model, then a whole class of error, the ones involving picking up figures from the wrong column, is eliminated.
- Formulae that refer to cells a considerable distance away, or on another worksheet, become much easier to read.

Names promote code reuse. An analyst developing a spreadsheet that concerns a project in Mozambique can save himself the bother of coding the tax regime in that country if he can lift it out of an existing model. A straight copy from one spreadsheet to another is most unlikely to work without adjustment if the formulae are expressed in terms of coordinates. It is unreasonable to expect the two spreadsheets to be coordinate-consistent; if they were, they would be the same spreadsheets, and there would be no point writing the second one. But it is quite reasonable, in an environment that uses names as a matter of course, to expect the two spreadsheets to follow a consistent naming convention; and if they do, the code fragment has every chance of working first time when it is copied.

The best modellers have libraries of code fragments waiting for reuse in this fashion, the most used of them having being subject to independent audit dozens of times as components of finished models. Often these libraries are supplemented by mechanical systems for coupling the fragments together into models.

There is also a theoretical appeal to using names, in my view. The solution to a problem has nothing to do with the way it happens to be laid out on a page. Einstein wrote E=mc2; he did not write that the quantity he has defined in the second paragraph of page two of his paper is equal to the one written about three lines above, multiplied by the square of the quantity he wrote about in another paper two months earlier. Nor should a modeller express his solution in terms that are dictated by how he happens to have laid it out on the page (and which will alter if a few lines are inserted or deleted for cosmetic reasons). This point is very similar to the justification offered for separating the workings of a model from the outputs: the calculation should not be influenced by how the results happen to be laid out.

Counterargument

Operis certainly makes use of names in its own models, and shows in its courses how to work with them, but it is much the least popular part of its training. There are many organisations which buy in to the Operis way of building a model in every respect save one: they don't bother with the names. Operis has lost some modelling assignments because the prospective client has preferred not to receive a model peppered with names. So much so, that the latest version of OAK, the Operis Analysis Kit, a spreadsheet development and audit tool, actually includes a facility for reversing the naming process, restoring formulae to the coordinate representation.
Why are names so unpopular? There are some objections to them.

- They take time to define. You need to be certain of some payback before making that investment.
- The expression Sheet !A1 refers unambiguously to a particular cell. The cell Revenue could refer to any number of cells on any of the worksheets. (Or it could refer to something else altogether.) How do we know that Revenue is correctly defined? Misdefined names give rise to a whole class of error that does not exist at all using the coordinate notation.
- What I am referring to here is functionality in Excel. It is not present in other spreadsheet packages, and supported only lightly in packages that purport to read data in the Excel file format. Using these features reduces portability to other platforms.
- Microsoft itself has done a number of things to undermine support for names in recent versions of Excel.

But I think the main reason that names are disliked is that people don't know how to manipulate them. They are not taught. It would only take half a day to learn; but if even a small number of consumers of a spreadsheet are prevented from altering the model by the use of a technique that is unfamiliar, the technique becomes less acceptable to the whole organisation in certain circumstances. For them, meaningful names have actually made the spreadsheet less useful than meaningless coordinates.

6. UNITS

One of the most basic errors that can be made in a spreadsheet is to combine inappropriately cells which have incompatible units. One cell may be in thousands of pounds, and another in millions: they can't be added together without adjustment. Some methodologies urge that every quantity is systematically labelled as to units.

My own approach is to urge that all spreadsheet quantities should be converted at the earliest available opportunity from whatever inputs they happen to be entered into base units. It doesn't matter what the base units are, as long as they are consistently applied throughout the model. If you are sure that quantities are in base units, you can combine them at will without worrying about multiplying or dividing by conversion factors.

My own choice of base units is invariably dollars, euros, pounds, tonnes, barrels (of oil) and so on. This gives the greatest flexibility, because the output can be shown in thousands, millions or billions merely by reformatting the spreadsheet, and so can be adapted to suit different audiences.

Counterargument

This flexibility is great for a professional spreadsheet programmer but having numbers display differently from the underlying value is thoroughly confusing for non-expert users of the spreadsheet and introduces a more dangerous possibility of error than it cures.

Even the expert can be confused when the data is exported. Copying to other Excel spreadsheets, or to, for example, Access, preserves the unformatted value. Copying to Word, or writing the data out as CSV or TXT files, gives the formatted version.
7. SPACES IN FORMULAE

Let me now mention one thing that Operis does not teach, but which was mentioned at EuSpRIG last year. We were urged to put spaces in our formulae to make them more readable.

**Counterargument**

If asked what operators are available in Excel, most people will correctly identify the mathematical operators, +, -, *, / and don't forget ^ and %.

Pushed a little harder, seasoned spreadsheet users will recall the comparison operators, >, <, = and their combinations, and the concatenation operator & for text.

But actually there are three other operators, colon, comma and space:

- The colon is the between operator: it constructs a range which includes all the cells between two other ranges.
- The comma is the union operator: it constructs a disjoint range which is composed of the components of two other ranges.
- The space is the intersection operator: it constructs a disjoint range which is composed of the components of two other ranges.

Space characters are therefore not white space in Excel. In early versions of Excel, you could only put a space in a formula if you meant it to be the intersection operator (or, obviously, as part of a literal, in quotes). Later versions relaxed the parser so that it would accept redundant spaces in the middle of formulae.

Nevertheless, I would no more put surplus space operators into my formulae than I would put surplus minus signs into them.

8. CONCLUSION

We have looked at issues ranging

- from the macro ...: which order to write a model, how to structure it;
- …through the middling ... : whether to use names or not, what units to choose;
- ... to the micro issue: whether or not it is helpful to add spaces for readability.

In every example, we can produce good, coherent arguments for following a particular policy, and find real world circumstances in which that policy would be quite the opposite of helpful.

At previous EuSpRIG meetings, I have been very uncomfortable when speakers have promoted their favourite way of doing things as the definitive method, good in all circumstances. Developing models is an exercise in engineering. It involves trade-offs. You can buy a fast car, or a cheap car, or one that carries lots of people. You may even find a valuable blend of two of these things. But you won't manage to get all three qualities in one car. Similarly, you can build a model that is laid out formally, or relatively easy to change, or very accessible to naive users; but you won't get all of them.

The methods I use are the ones that work for me, that make the trade-offs (see Figure 3) most helpful in my field of work. I know of other fields in which they will work very badly. There is a relatively high fixed cost to setting out a model in a structured fashion, and to using names.
In our experience, the result is that the model scales better with complexity, but the spreadsheet needs to be long-lasting to get a useful payback (point A below).

For those building models that are simpler, or short-lived, the fixed cost is not worth paying (point B below).

Figure 3 – Development cost / Model complexity trade off

In closing, I urge EuSpRIG to be very thoughtful before it promotes any particular best practice. About the only techniques that I believe in as having a near universal application are:

- keep it simple: avoid overmodelling, or modelling beyond the resolution of your data
- have some idea of how you will know the finished article when you see it
- avoid long formulae
- don't use the OFFSET or INDIRECT functions
- introduce circular logic into a spreadsheet if you can explain the mathematical behaviour of the resulting function.

REFERENCES


When, why and how to test spreadsheets

Louise Pryor
115 Hanover Street, Edinburgh EH2 1DJ, UK
www.louisepryor.com

ABSTRACT

Testing is a vital part of software development, and spreadsheets are like any other software in this respect. This paper discusses the testing of spreadsheets in the light of one practitioner’s experience. It considers the concept of software testing and how it differs from reviewing, and describes when it might take place. Different types of testing are described, and some techniques for performing them presented. Some of the commonly encountered problems are discussed.

