16th EuSpRIG Annual Conference
"Spreadsheet Risk Management"

ICAEW, Moorgate, London, 9th July 2015
Proceedings of EuSpRIG 2015 Conference

“Spreadsheet Risk Management”

First published 2015

Printed in United Kingdom by Five Star Printing Ltd, Claydon, Ipswich, Suffolk, UK

Held at the Institute of Chartered Accountants in England and Wales (ICAEW), Moorgate, London.

July 9th 2015

Copyright: European Spreadsheet Risks Interest Group

www.eusprig.org

ISBN: 978-1-905404-52-0

All rights reserved. Apart from fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, no part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form by any means, electronic, mechanical, photocopied, recording or otherwise, without prior permission of the Publisher who is the Committee of the European Spreadsheet Risks Interest Group (www.eusprig.org).

Conference Organised by EuSpRIG Committee

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

No responsibility is assumed by the Publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the materials herein.
EuSpRIG 2015 Conference

"Spreadsheet Risk Management"

EUSPRIG 2015 ORGANISING COMMITTEE

Patrick O'Beirne (Chair), Systems Modelling, Co. Wexford, Ireland
David Colver (Treasurer), Operis, London, UK
Grenville J. Croll, Bury St Edmunds, UK
Simon Thorne, Cardiff Metropolitan University, UK
Angela Collins, (Secretary) Community Health Partnerships

EUSPRIG 2015 PROGRAMME COMMITTEE

Simon Thorne (Chair), Cardiff Metropolitan University, UK
Grenville J. Croll, Spreadsheet Engineering, Bury St Edmunds, UK
Patrick O'Beirne, Systems Modelling, Co. Wexford, Ireland

Proceedings of the EuSpRIG 2015 Conference "Spreadsheet Risk Management"
ISBN : 978-1-905404-52-0
Copyright © 2015, European Spreadsheet Risks Interest Group (www.eusprig.org)
EUSPRIG 2015 CONFERENCE PROGRAMME

Thursday 8th July

08:55 Welcome by EuSpRIG chair Patrick O'Beirne

09:00 A Case Study of spreadsheets in scientific research; Ghada Altarawneh & Simon Thorne, Cardiff Metropolitan University

09:30 Welcome to ICAEW, Andrew Ratcliffe, Deputy President, ICAEW;

09:40 What We Don't Know About Spreadsheet Errors Today: The Facts, Why We Don't Believe Them, and What We Need to Do, Ray Panko, University of Hawai'i

10:30 Sponsors presentation: Successful implementation of Modelling standards across 300 Infrastructure models worth USD 9bn in a fund management environment, Corality Inc.

10:50 Coffee Break

11:20 Problem Solving in Sprego, Maria Csernoch, University of Debrecen

12:00 Developing a repeated model using the structured spreadsheet and implementation methodology, Paul Mireault, Montreal

12:40 Lunch

13:30 Exhibitors; Software Demo Sessions

14:10 Sponsor's Presentation; Rajesh Jain, CIMCON: "Increasing adoption of Spreadsheet Management and Controls in the Real World"

14:30 Auditing Spreadsheets with or without a tool, Simone Schalkwijk, Felienne Hermans, University of Delft

15:10 Development and Experimentation of a Software Tool for Identifying High Risk Spreadsheets for Auditing, Mahmood Shubbak, Simon Thorne, Cardiff Metropolitan University

15:50 Tea Break

16:20 Towards Reusable Spreadsheet Code: Experiments to Create and Interchange Encapsulated Excel Modules, Tom Grossman, Erik Burd, San Francisco State

17:00 A first look at the use of VBA in the Enron corpus; Patrick O'Beirne, Systems Modelling Ltd, Ireland

17:30 AOB

18:00 Conference Closes

19:00 Conference Dinner, courtesy of ICAEW

Proceedings of the EuSpRIG 2015 Conference "Spreadsheet Risk Management"
ISBN : 978-1-905404-52-0
Copyright © 2015, European Spreadsheet Risks Interest Group (www.eusprig.org)
## EUSPRIG 2015 PROCEEDINGS

## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Paper</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maria Csernoch</td>
<td>Problem Solving in Sprego</td>
<td>1</td>
</tr>
<tr>
<td>Paul Mireault</td>
<td>Developing a repeated model using the structured spreadsheet and implementation methodology</td>
<td>15</td>
</tr>
<tr>
<td>Simone Schalkwijk</td>
<td>Auditing Spreadsheets with or without a tool</td>
<td>29</td>
</tr>
<tr>
<td>Felienne Hermans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mahmood Shubbak</td>
<td>Development and Experimentation of a Software Tool for Identifying High Risk Spreadsheets for Auditing</td>
<td>47</td>
</tr>
<tr>
<td>Simon Thorne</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ray Panko</td>
<td>What We Don't Know About Spreadsheet Errors Today: The Facts, Why We Don't Believe Them, and What We Need to Do</td>
<td>79</td>
</tr>
</tbody>
</table>
PREFACE

Dear Colleagues,

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

This year's papers reflect new research testing theories and recommended practices, which has always been our focus. We welcome the spinoff of SEMS, the workshops on Software Engineering Methods in Spreadsheets, where academic researchers further explore the possibilities of applying successful software engineering methods to spreadsheets. I look forward to a more practice-orientated focus in EuSpRIG in future years.