1 INTRODUCTION

Any software engineer will tell you that testing is a vital part of software development. It’s usually the only way to tell that a software system actually does what it is meant to. In this paper I discuss my experience of testing spreadsheets, the techniques I have used and the problems I have encountered. Many developers of spreadsheets have had little or no exposure to software engineering concepts, and may not be aware of some of the simple techniques that can be used. The emphasis is on the practical aspects of testing spreadsheets rather than the theory, and there is no intention of providing a comprehensive review. My experience is primarily based on financial modelling spreadsheets, and the techniques I use may not be appropriate in other situations.

In the remainder of this section I introduce the notion of testing, compare it to code reviewing and describe when it might take place. The remaining sections cover the different types of testing, and I conclude with a section on common problems. Throughout the paper I discuss only the testing of spreadsheet formulae, omitting all consideration of macros and tools such as Excel’s solver and goal seek. Many of the techniques I describe are demonstrated in a sample Excel workbook available for download (Pryor 2004a). Some of them use XLSior, an Excel add-in I have developed to support good practice in spreadsheet development (XLSior).

1.1 What is testing?

Testing is the controlled execution of a spreadsheet, checking that what it does meets the specification. Typically this means using known inputs and checking the results against what would be expected from these inputs.

Testing is the only way to tell what the spreadsheet actually does. It is a vital step in the process of gaining confidence in the results (Pryor, 2004b). With a good testing process, tests are easy to run (so are run often), and logs are generated automatically, so that it is easy for a user to see what tests have been run and when, and what the results were.

1.2 Review is not testing

Testing is part of ensuring spreadsheet quality and complements spreadsheet review. The distinction between the two is an important one. A review consists of looking at the code
and trying to spot the errors, while to test a program you run it and look at the results. Both are necessary, and neither is likely to find all the errors on its own.

A truly effective spreadsheet review would follow the lines of the systematic code inspections originally devised by Fagan and described by Gilb (1988). Although effective, this is time consuming and requires more than one person. Moreover, spreadsheet code is notoriously difficult to review, including as it does

- Spreadsheet formulae and layout
- Data validation
- Conditional and other formatting
- Defined names
- Charts
- VBA
- Definitions for the solver, scenarios, data filtering, pivot tables and other tools

Automated tools can provide valuable help, but although their use is often referred to as “testing” they are limited to summarising information and detecting common sources of error, such as formulae not being copied correctly or numbers treated as text. These types of error are essentially syntactic. A spreadsheet that passes all the tests of an automated tool may still be incorrect if its semantics do not agree with its specification.

However thorough the reviewer, in the end they cannot do more than say that in their opinion running the spreadsheet will result in the correct behaviour. The end result of a review depends on the reviewer’s understanding of both the specification and, crucially, the implementation. Testing, on the other hand, may check only certain execution paths, but a successful test of those paths is much more objective than a review. Devising the tests requires some understanding of the specification, but a detailed understanding of the program logic is not necessary. The only way to tell whether a spreadsheet is producing the correct behaviour to run it; in other words to test it. Of course, the successful running of a few tests does not guarantee that the spreadsheet will produce the correct results for all possible inputs.

1.3 When to test

Testing can take place at different stages of the development process. The different types of testing are described in the standard software engineering books such as (Pressman & Ince 2000).

- **Unit testing** is the most detailed type of testing. Individual components are tested in isolation. Unit testing should take place frequently throughout the development process.

- **System testing** looks at the system as a whole, testing the final results. It should take place at a minimum when a spreadsheet is released for use, and preferably more often during development.

- **Regression testing** compares the results of a new version against those of a previous version. It is a specialised form of system testing, used to check that no unintended changes have been introduced.
• **Acceptance testing** is testing by the spreadsheet user (or on their behalf) when they receive the spreadsheet from the developer, to determine that it meets their requirements and is fit for use.

As usual with spreadsheets, these distinctions are less clear cut than with traditional software. The user is often the same as the developer, so there may well be no separate stage of acceptance testing. Because of the lack of modularity in spreadsheet design unit testing consists of testing individual calculations or tables of calculations. For a small spreadsheet, unit testing and system testing are essentially the same process. Acceptance testing may be performed by independent third parties, such as consultants who are asked to review a spreadsheet and pronounce on its fitness for purpose.

2 UNIT TESTING

The idea of unit testing is to make sure that individual calculations are correct. Unit testing is especially valuable when it can be performed automatically and often, as it can then be used to check for unanticipated side effects of changes (Astels, 2003).

2.1 Testing for invariants

A widely used form of unit testing is the cross checking of column and row totals; other similar tests include checking that a column of percentages add to 100%, or that specific values are always positive (or negative). These are tests for invariants; conditions that should always be true. They are run every time the spreadsheet is calculated, so the results are always up to date. It is also easy to summarise the results of these tests in one area of the spreadsheet, so that they can all be checked at once.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

Static tests

Sum of total cashflow = sum of individual cashflows

<table>
<thead>
<tr>
<th>Actual</th>
<th>Expected</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-150.00</td>
<td>3631.86</td>
<td>Pass</td>
</tr>
<tr>
<td>-150.00</td>
<td>3631.86</td>
<td>Pass</td>
</tr>
</tbody>
</table>

2.2 Using data tables

Most testing, though, requires that the spreadsheet be run with specially chosen input values, and the results checked against those that are expected. A number of useful tests can be performed by changing just one or two values on the spreadsheet. For example, if a discount rate is set to zero then the present value of a series of payments is the same as the sum of the amounts paid.

Data tables (in Excel) or multiple operations (in Open Office) can be used for these tests. They can be used to test several different calculations that depend on the same value, as in the example below. Depending on the calculation settings they are kept up to date as the spreadsheet is recalculated. The cell whose changing value is investigated by the table may contain a complex formula, allowing the isolation of a single calculation step.
The use of data tables in Excel has some disadvantages. First, the cell containing the input value must be on the same sheet as the data table. This means that you cannot have a single sheet (or set of sheets) that contain all your tests, but must scatter the tests throughout the workbook. Also, the presence of many data tables may slow down the spreadsheet calculation. In Excel you can set the calculation to Automatic (except tables), but this means that the tests are not automatically kept up to date, and there is no visible indication that this is the case.

<table>
<thead>
<tr>
<th>Wage infl</th>
<th>Constr wages</th>
<th>Op wages</th>
<th>max factor</th>
<th>min factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>-410.52</td>
<td>-4074.23</td>
<td>11.742721</td>
<td>1.002466</td>
</tr>
<tr>
<td>Expected</td>
<td>-400.00</td>
<td>-1749.00</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Result</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
</tbody>
</table>

The use of data tables in Excel has some disadvantages. First, the cell containing the input value must be on the same sheet as the data table. This means that you cannot have a single sheet (or set of sheets) that contain all your tests, but must scatter the tests throughout the workbook. Also, the presence of many data tables may slow down the spreadsheet calculation. In Excel you can set the calculation to Automatic (except tables), but this means that the tests are not automatically kept up to date, and there is no visible indication that this is the case.

<table>
<thead>
<tr>
<th>Wage infl</th>
<th>Constr wages</th>
<th>Op wages</th>
<th>max factor</th>
<th>min factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>-369.47</td>
<td>-4074.23</td>
<td>11.742721</td>
<td>1.002466</td>
</tr>
<tr>
<td>Expected</td>
<td>-400.00</td>
<td>-1749.00</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Result</td>
<td>Fail</td>
<td>Pass</td>
<td>Pass</td>
<td>Pass</td>
</tr>
</tbody>
</table>

2.3 Macros

A data table can be used either to test the effect of a single value on several results, or to test the effect of a pair of values on a single result. It cannot be used when the test depends on a large set of test data, such as setting whole ranges to zero, one, or other specific values.