Currently we have about 5,000 unique visitors to our website per month, the most popular page being the horror stories and an active YahooGroup of almost 900 members.

I would like to acknowledge with gratitude our sponsors who have supported us financially and professionally. Our logo sponsors this year are CIMCON, Corality, and SpreadsheetSoftware.com, and I recommend you to pick up their brochures and learn about their services. Our conference dinner sponsor is ICAEW, which I am sure all of you will enjoy.

It is my pleasure to once again acknowledge the keen work of our conference and programme organiser Simon Thorne from Cardiff Met. University. The committee also depends upon the wise counsel and active support of David Colver of Operis, Grenville Croll, Angela Collins (secretary), Morten Siersted of F1F9, and Tie Cheng.

These individuals have contributed a great amount of expertise in the organising of this conference, the publicity, the proceedings, and much more committee work in the background.

The programme of speakers includes a welcome keynote address from the eminent professor Ray Panko on testing, contributions by researchers from the UK, Netherlands, Hungary, Canada and the USA.

Thank you for your interest and participation, and we look forward to a stimulating and interactive conference!

Patrick O'Beirne
Chair EuSpRIG 2015

Patrick@eusprig.org
Problem Solving in Sprego

Maria Csernoch, Piroska Biró
4028 Kassaiút, 26. Debrecen, Hungary
csernoch.maria@inf.unideb.hu, biro.piroska@inf.unideb.hu

ABSTRACT

Sprego is a programming tool for novice and end-user programmers within graphical spreadsheet environments. The main idea of Sprego is to use as few general purpose functions as possible, and based on these functions we create multilevel formulas to solve real world programmable spreadsheet problems. Beyond providing the framework for the theoretic background and the tools which support Sprego, in order to demonstrate the power which lies within it, we present a converted authentic table and, based on this table, data retrieval tasks, their algorithms, and coding in full details.

1 INTRODUCTION

When testing first year students of Informatics, we have found that the spreadsheet is not considered a programming environment. Students think that serious programmers have nothing to do with birotical (office application) software and documents. However, it is not only students but also researchers who consider the management of birotical documents a form of low-level routine knowledge [Bell and Newton, 2013], and a subject which is responsible for the failure to teach computers and informatics, and should be banished from school [Gove, 2012, 2014]. This is definitely not the case. These claims prove that computational thinking [Wing, 2006] and deep approach computer problem solving [Csernoch and Biró, 2014, 2015a, 2015b, 2015c, 2015d] have not reached e-document handling and management.

Our research in testing the different problem solving approaches and methods [Biró et al., 2015a, 2015b; Csernoch and Biró, 2013, 2014, 2015a, 2015b] proved that document management, considered as a form of problem solving, does not differ from problem solving either in programming or in other sciences. The widespread and popular surface approach methods, disguised as user friendly methods, do not work, and they lead to the current high number of error prone documents, which waste human and computer resources and time [Panko, 2008; Tort et al., 2008; Van Deursen and Van Dijk, 2012]. On the other hand, the deep approach methods are proven to be efficient. These methods are concept- or CAAD-based (Computer Algorithmic And Debugging) [Csernoch and Biró, 2014, 2015b], and as such, the problem is in the focus, not the tools.

One of the consequences of the claims of the user friendly spreadsheet environments is that both students and end-users are lost in the enormous number of functions. On one hand, they are taught an extremely high number of functions. On the other hand, they do not understand the descriptions of the functions and their arguments in the wizards and help pages, because they do not have the vocabulary.

When completing the task “List the 15 spreadsheet functions which you think are the most important”, first year students of informatics named 99 functions. To find the reasons for this incredibly high number of “important” functions, we checked the coursebooks written on spreadsheets, and 171 functions were found. Isn’t this a frightening discovery? Further analysis of the spreadsheet coursebooks has led us to the conclusion that most of the authors are not able to distinguish between user manuals and coursebooks. These books do not provide meaningful examples, if they offer any at all;
they do nothing else but list the name and the argument list of the functions. Essentially, they repeat the wizards and the help pages with all their errors.

2 SPREGO

Sprego [Csernoch, 2014; Csernoch and Biró, 2015a, 2015d] is a deep approach problem solving programming method in the spreadsheet environment. It was proved around the early 90’s [Booth, 1992] that functional languages serve as a perfect first language, students being familiar with the concept of function from mathematics [Wakeling, 2007; Sestoft, 2011], and because of the simplicity of the language involved. The great advantage of these languages is that the focus is on the problem, not on the details of the coding [Csernoch and Balogh, 2010; Csernoch, 2012; Csernoch, 2014; Csernoch and Biró, 2015a, 2015d]. Sprego works on already existing tables, consequently, on data management and retrieval, and as such, focuses on the programming aspects of creating formulas. In general, Sprego is a programming tool within spreadsheets, supported by the following tools: Sprego functions, multilevel formulas, array formulas, debugging, and handling authentic sources.