A more flexible solution is to use a macro that will substitute in the appropriate values, recalculate the spreadsheet and record the results. The substitutions can be performed on individual cells or on whole ranges. For example, many of the spreadsheets I encounter perform their calculations in columns. The cells in a single column contain the same (relative) formula. That formula can be tested by substituting appropriate values in the columns forming the inputs to the formula. Often, setting the inputs to zero or one provides useful information. It is possible to test the validity of the calculations in each column using this technique.

I use the functionality provided by XLsior, which allows you to specify data substitutions that should be made and conditions that should then be true. It records the substitutions and conditions (and the results) for each test, providing an auditable record of the testing that has been performed.
XLSior also provides a summary of all the tests with their results. The tests do not update automatically whenever the workbook is recalculated, but must be specifically run. Because the records include a time stamp, it is easy to tell whether the test results are up to date.

A system of macros such as this allows flexible unit testing that is easy to reproduce. Whenever a change is made, it is easy to rerun all the tests and make sure that there are no unintended side effects.

### 2.4 Devising unit tests

The efficacy of testing depends on the tests that are used. When devising tests, you should try to break the spreadsheet. What might cause it to go wrong? In devising unit tests you are trying to isolate specific calculations or parts of calculations. For instance, suppose you have a stream of cash flows at irregular intervals. Tests might include making the intervals regular, and making all the cash flows the same amount. You should always try to test boundary values, for example setting interest rates to zero.

Other useful tests include making sure that the correct row (or column) is being found when using a lookup function. You can do this by setting the contents of the lookup table to known values, such as 11 in the first row and column, 23 in the second row and third column, and so on. Similarly, setting all elements except one of a range to zero can help in making sure that sumproducts are aligned correctly.

More generally, you should try to include tests that exercise every branch of a conditional, such as complex if statements or formulae involving maxima or minima.
3 SYSTEM TESTING

System testing involves testing the whole spreadsheets. Typically, you test a range of scenarios, covering both typical inputs and unusual combinations. To do system testing properly, you need an independent calculation of the expected results. Such independent calculations are, in my experience, rarely available and so full system testing is not often performed.

The only effective way of running reliable system tests is to use macros. The process is very similar to the regression testing described in the next section. The chief difficulty is determining what the correct answers should be.

4 REGRESSION TESTING

Regression testing is a special kind of system testing, in which the independent calculations are performed by a different version of the same spreadsheet (or sometimes by a precursor system). The chief aim of regression testing is to check that the results have not been altered by changes to the spreadsheet, or, where they have altered, to investigate the effects of the changes. It is important when adding new functionality to a spreadsheet, to make sure that the existing functionality is not affected. Regression testing on its own is not sufficient to test the correctness of a spreadsheet, as the older version against which the comparison is being made might itself contain errors.

Regression testing is really only feasible when the layout of the outputs does not change between versions, or when there are only a few outputs that need to be compared. It is more difficult to compare whole sets of calculations because of the difficulty of expressing the correspondences between the cells in the two versions. It is particularly useful for spreadsheets that are used to calculate results in a standard format, possibly for use in another program.

A typical set of regression tests would consist of a number of scenarios that cover a wide range of the possible inputs. The old and new versions are both run on each scenario and the results compared. It is often possible to reduce the time that the testing takes by running the old version through each scenario once and recording the results, although this is only necessary if the calculations are very slow.

I have usually found it necessary to write custom macros for each spreadsheet that is to be tested. The testing is usually performed in a third spreadsheet. It is made significantly easier if the important results on the spreadsheet to be tested are on a single sheet. I set up a testing spreadsheet with four sheets for each scenario: one to hold the input values, two to hold the results (one from each version), and one to hold the comparison between the results.

The macro I use usually has the following form

```
Open the old and new workbooks
For each scenario
    Copy the inputs to both versions
    Recalculate both versions
    Copy the results to the results sheets
Next scenario
```

I usually use the Import feature in XLsior, which records the source and time of the imports, to copy the results. The comparison sheet can be used to check that the absolute values of the results are the same, or that they are within a specified tolerance.
5 PROBLEMS

Although systematic, automated testing is a useful tool in dealing with spreadsheets, the ride is not always smooth. I use unit testing techniques a lot, and regression testing when appropriate, but rarely do full independent system tests. Even with unit tests, there are a number of problems that are encountered regularly.

- It is often difficult to devise effective tests if there is no specification other than the spreadsheet itself. This is not a problem when developing a spreadsheet from scratch, but can be a serious obstacle if you are using tests to help review someone else’s spreadsheet. The tests soon become tautologous if the formulae are being used to derive the specification that is used to drive the tests for the formulae.

- Dealing with floating point arithmetic is sometimes tricky. It is often necessary to use a pair of inequalities instead of a single equality.

- If a test fails, it may be a problem with the test rather than with that which is being tested. This can be very frustrating.

- It can be very time consuming to set up a full range of tests. I usually address this by not attempting to do so at the outset. I start with a few tests of significant calculations, and gradually add more tests. Typically I add new tests for any changes that I make. Whenever an error is found I add a test that would have detected it, and other tests that would detect similar errors. And sometimes I just add more tests when I have time, or as an idea occurs to me for a good one.

- It is often difficult to devise tests if there are large, complex formulae. The use of simpler formulae with explicit intermediate results makes testing much easier, as you can test the derivation of the intermediate results independently of each other and of the way in which they are combined.

None of these problems is specific to spreadsheets, and I encounter them regularly when testing systems written in more conventional programming languages.

CONCLUSIONS

Systematic testing of spreadsheets is both possible and desirable. Of the problems that are encountered, some (such as the difficulty of regression testing when the layout has changed) are peculiar to spreadsheets while others (the care needed when dealing with floating point arithmetic, lack of specification) occur whatever the programming language used. There are some simple techniques that can be used to set up automated tests, not all of which require macros.

REFERENCES


XLSior is available from http://www.xlsior.com
Computational Models of Spreadsheet-Development
Basis for Educational Approaches

Karin Hodnigg ¹), Markus Clermont ²), Roland T. Mittermeir ¹)

¹) Institut für Informatik-Systeme
Universität Klagenfurt
AUSTRIA
Roland.Mittermeir@uni-klu.ac.at, Karin.Hodnigg@uni-klu.ac.at

²) Software Quality Research Laboratory,
Department of Computer Science and Information Systems
University of Limerick
IRELAND
Markus.Clermont@ul.ie

ABSTRACT

Among the multiple causes of high error rates in spreadsheets, lack of proper training and of deep understanding of the computational model upon which spreadsheet computations rest might not be the least issue. The paper addresses this problem by presenting a didactical model focussing on cell interaction, thus exceeding the atomicity of cell computations.

1 INTRODUCTION

This paper departs from the perspective that spreadsheets are end-user programs. The main objective of spreadsheet development is “manipulation and presentation of data found in tabular form” [Filby, 1998]. The intuitiveness of spreadsheet development hides to a large degree that it is actually a programming activity. Typing constant values into some cells and a formula into another cell is not seen as programming. It is rather comparable to using a pocket calculator. The immediate presentation of the result even supports this notion. This allows introducing novices without much ado. One learns to use an environment instead of learning a model. While this can be seen as base for the high popularity of spreadsheet systems, it hides the reality that spreadsheet developers are expressing themselves in an inherently functional formula language.

[Nardi, Miller, 1990] identified immediate feedback through formula evaluation, tabular grid and related layout definition, the possibility to shift complexity by splitting formulas over different cells, and the rather declarative nature of most aspects of spreadsheet languages as sources of success. But while these features are certainly helpful for those writing small spreadsheets, they easily become obstacles when complexity increases. Certainly with large and evolving sheets, lack of higher level abstractions becomes a burden.

Only values are represented (formulas are eagerly evaluated), and formulas, but for a selected single cell, are hidden. This leads to a specific characteristic of spreadsheet pro-
grams: hiding control and data flow information behind “static” values. During maintenance, this complicates comprehension of existing spreadsheets. Intertwining the layout of results and dependencies of computations is another source of conceptual complexity. Cells can reference each other over large geometrical distance. Hence, comprehension of a spreadsheet is a non-trivial task and consequently many errors are introduced or remain unnoticed. [Sajaniemi, 1998].

High error rates found in business spreadsheets ([Panko, 1998], [Mittermeir et al, 2002], [Brown Gould, 87]) indicate that the spreadsheet quality issue cannot only be resolved by powerful tools. One has rather to agree with Hoare’s statement that “a significant challenge for programming theory is to […] develop an understanding to assist in the selection of an appropriate tool for each purpose.” [Hoare, 1999].

Based on these reflections, this paper first identifies some crucial aspects of the spreadsheet paradigm. Then, it shows how differently basic issues are solved by different implementations of spreadsheet. This calls for a common conceptual background, which will be developed in section 5.

2 THE SPREADSHEET PARADIGM

To identify the target of this research, one is tempted to ask, whether there is such a thing as a “spreadsheet language” and if so, what this language might be. For a given product, it makes no difference, whether one types =IF (A1 = B1; …) or =WENN( A1 = B1; …). These commands have the same effect on the data. Likewise, it makes no difference, whether this command has been typed in, selected by mouse click from some panel, or copied from a cell holding a similar formula which was edited afterwards. The clue is that the system provides the concept of an alternative and this concept is presented in different linguistic forms to the user. But the differences in linguistic form are rather shallow and users have to develop a conceptual model resting on the concepts behind the functions implemented in various spreadsheet products. Since all of these functions rest on common mathematical concepts, users must not be blamed if they assume that the sheet behaves in exactly the way they expect these mathematical functions to behave.

As the mathematics of the functions used in spreadsheets are well known to domain experts, the functional nature of cell-computations makes spreadsheet programming impressively simple [Moström, 1998]. Moström and Carr subsume the basic knowledge needed to implement a spreadsheet as follows:
- There is a tabular grid consisting of (addressable) cells.
- A cell can hold either a formula or a static value.
- The formula language is declarative, having the form
  
  =<cell_addr> [ <operator> <cell_addr> ] or
  =<function>(<cell_addr1>,..,<cell_addrn> [;<system parameter>])

  with system parameter being an element of the spreadsheet system rather than the mathematical base of the language.
- Cells can be referenced as solitaires (A1) or by range reference (A1:A12). A reference can be either absolute or relative.

This resembles functional programming and clearly contrasts with “conventional” programming. The spreadsheet language is mathematically traceable and highly declarative as it “emphasises on the evaluation of expressions” [Montigel, 2002]. Normally, spreadsheet languages focus on spatial relations of data, not on the temporal sequence.
This applies at least to very basic concepts for implementing spreadsheet programs and lead to the statement that the spreadsheet language is a “programming language for the masses” [Moström, 1998]. Even without specific training, everybody can write models based on the writers domain expertise. But there is no warning when those limits are left because extensions of the spreadsheet paradigm like those discussed in section 4 violate basic assumptions of the model.

The Conceptual Mismatch

For a novice, developing a spreadsheet is like a child’s building a play-house. Formulas are written into cells like placing bricks on bricks. The intermediate results can be admired after each step. With sufficiently small problems this cell-by-cell approach can follow an implicit DDG almost in breath-first manner. The geometrical placement of cells is almost irrelevant.

![Figure 2: A user’s view based upon [Igarashi, 1998]](image)

But what is easy during construction might prove to be difficult for later comprehension. When perusing cell B2 in Fig. 2, the cell’s value is shown in the sheet and the underlying formula is displayed in the formula bar. The constant values in A1 and A3 and the formulas in A2 and B2 are related by invisible dependencies. Clicking into one of these cells, the spreadsheet GUI shows the “first level”-dependencies, i.e., the addresses of source cells. A spreadsheet programmer may notice that A2 depends on A1, but which cells depend on B2 remains totally hidden. Thus, changes to B2 may result in changes “anywhere”. Theoretically, any computation might be affected by changes in a given cell.

Navarro-Prieto and Canas’ results indicate that spreadsheet writers have developed good mental structures for data flow information [Navarro-Prieto, Canas, 1999]. [Tukiainen, 2001], in contrast, points out the need to memorize (invisible) coherence in a spreadsheet without explicit representation. We conjecture that comprehension becomes increasingly difficult with growing size of the sheet. It will matter specifically, when references exceed the window visible on the screen. Assuming maintenance of the sheet, the situation is aggravated. With fading memory, the basis for Navarro-Prieto’s hypothesis is lost.

Another issue exhibited by Fig. 2 is a likely conceptual mismatch between the displayed values and the underlying calculation model. The user does not “see” the data dependencies. They are hidden behind (possibly misconceived static) values. For a spreadsheet maintainer a visit of every content-bearing cell is necessary to build a mental model of the
spreadsheet program. Every visit extends the user’s conceptual model, thus only the “final” (i.e. the most recent) model corresponds to the real data flow of the program. The fact that users see data dependence at best incrementally is particularly detrimental when those are not in line with the generally assumed left-to-right, top-to-bottom assumption and when the value-perspective allows several data-flow interpretations.

3 SPREADSHEET PROGRAMMING CHARACTERISTICS

Though intuitive, assumed data flow semantics reach their limits in critical situations. As stated in [Clermont, 2003], spreadsheet programs “share many features with data flow concepts”, but “some of the key concepts, such as the consumption of tokens” are not part of the spreadsheet paradigm. Questioning this statement, one realizes that spreadsheet semantics are tool-dependent. Established models to describe programming language semantics are not fully applicable and hence the broadly shared interpretation of spreadsheets as dataflow programs breaks down.

3.1 Evaluation Strategies

A decisive difference between conventional (functional) programs and spreadsheet programs is evaluation time and process. Conventional programs are fully specified before they are evaluated whereas spreadsheet evaluation takes place after each incremental development step. The end of the development process is never made explicit to the system, though. It stops, when the developer is satisfied with the results of the computation.

So far, we have been unspecific when referring indiscriminately to the functional paradigm and to the data flow paradigm. Addressing evaluation, though, one has to recognize that these concepts differ in evaluation order and concept.

Data Flow Semantics

Data flow programs (DFP) like spreadsheet programs do not need an explicitly defined control flow. Order of execution is implicitly defined. In DFP, data dependencies control the sequence of function evaluation. As a data flow program is usually visualized by a data flow graph, one may conceptualize evaluation of a given node as soon as all its edges bear data, i.e., all required information is available. The data propagated is a token that holds the result yielded by the computation of the node placed immediately downstream. This evaluation concept is based upon the consumption of tokens. Re-evaluation of the DFP implies re-computation of all tokens.

With spreadsheet semantics, however, there is no “value” marking of these edges of the DDG but rather a marking of “change”. [Yoder Cohn, 2002] point to this crucial difference: In a data flow program, a cell is re-evaluated only, if all of its sources have new values for processing. In spreadsheets though, a single re-evaluation marker suffices to trigger re-evaluation. This marker is an explicit element of control: Moreover, for treating loops, DFPs include loop nodes as special concept. But loops are not part of the standard spreadsheet paradigm.1

1 ) Excel allows recursion with limited iterations (see 4.2). But this is rather not part of the standard repertory of spreadsheet writers and has conceptual limits which may result in unwanted side-effects.
Graph Reduction Semantic

Passing control seems akin to functional programming with its graph reduction semantic [Sestoft, 2001], [Dermoudy, 2003]. Here, each formula is interpreted as functional statement. Graph reduction semantics imply that a function call and its arguments are replaced with the result of function application. Since the result of a function can be used more than once in a program, reduction has to be repeated for each occurrence.