2.1 Sprego functions

We use as few general purpose functions as possible (Table 1). The Sprego functions are classified based on the development made by the students. Sprego1 and Sprego2 consist of the dozen functions which are required. Sprego1 is for beginners, while the functions of Sprego2 are more demanding. Sprego3 is for intermediate and advanced students and end users. We have to emphasize here that all the three groups include general purpose functions.

Table 1. Sprego functions

<table>
<thead>
<tr>
<th>Sprego1</th>
<th>Sprego2</th>
<th>Sprego3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM()</td>
<td>MATCH()</td>
<td>SMALL()</td>
</tr>
<tr>
<td>AVERAGE()</td>
<td>INDEX()</td>
<td>LARGE()</td>
</tr>
<tr>
<td>MIN()</td>
<td>ISERROR()</td>
<td>AND()</td>
</tr>
<tr>
<td>MAX()</td>
<td>OR()</td>
<td></td>
</tr>
<tr>
<td>LEFT()</td>
<td>NOT()</td>
<td></td>
</tr>
<tr>
<td>RIGHT()</td>
<td>ROW()</td>
<td></td>
</tr>
<tr>
<td>LEN()</td>
<td>COLUMN()</td>
<td></td>
</tr>
<tr>
<td>SEARCH()</td>
<td>OFFSET()</td>
<td></td>
</tr>
<tr>
<td>IF()</td>
<td>SUBSTITUTE()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ROUND()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RAND()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INT()</td>
<td></td>
</tr>
</tbody>
</table>

2.2 Multilevel formulas

Due to the low number of Sprego functions there is a need to create multilevel formulas. Multilevel formulas are required to solve real programming tasks.
2.3 Array formulas

Array formulas are those formulas whose output is an array, or which accept array(s) as argument(s), while the default arguments are single values. In the first case the output of the formula is an array—array result array formulas, ARAF—, while in the second case both array and single result—single result array formulas, SRAF—outputs are possible. With the use of array formulas, the copying of formulas and its consequences, which is one of the main sources of errors, is ruled out.

2.4 Debugging

The first three tools evoke a fourth one, debugging. The combination of the Sprego functions with the multilevel array formulas makes thorough debugging available in spreadsheets. Similar to high level programming languages, both manual and spreadsheet supported debugging can be carried out.

2.5 Authentic sources

By using authentic tables in the teaching of Sprego we would avoid the fiasco of other school programming languages. Most of the students find the programming languages designed for educational purposes useless. They are not able to transfer knowledge from the school programs to solve real word problems in other environments. With Sprego we can provide the students with real world problems; consequently, there is no gap between the student and the end user statuses.

In Sprego programming we follow the phases of the well accepted deep approach problem solving methods [Polya, 1954; Booth, 1992; Case&Gunstone, 2002; IEEE&ACM Report, 2013; Csernoch&Biró, 2013, 2014, 2015b]: (1) Based on the available data and being aware of the expected output we create a plan, (2) following this we carry out the coding, and the final step is (3) the discussion of the problem.

- Familiarity: Understanding the concept, seeing clearly what is required, seeing how the various items are connected, how the unknown is connected to the data, building the algorithm.
- Usage: Carrying out the plan, which, in a programming environment, is the coding of the algorithm.
- Assessment: Looking back at the complete solution, considering a concept from multiple viewpoints, justifying the selection of a particular approach, discussing, debugging it.

In the following section we provide a table downloaded from the LOL (League of Legends) message board [LOL, 2015] converted to a spreadsheet table. Based on these data, we present several tasks (Tasks 1–6) with their detailed solutions. Solving these problems we follow strictly the consecutive steps of deep approach problem solving: (1) understanding the problem, the concept, (2) building the algorithm, (3) coding, and (4) debugging.

3 TASKS AND THEIR SPREGO SOLUTIONS

The converted LOL message board table consists of five columns and 1001 rows, where the first row holds the column titles (Figure 1 and Figure 2). The five columns consist of the following data.
- A: The title of the message.
- B: The name of the account and its server, where the message arrived from.
- C: The classification of the message (theme) and how long ago it was posted.
- D: The number of comments (NOF comments), with an accompanying string.
- E: The number of views (NOF views), with an accompanying string.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Title</td>
<td>Account (server)</td>
</tr>
<tr>
<td>2</td>
<td>Arcade Sona's future</td>
<td>Reisenh (EUW)</td>
</tr>
<tr>
<td>3</td>
<td>Season 5 changes have made League</td>
<td>Maximum Kawaii (EUNE)</td>
</tr>
<tr>
<td>4</td>
<td>THE EU FORUM AWARDS</td>
<td>Riot Draggles (EUW)</td>
</tr>
<tr>
<td>5</td>
<td>[BUG] decreased framerate patch 5.4</td>
<td>Papa Lovegood (EUW)</td>
</tr>
<tr>
<td>6</td>
<td>&quot;(&quot; ) URGOT BUFFS ON PBE (&quot; )&quot;</td>
<td>Dahaka66 (EUW)</td>
</tr>
<tr>
<td>7</td>
<td>Feedback to riot employees (youtube.com)</td>
<td>proiaplegamer (EUNE)</td>
</tr>
<tr>
<td>8</td>
<td>vulgar</td>
<td>BBS CursedSoul (EUW)</td>
</tr>
<tr>
<td>9</td>
<td>Give riven energy.</td>
<td>DarkSlicedOfcake (EUW)</td>
</tr>
<tr>
<td>10</td>
<td>Try the champion before you buy them</td>
<td>CandyLandRemixed (EUW)</td>
</tr>
<tr>
<td>11</td>
<td>i always flame. What should i do?</td>
<td>filojistoNNN (EUW)</td>
</tr>
<tr>
<td>12</td>
<td>@ RIOT</td>
<td>trojanfighter (EUW)</td>
</tr>
<tr>
<td>13</td>
<td>Urgot Rework Idea</td>
<td>MB Ghost 2 Ghost (EUW)</td>
</tr>
<tr>
<td>14</td>
<td>New splash arts are WORSE!</td>
<td>GingerPawar (EUW)</td>
</tr>
<tr>
<td>15</td>
<td>My account has been changed to Russia</td>
<td>due to the most liked</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>NinjaJesus729 (EUW)</td>
</tr>
</tbody>
</table>