Consequently, [Clermont 2003] postulates that spreadsheet program evaluation seemingly follows graph reduction principles. There are two main arguments that show that spreadsheet programs are no pure graph reduction programs though: loops and change propagation.

- **Recursion and Loops:** The functional programming paradigm does not include loops. Recursion is the concept to express repetition. Recursion, however, is not part of the spreadsheet paradigm since it inhibits the visibility of intermediate results and postulates inherently the provision of a global control flow.

- **Change propagation:** The interactivity of spreadsheet programs leads to a sophisticated change propagation technique [Clermont, 2003], [Yoder Cohn, 2002]. If a cell’s content changes three steps happen to maintain consistency:
  - the formula’s value has to be re-evaluated,
  - depending formulas have to be re-evaluated, and
  - formulas within the transitive closure have to be re-evaluated.

Thus, re-evaluation is mainly token-driven. [Burnett et. al., 2001] coined the term “continuous evaluation” to highlight immediate currency of results.

3.2 Inconsistent Evaluation Strategies

To resolve these contradictions, [Clermont, 2003] suggests spreadsheet programs to be considered partly as graph reduction program and partly as data flow program. Which of the two applies in a given situation becomes relevant during spreadsheet maintenance. A distinction has to be made between local and global evaluation:

- **Local Evaluation:** Evaluation of a cell’s initial value (starting from its formula) is based upon graph reduction according to the spreadsheet’s DDG.

- **Global Evaluation:** If changes in a cell occur, they are propagated via the data flow graph to all dependent cells through change tokens. This data flow graph corresponds to the reverse DDG of the spreadsheet. Thus, cells are only re-evaluated when needed.

Hence, spreadsheet programs incorporate both concepts, depending on viewpoint. Due to interactivity and visibility of all cells, the global evaluation strategy implemented has to be eager. However, this does not prevent lazy evaluation to be locally applied, e.g. Excel evaluates IF-clauses lazily.

There is another distinction between functional and spreadsheet programs. By the nature of the spreadsheet paradigm, every cell on the spreadsheet has to be considered as output whereas a functional program has a set of selected outputs. In [Yoder Cohn, 1994] an approach is presented that is based upon demand-driven evaluation of the cells of interest only. But the authors rely on “keeping all cell values up to date”.

Looking behind these differences, a spreadsheet program unifies functional and data flow concepts. As stated in the references, the very (natural) base of functional languages is data flow graphs where functions are nodes and edges represent the data dependencies.

2) For exceptions and approximations see section 4.2.
4 IMPLEMENTATION DIFFERENCES

While section 3 discussed principles of spreadsheet evaluation this one concentrates on actual implementations of spreadsheet systems. How do they treat evaluation and where are differences between systems or between concept and implementation? Apparently, such differences will constitute risks for development and pitfalls for education.

4.1 System Specific Evaluation Strategies

To analyze how system builders resolve the crucial design issues for an evaluation strategy, reducing unnecessary re-computation and maximizing use of available computational resources [Yoder Cohn, 1994], the implementation of Microsoft’s Excel® and Gnumeric are used. For the open source product Gnumeric, the sources used are quoted in the discussion. For Excel (2000) we had to rely mainly on the Online-Help.

Excel Value Recalculation and Change Propagation

To be efficient, Excel performs so-called “minimum recalculation” using the following strategy [La Penna, 2001]. It keeps an internal list of all linked (interdependent), cells bearing formulas in a workbook (like a2>b2>b4). Cells with constant values are not part of the list, as they cannot be affected by change propagation. If a change occurs, all cells (transitively) dependent on the cell that changes are marked with a recalculation flag. Recalculation starts according an internal list of dependent cells.

![Excel Value Recalculation and Change Propagation](image)

Figure 3 Formula Evaluation in Microsoft Excel 2003

Fig. 3 shows Excel 2003’s Formula Evaluator applied to the sheet of Fig. 2. The evaluation path starts at the formula in B4, a data flow sink in this sheet. It continues in a step-wise manner against the direction of data flow to cells B2, A2, till A1, a constant. Thus, graph reduction is performed and every single step can be seen by the user.

Gnumeric Value Recalculation and Change Propagation

Gnumeric distinguishes between two types of dependencies: single dependency, which is a reference to an individual cell (=A1), and range dependency, which encompasses a set
of cells \((=\text{SUM}(A1:A5))\). Since it is necessary to determine dependencies for every (transitively) dependent cell, both dependency types are mapped to distinctive data structures to ease the lookup of a cell’s dependents.

To recalculate values, a recursive approach traverses the given expression tree, re-calculating each source cell (recursively, if the source cell contains a formula). This string reduction approach is rather inefficient if the same cells are to be re-evaluated more than once. Hence, Gnumeric buffers the cells in an evaluation queue, traversing the dependency data structure. Value recalculation then corresponds to graph reduction manner.

### 4.2 Circular references and Iteration

The spreadsheet paradigm does not provide any kind of global control flow. Thus, loops (iterations) and recursion are not defined. There are spreadsheet systems that simply prohibit (even the accidental use) of circular references. Francoeur states for an ExcelComp-Tool that "an admissible spreadsheet contains no directed cycles“, i.e. no recursion [Francoeur, 2002]. Accidental use happens though by incorporating the cell holding an aggregation function itself into the scope of aggregation e.g. writing \(=\text{SUM}(A1:A3)\) into \(A3\). Surprisingly enough, there is no common approach to handle these circular references.

#### Microsoft Excel Circular References

Excel provides an ignorable warning. If a circular reference is accepted “as is” then every cell containing a function leading to the circle is considered to be a terminal node (constant). Neither this cell nor its dependents will ever be re-evaluated after the warning has been ignored. Without any special marking, the cell provides the value zero. This is quite problematic since “0” could be a legitimate value expected by the user and the rupture in the evaluation path will lead to wrong results anywhere. Dependent values will remain on whatever value they had before the recursive case occurred and remain so, even if any of their other sources is changed.

By a special command-panel Excel provides an iteration scheme exceeding the spreadsheet paradigm. It allows to evaluate a cell containing a circular reference over a user defined constant number of iterations. If a change anywhere in the sheet, even outside the transitive closure of the cell’s sources occurs, the cell is re-evaluated. Excel offers this to accommodate requirements of some scientific computations. Nevertheless, the introduction of an even reduced iteration model is a substantial intervention into the “traditional” spreadsheet paradigm.

#### Gnumeric Circular References

Gnumeric includes circular references into its concept. No warning or indication is given to the user. Gnumeric does not consider circular referencing cells being terminal nodes. In some (!) cases it treats circular references as a kind of two-staged loop. If the value of a cell with a circular reference has to be evaluated, Gnumeric supposes it to start with a given value zero (0), computes the function over the non-recursive part and takes this value to recompute over the full extent. If recalculation of that cell is necessary, i.e. if changes in one of its source cells occur, Gnumeric consults the given (old) value and uses it for re-computing the new result.
A small experiment shows this behaviour: $A1$ holds a constant value ($1$); $A2$ builds the sum of $A1$ and itself ($=A1+A2$). After the input of this formula, Gnumeric yields $2$ in $A2$. If the value in $A1$ is changed to $4$, $A2$ becomes $10$. If subsequently $A1$ is changed back to $1$, $A2$ becomes $12$. Thus, the “previous” value of both cells and the “new” value of the changed cell are used to build the sum: $(2 = 1+1+0, 10 = 4+4+2, 12 = 1+1+10)$. I.e., a non-recursive computation is performed and its result is added to the previous value contained in the recursive cell. Aggregation functions such as $AVG()$ and $SUM()$ provide results in a similar way. Trying to incorporate subtraction in $A2$ ($=A1-A2$) leads, unexpectedly, to $0$ with no re-evaluation taking place though. With this feature, the treatment of circular references seems even more dangerous, since probably incorrect values are computed without any warning.

4.3 Copy/Paste and movement heuristics

A discussion of spreadsheets will be incomplete if specialties of the development process such as “drag-and-drop” or “copy/paste” are not considered. As formulas are parameterised by (either constants and/or) relative cell references, the way these references are adjusted is crucial for spreadsheet correctness.

Moving Cells

Moving cells from by a drag-and-drop operation is a common operation in spreadsheet development. It is distinct from cut and re-paste at a different location, since by drag-and-drop the link to the referenced cells persist while cut-and-paste preserves the geometrical pattern of the relative addresses. To keep the spreadsheet consistent can be resolved in two ways. References pointing to the “moving” cells could move with the cells (according to a pointer idea) or references starting from the cells could rather keep the reference treating them as a “geometrical” pattern. In the latter case, the DDG will remain unaffected except that the moving node will get a different address-label.

Both systems, Gnumeric and Excel provide this feature with computational reference keeping dominance over geometrical patterns. This principle is implemented for both, moving cells, and inserting columns or rows. The deletion of cell block contents leads to zeros in cells that refer to the removed block, since the cells exist but do no longer hold any value and zero is considered as default value for empty cells. If whole columns or rows are totally deleted, though, a reference problem pops up as the referenced cell does not exist any more. In this case Excel displays the error value #REF, Gnumeric does not display a value, although the cell carries the #REF!-value. Aggregation functions such as $SUM$ or $AVG$ play a special role in this context though. The range covered by these formulas dominates over reference or geometrical patterns.

Figure 4 highlights the interaction between development steps and aggregation functions. Cells $A1$ and $A2$ contain constants. $A3$ and $A4$ contain the formulas $=A1*10$ and $=A2*10$ respectively. $A5$ contains an aggregation function with the range $A1:A2$. If the formula-block $A3:A5$ is copied to column $C$, the references are adapted according to the geometrical pattern of their copy source. If the same block is moved from $A3:A5$ to column $B$ though, the references to the source cells persist. If a referenced cell, say $A2$, is moved to $D2$, then the referencing cell (still $A4$) adapts and in a notion of pointer semantics points to the new cell containing the referenced value. Thus $A4$ contains the formula $=D2*10$. Interestingly enough, the aggregation formula does not adapt as well. $A5$ still contains the formula $=SUM(A1:A2)$ and yields the value $1$. The content of cell $A6$ will in this case yield $31$. Apparently, history is only partly preserved and the notion of physical areas dominates over development history.
4.4 Filling Cells

All spreadsheet systems provide “filling” operations. Starting from a given cell users can automatically “fill” geometrically neighbouring cells either with values or formulas. In all cases, default adjustments are made. Whether these defaults are intuitive and meet the developer’s expectation depends on the situation and on the developer’s conceptual model.

**Filling Cells with Values**

Automated filling with values seems straightforward. Nevertheless, there are some differences between the systems considered. Both Excel and Gnumeric provide a “copy of constants” operation to duplicate the value of a single starting cell and a “series copy” operation that successively increments values (e.g. \(a_2 = a_1 + 1\)). The distinction (common to every spreadsheet program) is based upon the “Control”-key.

If more than one value is selected as starting point to the value series, Excel tries to figure out the subsequent values by building a geometric series. So, if a user wants to fill a block with four values down a column, let them be \((3, 7, 2, 5)\), a geometric series is built up \((3, 7, 2, 5, 4.5, 4.6, 4.7, 4.8, 4.9, 5, 5.1, 5.2, 5.3, 5.4, 5.5)\). This may not be exactly what the user expects. A copy operation with the constant values \((3, 7, 2, 5, 3, 7, 2, 5, \ldots)\) is triggered via the control-key, which is in this case non-intuitive.

Gnumeric on the other hand takes the differences of the last two values of a block to compute sequent values. The filling of a column with the values \((3, 7, 2, 5)\) leads to a sequence of following values \((3, 7, 2, 5, 8, 11, 14, 17, 20, \ldots)\). In Gnumeric, the control-key does not provide another function.

**Filling Cells with Formulas**

When clicking on cells filled with formulas and dragging them over an area, references are treated as geometrical pattern and constants maintain their value. Thus, there is no adaptation as discussed above. If more than one formula is selected to be copied, the block of formulas is taken and cloned, if the selection window is dragged over either rows or columns. Hence, different constants in two successive formulas remain different in the newly filled formula block. In [Igarashi et al., 1998] an interactive graphical induction approach is presented. There, the structure of a spreadsheet program and regular patterns are used to induce a continuative formula pattern.

4.5 Conclusion on differences

The comparison showed that even such typical spreadsheet operations like movement and filling of cells by “drag-and-drop” are implemented differently in frequently used sys-
tems. This can cause unexpected results. Even if differences seem to be marginal, they indicate that common spreadsheet didactics is hard to achieve and system specifics have to creep in. Worst about these differences is the marginality of distinction. Different spreadsheet systems behave identical in most situations, but not always! One has to conjecture that there is no common spreadsheet language which can be defined as the union of a distinct formula language and tabular layout issues. To fully understand spreadsheet development, one has to learn details of the system too. Moreover, adaptation heuristics, defined to help spreadsheet developers, cause effects of non-associativity in the sequence of certain development steps.

5. SPREADSHEET SEMANTICS

The previous section has shown limits of comprehending spreadsheets and hence of teaching spreadsheet development on a cell-level basis. The naïve perception of spreadsheets as an arrangement of cells (c.f. the bricklayer’s approach described in sect. 2) reaches its limits either when computations become too involved or when due to maintenance operations incorporating new requirements the sheet evolves over time. Here, a model is introduced that should help to comprehend the interdependencies between cells without falling into the problems mentioned for data flow semantics or reduction semantics discussed in section 3.

As any model describing the semantics of a language, a model for spreadsheet semantics has to be expressive and faithful with respect to the intended semantics spreadsheet users and developers of spreadsheet systems have in mind. Further, such a model has to be simple in so far as it requires only a minimal number of primitives. Finally, considering the spreadsheet user-community, it has to be highly intuitive. The latter argument is to be seen as a distinction between programming language semantics and spreadsheet language semantics, since the former are to be understood by programming professionals whereas the latter are to be understood by application experts who are rather programming laypersons.

As shown in section 4, the divergent semantics of spreadsheet systems conflict with a notion of common spreadsheet semantics. Hence, one must not expect that such a model covers all detailed variations implemented. However, even if it does not cover them, it should at least not be in conflict with them.

5.1 Relationships and Visibility

Another peculiarity of spreadsheets is to be considered. According to the bricklayer-semantics, each brick (cell) can be placed anywhere on the sheet. (Transitive) dependencies are established due to the relationship(s) a formula establishes with the cells it references. These references are normally represented as relative positional distances from the target cell to the source cells. Hence, whenever the target cell moves, the references to the source cells experience an identical movement irrespective of whether the respective positions in the sheet contain appropriate values or not. Absolute references, i.e., references to fixed positions are also possible but not the norm. However, even in these cases the absolutely referenced cell can be anywhere on the spreadsheet. Cells serving as parameters (or data sources) for a particular formula can be arbitrarily spread over the sheet. Modifying the spreadsheet program by inserting or deleting rows or columns affects neither relative nor (interestingly!) absolute references. The relationship once established to a particular cell (with its content, either constant value or computed value) remains. Thus distance (relative) or address (absolute) is adapted as shown in section 4.3.
The relationships holding in the computational perspective of the sheet are stronger than the positional aspects.