Figure 1. Columns A–B of the converted LOL table

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Theme and time</td>
<td>NOF comments</td>
<td>NOF views</td>
</tr>
<tr>
<td>2</td>
<td>Champions &amp; Gameplay about 19 hours</td>
<td>14 new</td>
<td>680 Views</td>
</tr>
<tr>
<td>3</td>
<td>Champions &amp; Gameplay about 8 hours</td>
<td>14 new</td>
<td>149 Views</td>
</tr>
<tr>
<td>4</td>
<td>Forum Games &amp; Contests about 24 hours</td>
<td>32 new</td>
<td>1.1k Views</td>
</tr>
<tr>
<td>5</td>
<td>Suggestions &amp; Bug Reports about 6 days</td>
<td>4 new</td>
<td>59 Views</td>
</tr>
<tr>
<td>6</td>
<td>Champions &amp; Gameplay about 17 hours</td>
<td>11 new</td>
<td>412 Views</td>
</tr>
<tr>
<td>7</td>
<td>Off-topic about 22 hours</td>
<td>13 new</td>
<td>269 Views</td>
</tr>
<tr>
<td>8</td>
<td>Champions &amp; Gameplay about 15 hours</td>
<td>5 new</td>
<td>82 Views</td>
</tr>
<tr>
<td>9</td>
<td>Champions &amp; Gameplay 2 days ago</td>
<td>125 new</td>
<td>2.5k Views</td>
</tr>
<tr>
<td>10</td>
<td>Community Creations about 7 hours</td>
<td>7 new</td>
<td>131 Views</td>
</tr>
<tr>
<td>11</td>
<td>Player Behaviour 25 minutes ago</td>
<td>9 new</td>
<td>39 Views</td>
</tr>
<tr>
<td>12</td>
<td>Player Behaviour about an hour</td>
<td>0 new</td>
<td>11 Views</td>
</tr>
<tr>
<td>13</td>
<td>Champions &amp; Gameplay a day ago</td>
<td>8 new</td>
<td>52 Views</td>
</tr>
<tr>
<td>14</td>
<td>Off-topic 3 days ago</td>
<td>77 new</td>
<td>1.6k Views</td>
</tr>
<tr>
<td>15</td>
<td>Help &amp; Support 3 days ago</td>
<td>1 new</td>
<td>147 Views</td>
</tr>
</tbody>
</table>

Figure 2. Columns C–E of the converted LOL table

3.1 Array result array formulas

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Write out the name of the account in a separate column.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution 1</td>
<td>S1–S3, Table 2.</td>
</tr>
</tbody>
</table>
Characteristics of Task 1

- Accounts are on the left side of the strings.
- They are of different lengths.
- Accounts are followed by a Space and an opening parenthesis characters.

Algorithm of Task 1

- Deciding on the position of the ( character. Output: a number. S1
- Calculating the length of the account. Output: a number. S2
- Cutting out the account. Output: a string. S3

Coding of Task 1

S1. `{=FIND("(",C2:C1001)}`
S2. `{=FIND("(",C2:C1001)-2}`
S3. `{=LEFT(C2:C1001,FIND("(",C2:C1001)-2))}

Table 2. The input and consecutive outputs of the solution of Task 1

<table>
<thead>
<tr>
<th>Account (server)</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReisenII (EUW)</td>
<td>10</td>
<td>8</td>
<td>ReisenII</td>
</tr>
<tr>
<td>Maximum Kawaii (EUNE)</td>
<td>16</td>
<td>14</td>
<td>Maximum Kawaii</td>
</tr>
<tr>
<td>Riot Draggles (EUW)</td>
<td>15</td>
<td>13</td>
<td>Riot Draggles</td>
</tr>
<tr>
<td>Papa Lovegood (EUW)</td>
<td>15</td>
<td>13</td>
<td>Papa Lovegood</td>
</tr>
<tr>
<td>DahakaGG (EUW)</td>
<td>10</td>
<td>8</td>
<td>DahakaGG</td>
</tr>
<tr>
<td>Proapllegamer (EUNE)</td>
<td>15</td>
<td>13</td>
<td>proapllegamer</td>
</tr>
<tr>
<td>BBS CursedSoul (EUW)</td>
<td>16</td>
<td>14</td>
<td>BBS CursedSoul</td>
</tr>
<tr>
<td>DarkSliceOfCake (EUW)</td>
<td>17</td>
<td>15</td>
<td>DarkSliceOfCake</td>
</tr>
<tr>
<td>CandyLandRemixed (EUW)</td>
<td>18</td>
<td>16</td>
<td>CandyLandRemixed</td>
</tr>
<tr>
<td>filojistoNNN (EUW)</td>
<td>14</td>
<td>12</td>
<td>filojistoNNN</td>
</tr>
<tr>
<td>trojanfighter (EUW)</td>
<td>15</td>
<td>13</td>
<td>trojanfighter</td>
</tr>
<tr>
<td>MB Ghost 2 Ghost (EUW)</td>
<td>18</td>
<td>16</td>
<td>MB Ghost 2 Ghost</td>
</tr>
<tr>
<td>GingerPowar (EUW)</td>
<td>13</td>
<td>11</td>
<td>GingerPowar</td>
</tr>
<tr>
<td>NinjaJesus720 (EUW)</td>
<td>15</td>
<td>13</td>
<td>NinjaJesus720</td>
</tr>
</tbody>
</table>

Task 2

Write out the number of comments without the text.

Solution 2

Solution: S4–S7, Table 3

Characteristics of Task 2

- Numbers
- They are on the left side of the string
- They have different numbers of digits.
- Following the numbers there is a Space and an n character (the first character of new comments).
Algorithm of Task 2

- Finding the position of the n character. Output: a number.S4
- Calculating the length of the number. Output: a number.S5
- Cutting out the number from the original string. Output: a string.S6
- Converting the text into a number. Output: a number.S7

Coding of Task 2

S4. `{=FIND("new",E2:E1001))`  
S5. `{=FIND("new",E2:E1001)-2}`  
S6. `{=LEFT(E2:E1001,FIND("new",E2:E1001)-2))`  
S7. `{=LEFT(E2:E1001,FIND("new",E2:E1001)-2)*1}`

Table 3. The input and consecutive outputs of the solution of Task 2

<table>
<thead>
<tr>
<th>NOF comments</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 new Comments</td>
<td>4</td>
<td>2</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>14 new Comments</td>
<td>4</td>
<td>2</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>32 new Comments</td>
<td>4</td>
<td>2</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>4 new Comments</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>11 new Comments</td>
<td>4</td>
<td>2</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>13 new Comments</td>
<td>4</td>
<td>2</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>5 new Comments</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>125 new Comments</td>
<td>5</td>
<td>3</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>7 new Comments</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>9 new Comments</td>
<td>3</td>
<td>1</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>0 new Comments</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8 new Comments</td>
<td>3</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>77 new Comments</td>
<td>4</td>
<td>2</td>
<td>77</td>
<td>77</td>
</tr>
<tr>
<td>1 new Comment</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Task 3  
Solution 3  
Write out the name of the server.  
Solution: S8–S12,  
Table 4

Characteristics of Task 3

- Servers are on the right side of the original string.  
- They are in a pair of parentheses.  
- They are texts.  
- They are of different lengths.

Algorithm of Task 3

- Calculating the length of the original string, with account and server together. Output: a number.S8
– Calculating the difference between the length of the text and the position of the opening parenthesis. Output: a number.S9
– Cutting out the name of the server and the closing parenthesis from the right side of the string. Output: a string.S10
– Calculating the length of the short string. Output: a number.S11
– Cutting out the name of the server. Output: a string.S12

Coding of Task 3

S8. {=LEN(C2:C1001)}
S9. {=LEN(C2:C1001)-FIND("",C2:C1001)}
S10. {=RIGHT(C2:C1001,LEN(C2:C1001)-FIND("",C2:C1001))}
S11. {=LEN(RIGHT(C2:C1001,LEN(C2:C1001)-FIND("",C2:C1001))))}
S12. {=LEFT(RIGHT(C2:C1001,LEN(C2:C1001)-FIND("",C2:C1001)),LEN(RIGHT(C2:C1001,LEN(C2:C1001)-FIND("",C2:C1001)))-1)}

Table 4. The input and consecutive outputs of the solution of Task 3

<table>
<thead>
<tr>
<th>Account (server)</th>
<th>S8</th>
<th>S9</th>
<th>S10</th>
<th>S11</th>
<th>S12</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReisenII (EUW)</td>
<td>14</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>Maximum Kawaii (EUNE)</td>
<td>21</td>
<td>5</td>
<td>EUNE</td>
<td>5</td>
<td>EUNE</td>
</tr>
<tr>
<td>Riot Draggles (EUW)</td>
<td>19</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>Papa Lovegood (EUW)</td>
<td>19</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>DahakaGG (EUW)</td>
<td>14</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>proaplegamer (EUNE)</td>
<td>20</td>
<td>5</td>
<td>EUNE</td>
<td>5</td>
<td>EUNE</td>
</tr>
<tr>
<td>BBS CursedSoul (EUW)</td>
<td>20</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>DarkSliceOfCake (EUW)</td>
<td>21</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>CandyLandRemixed (EUW)</td>
<td>22</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>filojistoNNN (EUW)</td>
<td>18</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>trojanfighter (EUW)</td>
<td>19</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>MB Ghost 2 Ghost (EUW)</td>
<td>22</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>GingarPowar (EUW)</td>
<td>17</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
<tr>
<td>NinjaJesus720 (EUW)</td>
<td>19</td>
<td>4</td>
<td>EUW</td>
<td>4</td>
<td>EUW</td>
</tr>
</tbody>
</table>