This freedom is lost with aggregation functions. For them, the concept of a (physical) area has been defined. This is a rectangular block of horizontally and vertically consecutive cells. In this case, however, the target cell receives its value from cells placed within the geometrical confinement of this rectangle. Deletion and/or insertion of rows or columns may affect this area, if they take place within the borders of the area (not, if they take place at the border). Thus the area has a certain degree of flexibility. Further, areas yielding results into aggregation functions placed in different cells might overlap (which would be a contradiction with pure data-flow semantics). Thus, there is, like with individual source cells, no unique ownership of sources.

While cells on a yet empty sheet and cells containing only constant literals are globally visible indeed, this does not hold for cells containing formulas. A cell holding a formula that references only constants might still be conceived as globally visible. However, this cell cannot “see” any cell that directly or indirectly serves as target for the value of its own computation. Otherwise, the computation would contain circular references. Therefore, there is an implicit visibility arrangement between cells. This arrangement depends on the (transitive) target-source relationship between cells.

5.2 The Projection-Screen Model

Computations confined to individual cells are not a problem in spreadsheet education, since they are conceived as functions well understood in the application domain. The functional nature of cell-based computation provides clear scoping. The global model remains conceptually unsupported though. It is not adequately addressed in introductory teachings, conventional models cannot fully account for all effects, and typical spreadsheet operations might conflict with them.

Therefore, we present a model based on the interrelationship of cells. Drawing upon the instant visibility of results of any computation and the implicit relationship between cells due to data dependency, we interpret cells as optical devices, reading results from screens (cells) placed in front of them and projecting the result of their computation on their own screen which supposedly is placed on the back of the viewing mechanism.

Projection-Screens without aggregation

Spreadsheets containing just empty cells, cells with constants and cells with non-aggregative formulas serve as point of departure. For all cells hold:

- **Empty cells** can be ignored, since they do not partake in any computation.
- Cells containing constant literals might contain labels or constants to be used in computations, i.e. by other formula cells.
  - **Labels** do not partake in any computation. Hence, one might be tempted to treat them like empty cells. However, not the cell holding a literal decides on its usage. A literal cell is globally visible. Hence, any other cell in the sheet can at any time in the development process reference this cell. Hence, labels are treated like computational literals.
  - **Computational literals** (usually numeric values) are treated as primitive 0-argument formulas. Their result is the value denoted by the very literal. 0-

---

3) For exceptions see section 4.2, treating circular references in different implementations.
argument formulas (e.g. NOW()) look at no other screen. They only present their own value on their own screen, thus making it globally visible.

- Non-aggregative formula cells have as many arguments as they are (relatively or absolutely) referencing cells in their formula. They read in a non-destructive manner the values from screens mentioned in the formula and project the result of their computation on their own screen. The screens they read from have to be conceptually positioned “in front of them”.

This model requires a unique viewing direction among screens. But it is completely independent of geometrical placement of source or target cells and neutral with respect to computations referencing individual cells. It shares directionality with the data-flow model, but in contrast, nothing flows. Formula cells just read from the projection-screens “in front” of them and hide their own results from screens “in front.” As the viewing devices are constantly attentive, they realize when a value of one of the screens in front of them changes. This leads to re-evaluation of the own formula. Thus, intermediate results, whether they stem from literals (0-argument formulas) or regular formulas, can be shared by as many target formulas as needed as long as the direction of visibility ”look in front of you, write the results on your back” is upheld.

However, from a risk assessment perspective, one might check, whether the direction of visibility can be linearly mapped to a partial order in the geometric placement of cells. Deviations might serve as complexity measure. Computation of related risk indices would go beyond the scope of this paper though.

![Figure 4: Projection Screens a) without und b) with aggregation functions](image)

**Projection-Screens with aggregation**

One might be tempted to treat aggregation functions simply as shorthand for explicitly mentioning a (huge) set of arguments. But section 4.3 shows that aggregative and non-aggregative functions differ from an evolutionary perspective. Thus, the semantics of aggregation needs special treatment.

Still resting on visibility, one might consider the area under aggregation as the set of screens illuminated by a common spotlight. This concept withstands evolution, since deleting a portion will reduce the area of visibility, while inserting empty space will enlarge it. If, later on, this empty space is filled, it is seen by the aggregation-function’s viewing mechanism as if it had ever been there. Like on stage, the aggregation function
sets a spotlight on that ensemble of actors (still all looking up-front, carrying their result clearly displayed on the screen carried on their back) that partake in this particular aggregation. Like on stage, different spotlights (on the stage they have possibly different colour) might illuminate different actors (cells) and some of them might be in the focus of different spotlights. Hence, they yield their values for different aggregation functions. Actors leaving the illuminated areas are no longer seen by spectators.

5.3 Discussion of selected evolutionary steps

For the sake of demonstration, some prototypical patterns frequently recurring on spreadsheets are analyzed with respect to the projector-screen model.

Many-handed figures

Certain goddesses such as Bodhisattvas are depicted with many hands reaching out in different directions. Reducing this to spreadsheets amounts to a cell (or block of cells) affected by evolution that has several dependencies outside of the block manipulated. In its most concentrated form, one might conceive of an *IF*-statement, consisting of 

\[ \text{<condition>, <argumentT>, <argumentF>} \]

where each of the three formulas might have references to other cells that extend over the geometrical area affected by movements, insertions, or deletions. As long as these manipulations do not affect a cell directly addressed in this “many-handed-statement” (which could happen with deletions or with movements over an area where such a source-cell is located), the computations in the statement are not affected. The principle that computational references dominate over location-based references applies. Thus, not only relative addresses are adjusted properly. Even absolute references are adjusted to maintain established data connections.

This is consistent with the projection-screen model. Visibility is strictly based on position of the screen relative to the screen it reads from. On the first level, those exhibit only constant values (either input cells or cells used as constants of the spreadsheet program). Every formula computing intermediate results used by a cell on the path between the panel of constants and the cell considered puts this cell one level towards the rear of this screen/projector scene. But as insertions of rows or columns have no effect on this distance nor do deletions of rows or columns (that do not directly affect a node on this chain) have any effect on this arrangement, the projection-screen model is consistent with such operations. The same applies if not a single cell but a block of cells is considered. Even if insertions (deletions) modify the size of this block, no changes in screen positioning are needed and, therefore, no visibility changes are induced.

Queue on a staircase

The previous case considered independent cells or independent areas. There are cases with connections within a block of cells that is treated as conceptual entity. Running numbers are the simplest example. One starts with some constant, say \( i \) in e.g. cell \( C3 \), writes \( =C3+i \) into cell \( C4 \), and fills cells \( C5 \) till \( Cnn \) by dragging down \( C4 \). Right to these running numbers, usually information with application semantics is given. It might be necessary to insert or delete a row or to move part of this construct to some other place. Although the block with running numbers is a conceptual entity, the spreadsheet system deals with it as set of neighbouring cells with each one having just one external reference. Hence, the rules for the many-handed figures (here: single-handed) apply.

This also holds for the projection-screen model. One might envision a staircase where the front element holds the constant (here \( i \)) and displays it on the screen on its back. All
other elements are looking at the (single) screen immediately in front of them and display whatever they read incremented by 1. Adding a step to this staircase, or moving the tail of the queue some steps back (or the front of the queue some steps forward) does not change this visibility. Geometrical reference is adjusted to maintain the relationship to the visible screen in front of the viewer directly affected by the geometric rupture. Deletion of individual cells, however, does have an effect in this case, since not only the step of the deleted is removed; its screen is also blinded. Hence, reference is lost and #REF is displayed as error message for the cell that lost its ancestor, but also for the cells “in the back” of the respective cell. However, when the head of this affected sub-queue gets fixed, its tail and thus the complete queue is fixed automatically. The general visibility system and the computation mechanism of the dependent cells is not affected in this case. Those cells just cannot produce interpretable results because (one of) their ancestor(s) shows no result on its otherwise perfect screen.