Task 4
Solution 4
Write out the number of views without the text.
Solution: S13–S18,
Table 5, S1–S5, Table 6

Characteristics of Task 4

– Numbers followed by the string Views.
– Two different kinds of numbers: (1) whole numbers, (2) real numbers in thousands, marked by k.
– Numbers are on the left side of the original string.
Algorithm of Task 4 (1st section)

- Finding the position of the V character (the first character of Views). Output: a number. S13
- Calculating the length of the number. Output: a number. S14
- Cutting out the number from the original string. Output: a string. S15
- Converting the string to a number. Two different outputs: (1) a number, (2) an error message. The formula works on the whole numbers. S16
- Shortening the S15 string by one character. The original purpose of the removal of the k character from the end of the thousand numbers. Two different outputs: (1) whole numbers one digit shorter, (2) thousand numbers, as a real number [without the k character]. S17
- Multiplying the numbers by 1000. Two different outputs: whole numbers 100 times greater than the original value, (2) kilos are converted to a rounded number. S18

Coding of Task 4 [1st section]

S13. `{=FIND("V",F2:F1001)}`
S14. `{=FIND("V",F2:F1001)-2}`
S15. `{=LEFT(F2:F1001,FIND("V",F2:F1001)-2)}`
S16. `{=LEFT(F2:F1001,FIND("V",F2:F1001)-2)*1}`
S17. `{=LEFT(F2:F1001,FIND("V",F2:F1001)-3)*1}`
S18. `{=LEFT(F2:F1001,FIND("V",F2:F1001)-3)*1000}`

Table 5. The input and consecutive outputs (S13–S18) of the solution of Task 4

<table>
<thead>
<tr>
<th>NOF Views</th>
<th>S13</th>
<th>S14</th>
<th>S15</th>
<th>S16</th>
<th>S17</th>
<th>S18</th>
</tr>
</thead>
<tbody>
<tr>
<td>680 Views</td>
<td>5</td>
<td>3</td>
<td>680</td>
<td>680</td>
<td>68</td>
<td>68000</td>
</tr>
<tr>
<td>149 Views</td>
<td>5</td>
<td>3</td>
<td>149</td>
<td>149</td>
<td>14</td>
<td>14000</td>
</tr>
<tr>
<td>1.1k Views</td>
<td>6</td>
<td>4</td>
<td>1.1k</td>
<td>#VALUE!</td>
<td>1.1</td>
<td>1100</td>
</tr>
<tr>
<td>59 Views</td>
<td>4</td>
<td>2</td>
<td>59</td>
<td>59</td>
<td>5</td>
<td>5000</td>
</tr>
<tr>
<td>412 Views</td>
<td>5</td>
<td>3</td>
<td>412</td>
<td>412</td>
<td>41</td>
<td>41000</td>
</tr>
<tr>
<td>269 Views</td>
<td>5</td>
<td>3</td>
<td>269</td>
<td>269</td>
<td>26</td>
<td>26000</td>
</tr>
<tr>
<td>82 Views</td>
<td>4</td>
<td>2</td>
<td>82</td>
<td>82</td>
<td>8</td>
<td>8000</td>
</tr>
<tr>
<td>2.5k Views</td>
<td>6</td>
<td>4</td>
<td>2.5k</td>
<td>#VALUE!</td>
<td>2.5</td>
<td>2500</td>
</tr>
<tr>
<td>131 Views</td>
<td>5</td>
<td>3</td>
<td>131</td>
<td>131</td>
<td>13</td>
<td>13000</td>
</tr>
<tr>
<td>39 Views</td>
<td>4</td>
<td>2</td>
<td>39</td>
<td>39</td>
<td>3</td>
<td>3000</td>
</tr>
<tr>
<td>11 Views</td>
<td>4</td>
<td>2</td>
<td>11</td>
<td>11</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>52 Views</td>
<td>4</td>
<td>2</td>
<td>52</td>
<td>52</td>
<td>5</td>
<td>5000</td>
</tr>
<tr>
<td>1.6k Views</td>
<td>6</td>
<td>4</td>
<td>1.6k</td>
<td>#VALUE!</td>
<td>1.6</td>
<td>1600</td>
</tr>
<tr>
<td>147 Views</td>
<td>5</td>
<td>3</td>
<td>147</td>
<td>147</td>
<td>14</td>
<td>14000</td>
</tr>
</tbody>
</table>

Algorithm of Task 4 [2nd section]

- Searching for the k character in the string. Two different outputs: (1) error message with whole numbers, (2) a number, the position of the k character with real numbers. S1
– Checking the output of the formula which searches for the k character. The question is whether the k is found or not. Output: (1) TRUE, with the whole numbers, no k was found, the function searching for k returned with an error, (2) FALSE, with the real numbers, the function searching for k returned with a correct output, there is no error. S2

– Setting a yes/no question based on the results of the check for an error. Output: zeros, because the question is set, but none of the answers. S3

– Setting the output, if the answer is yes to the question. The answer is copied from S16. Output: (1) whole numbers and (2) 0s. S4

– Setting the output, if the answer is no to the question. The answer is copied from S17. Output: (1) whole numbers and (2) numbers rounded to hundreds. S5

Table 6. The second section (S1–S5) of the solution of Task 4

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>680</td>
<td>680</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>149</td>
<td>149</td>
</tr>
<tr>
<td>4</td>
<td>FALSE</td>
<td>0</td>
<td>0</td>
<td>1100</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>412</td>
<td>412</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>269</td>
<td>269</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td>4</td>
<td>FALSE</td>
<td>0</td>
<td>0</td>
<td>2500</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>131</td>
<td>131</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>52</td>
<td>52</td>
</tr>
<tr>
<td>4</td>
<td>FALSE</td>
<td>0</td>
<td>0</td>
<td>1600</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>TRUE</td>
<td>0</td>
<td>147</td>
<td>147</td>
</tr>
</tbody>
</table>

Coding of Task 4 [2nd section]

S1. `{=FIND("k",F2:F1001)}`
S2. `{=ISERROR(FIND("k",F2:F1001))}`
S3. `{=IF(ISERROR(FIND("k",F2:F1001)),,)}
S4. `{=IF(ISERROR(FIND("k",F2:F1001)),LEFT(F2:F1001,FIND("V",F2:F1001)-2)*1,)}
S5. `{=IF(ISERROR(FIND("k",F2:F1001)),LEFT(F2:F1001,FIND("V",F2:F1001)-3)*1000)}`

3.2 Conditional single result array formulas [CSRAF]

One of the most error prone classes of functions is the conditional built-in functions, found in different categories in spreadsheets. For a more convenient reference, we have created the *IF?() expression. The class of *IF?() functions holds all the conditional spreadsheet functions. To handle conditions the database function would also be appropriate, but they are even more burdensome than the *IF?() functions.

CSRAF scan substitute either the *IF?() or the database functions. Due to the limitations and the inconsistencies of these built-in functions [Csernoch, 2014; Csernoch and Biró; 2015a] CSRAFs offer a much wider variety of options, consequently they can be used to
solve problems which we have never considered within the traditional spreadsheet framework.

<table>
<thead>
<tr>
<th>Task 5</th>
<th>Type a number in H1003. How many messages have received more than H1003 views? (We use the output vector of Task 4, stored in column I.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution 5</td>
<td>Solution: S6–S8, Table 7</td>
</tr>
</tbody>
</table>

**Characteristics of Task 5**

- We have to separate those messages which have received more than H1003 views from those which have received fewer.
- We have to mark those messages which have received more than H1003 views. Each time a message is received we draw a little stick on a piece of paper. On a computer the easiest method is to store 1 for each match. If there is no match we leave the message unnoticed.

**Algorithm of Task 5**

- Asking 1000 questions whether the “NOF views” is greater than H1003, or not. Output: a vector of 1000 components of TRUE’s and FALSE’s, whose first component is displayed in the cell.S6
- Deciding on the output if the answer is yes: 1. Deciding on the output if the answer is no: default FALSE. Output: a vector of 1s and FALSE’s, whose first component is displayed in the cell.S7
- Summing the components of the vector. Output: a whole number, the number of 1s stored in the vector.S8

**Coding of Task 5**

S6. \{=I2:I1001>H1003\}
S7. \{=IF(I2:I1001>H1003,1)\}
S8. \{=SUM(IF(I2:I1001>H1003,1))\}

Table 7. The first component of the output vector with two possible inputs (H1003) in Task 5

<table>
<thead>
<tr>
<th>NOF Views</th>
<th>H1003</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
</tr>
</thead>
<tbody>
<tr>
<td>680 Views</td>
<td>500</td>
<td>TRUE</td>
<td>1</td>
<td>47</td>
</tr>
<tr>
<td>680 Views</td>
<td>1600</td>
<td>FALSE</td>
<td>FALSE</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task 6</th>
<th>Type a server in G1004. Give the average and maximum of comments to messages arriving from the G1004 server. (We use the output vector of Task 3, servers stored in column G, and the output vector of Task 2, NOF comments stored in column H.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution 6</td>
<td>Solution: S9–S11/S12, Table 8</td>
</tr>
</tbody>
</table>

To solve Task 6 we use exactly the same algorithm as we did in Task 5.
Characteristics of Task 6

- We need to store only those “NOF comments” which arrived to messages coming from the server given in G1004.
- The average/maximum of these numbers should be calculated.