But what, if not the cell but just the formula is deleted? In this case, the queue starts with 1 again without reporting any problem. Is this consistent? It is! Due to the default value 0 for empty cells, the cell behind the empty one notices 0 in the predecessor, increments it and thus displays 1 and all cells behind it act accordingly. Thus, a new queue is defined.

Flying carpets

Finally, one should look at aggregation functions. Here, the cells to be aggregated over are affected by operations relating to the geometrical arrangement of the sheet. But the cell containing the aggregation function can, like a flying carpet, be freely moved to a new geometric position without losing sight. One might assume that this causes problems in a model relying strictly on data dependencies. The spotlight-interpretation of visibility helps though.

The spotlight covers an area (on stage as well as on the sheet). This illuminated area is independent of whether the area is populated or not. The viewing mechanism has always to be in the back such that all items in the illuminated part can be seen. Thus, conceptually, it might be necessary to step back, if something is inserted that is already at a level far away from the front panels showing constants. If maintenance operations change the size of the illuminated area, it is important to note that the scope of illumination is always defined by the fringe positions. This border does not change. Hence, deletions shrink and insertions widen the focus. Other than that, the basic mechanism remains and thus the analogy holds. Since aggregation requires only visibility, the analogical model creates no contradictions, if parts of the illuminated area are illuminated by different spots (say, different wave-length) such that each spot serves to identify the input to its particular cell holding some distinct aggregation function.

Recursive images

As there are different implementations, we cannot give a single consistent answer for recursion. However, the projection-screen model can cope with both situations mentioned. The single evaluation step identifies a problem, shuts off projection and replaces it with a still-picture. The pseudo-recursion implemented as limited number of iterations places an additional mirror in a slightly angled position such that each iteration can see the non-recursive portion as well as the result of the last iteration.

4) In a variation of the queue in a staircase, one might think of situations where only every n-th step holds an incrementing formula. In this case, the argument raised for deletions obviously applies only if a formula bearing cell is deleted. Otherwise, the many-handed figure case applies.
6. Summary

Cell based specification and immediate feedback made spreadsheets a programming device for non-programmers. Spreadsheets provide abstraction through information hiding and modularity. Operations such as copy/paste, drag, and fill support a “next cell”-development approach. While this is convenient when developing a spreadsheet, it is harmful if changes and modifications have to be made. Here, a solid conceptual model is needed.

Current spreadsheet implementations do not strictly follow any of the established conceptual models. They rather follow a teleological approach of “what the user probably intends to do”. But phrases containing the word “probably” are problematic as they do not hold for all situations. This poses a challenge for education. If limits and “critical factors” remain unnoticed or misconceived, spreadsheet quality is seriously impacted.

This paper presented a model to explain spreadsheet mechanics to beginners that extends to expert level concepts. It should not be one of the dangerous crutches that break when users try to leave their cradle. Whether this is true has yet to be tested in formal experiments and by exposing the model to the community. After all, with spreadsheet education one has to consider that spreadsheet programmers are not interested in programming per se [Peyton-Jones, Blackwell, Burnett, 2003]. But nevertheless, progress needs education. This holds for spreadsheets as for any other intellectual activity.

7. REFERENCES

[Brown Gould, 1987]  

[Burnett, 2001]  

[Burnett et. al., 2001]  

[Clermont, 2003]  

[Filby, 1998]  

[Francoeur 2002]  

[Hoare, 1999]  

[Igarashi et al., 1998]  
[LaPenna, 2002]
Loreen La Penna (2002), “Recalculation in Microsoft Excel 2002”, Microsoft Corporation, 

[Meeks, 2001]
Michael Meeks (2001) “A discussion of the new dependency code, version 0.2”, Gnumeric (1.1.16) Docs

[Mittermeir et. al, 2002]
Industrial Context", Proceedings of the 3rd Annual Symposium of the EuSpRIG, pages 35-46

[Montigel, 2002]
Markus Montigel, "Visual Programming Languages (Spreadsheet languages)", 14.30 p.m. 2004/5/8
http://www.cs.uno.edu/~markus/02_Courses/past/csci6990/6990.03_Spreadsheet.ppt.pdf

[Moström Carr, 1998]
Proceedings of the 10th Annual Workshop of the PPIG’98, pages 117-127

[Nardi, Miller, 1990]
Bonnie Nardi and James Miller, “An Ethnographic Study of Distributed Problem Sov9ing in Spreeadsheet 

[Navarro-Prieto Canas, 1999]
Raquel Navarro-Prieto and Jose J. Canas (1999), “Mental Representation  and Imagery in Program 
Comprehension”, Proceedings of PPIG99 (Psychology of Programming Interest Group)

[Panko, 1998 ]
Volume 10 (2), pages 15-21

[Pennington, 1987]
Second Workshop, pages 100-113

[Peyton-Jones et al., 2003]
Simon Peyton-Jones, Alan Blackwell and Margaret Burnett, “A User-Centred Approach to Functions in 
Excel”, http://research.microsoft.com/~simonj/Papers/excel, 14.30 p.m. 2004/4/6

[Sajaniemi, 1998]

[Sestoft, 2001]
Peter Sestoft (2002) “Demonstrating Lambda Calculus Reduction”, in “The essence of computation: 
complexity, analysis, transformation”, pages 420-435, Springer Verlag

[Tukiainen, 2001]
Novice Users”, Interacting with Computers (IWC) , 13 (4), p. 427-446

[Yoder Cohn, 1994]

[Yoder Cohn, 2002]
PREMIUM SOLVER PLATFORM

FOUR BUILT-IN SOLVERS, SIX MORE AT YOUR COMMAND

Power to Solve Your Problem.
With built-in LP/Quadratic, GRG Nonlinear, Interval Global and Evolutionary Solvers, the Premium Solver Platform can handle virtually any kind of optimization model. But if you need more capacity and speed, choose from six powerful, plug-in Solver Engines:

World-Class Linear/Quadratic Programming and Mixed-Integer Programming.
Choose our LargeScale LP Solver for LP, or our versatile LargeScale SQP Solver for LP and QP models up to 100,000 variables. For even bigger LPs and the world’s toughest LP/MIP problems, choose our XPRESS Solver Engine.

World-Class Smooth Nonlinear Optimization.
Our LargeScale GRG and LargeScale SQP Solvers handle difficult NLP problems with tens of thousands of variables, with remarkable speed thanks to the Platform’s fast automatic differentiation features. For even larger NLPs - and even faster solutions - choose our groundbreaking interior point nonlinear optimizer, the KNITRO Solver Engine.

World-Class Global and Non-Smooth Optimization.
With built-in multistart methods and Interval Global and LGO Global Solvers, the Premium Solver Platform is the world’s best platform for global optimization. And the Platform’s built-in hybrid Evolutionary Solver regularly yields remarkable results on ‘impossible’ non-smooth optimization problems.

Try the Full Solver Squadron Today.
Download fully functional, full capacity and speed, free trial versions (limited to 15 days) of the Premium Solver Platform and all these Solver Engines from our Website. Ask us about academic pricing, DLL versions for your custom applications, and our highly valued annual support contracts.

FRONTLINE SYSTEMS
The Leader in Spreadsheet Optimization
Tel: 01284 748035
Fax: 01284 748021
email: uksales@frontsys.com
web: www.solver.com

Premium Solver Platform, Xpress®, LGO and KNITRO® are trademarks of their respective owners.