Algorithm of Task 6

- Asking the question. Output: a vector of TRUE’s and FALSE’s, whose first component is displayed in the cell.S9
- Deciding on the output of questions. Output: a vector of whole numbers [NOF comments] and FALSE’s, whose first component is displayed in the cell.S10
- Calculating the average/maximum of the components of the vector. Output: a real/whole number, the average/maximum of “NOF comments” stored in the vector.S11/S12

Coding of Task 6

S9. \{=G2:G1001=G1004\}
S10. \{=IF(G2:G1001=G1004,H2:H1001)\}
S11. \{=AVERAGE(IF(G2:G1001=G1004,H2:H1001))\}
S12. \{=MAX(IF(G2:G1001=G1004,H2:H1001))\}

Table 8: The first component of the output vector with two possible inputs (G1004) in Task 6

<table>
<thead>
<tr>
<th>Account (server)</th>
<th>G1004</th>
<th>S9</th>
<th>S10</th>
<th>S11</th>
<th>S12</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReisenII (EUW)</td>
<td>EUW</td>
<td>TRUE</td>
<td>14</td>
<td>5.17</td>
<td>130</td>
</tr>
<tr>
<td>ReisenII (EUW)</td>
<td>EUWE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>4.59</td>
<td>76</td>
</tr>
</tbody>
</table>

4 CONCLUSION

Sprego is a programming method within the spreadsheet framework. The main characteristics of Sprego are the use of general purpose functions and the building of multilevel formulas based on these functions. Our testing and analyses [Biró and Csernoch, 2013a, 2013b, 2014a, 2014b; Csernoch and Biró, 2013, 2014] have proved that Sprego can be used effectively both as a first programming language and as the language of end-user programmers. Due to the simplicity of the language and the few functions in use, with Sprego we can build up knowledge in long term memory, which would lead to less error prone documents. Beyond the reliability and stability of the documents, Sprego formulas are version and application independent. The documents can be freely opened both in MS Excel and OpenOffice/LibreOfficeCalc, without the need to check the versions of these programs.

In Sprego programming we handle problems in a way that is well accepted in traditional sciences and traditional programming languages, where the deep approach methods have proved effective. Sprego fulfills all the requirements of the deep approach metacognitive problem solving methods. The tasks and their solutions presented in this paper clearly demonstrate that programming in non-traditional environments would be as effective as in traditional programming languages.
5 ACKNOWLEDGEMENT

The research was supported partly by the TÁMOP-4.1.2.B.2-13/1-2013-0009, SZAKTÁRNÉT projects. The project has been supported by the European Union, co-financed by the European Social Fund. The research was supported partly by the Hungarian Scientific Research Fund under Grant No. OTKA K-105262.

REFERENCES & BIBLIOGRAPHY


Copyright © 2015, European Spreadsheet Risks Interest Group (www.eusprig.org) & the Author(s)


Developing a Repeating Model Using the Structured Spreadsheet Modelling and Implementation Methodology

Paul Mireault
Founder, SSMI International
Honorary Professor, HEC Montréal
Paul.Mireault@SSMI.International

ABSTRACT

Spreadsheets often have variables and formulas that are similar, differing only by the fact that they refer to different instances of an entity. For example, the calculation of the sales revenues of the South and East regions are Revenues South = Price*Quantity Sold South and Revenues East =Price*Quantity Sold East. In this paper, we present a conceptual modelling approach that takes advantage of these similarities and leads the spreadsheet developer to the formula Revenues = Price*Quantity. We then present simple but strict rules to implement the spreadsheet.

1 INTRODUCTION

Errors in spreadsheets have caused financial losses for many companies and organizations, as illustrated by the EuSpRIG Horror Stories [EuSpRIG, 2015] web page. Panko [Panko, 2008] cites a study reporting that 95% of spreadsheets have errors.

Many authors have identified the following spreadsheet characteristics that can cause errors:

• Far references. A formula that references a cell that is not immediately visible and understood is harder to understand [Raffensperger, 2003].

• Transitive references. Formulas that reference a reference of a variable are candidates to maintenance problems. When we introduce a nuance and create a new variable, formulas may refer to one or the other nuance of the variable.

In [Mireault, 2015], we introduced the Structured Spreadsheet Modelling and Implementation methodology and illustrated it with a simple problem. The methodology is based on well-established concepts of Computer Science, Software Engineering and Information Systems. The basic idea of the methodology is to develop spreadsheets in two steps, the conceptual model first and the implementation second.

One of the important conception rules is to keep formulas as simple as they can be, avoiding having more than one mathematical operator or function in the definition of a variable. For example, the formula Total Cost = Fixed Cost + Unit Cost * Quantity uses two different mathematical operators, addition and multiplication, and should be replaced with Variable Cost = Unit Cost * Quantity and Total Cost = Fixed Cost + Variable Cost. Such simpler formulas would have a low complexity score according to [Hermans, et al., 2012].

[Mireault, et al., 2015] showed how a model developed with the SSMI methodology can be easily expanded to transform parameters that were entered by user into variables that are calculated from other inputs.
In this paper, we present an extension to the SSMI methodology that is used to model cases where a set of formulas and variables is repeated for different instances of an entity. For example, the calculation of the total cost is similar for all our regions, South, East and North.

2 MODELLING A REPEATING SUB-MODEL

We are sometimes faced with the situation where sets of variables have formulas that are similar. In such situations, we are also tempted to name the variables we create with the same prefix and differentiate them with a different suffix. For example, we might have variables named Profit Region A, Profit Region B and Profit Region C. If we use the straightforward modeling technique presented in [Mireault, 2015], we will end up with a model that is unwieldy and difficult to modify. But there is a way of keeping the model simple: it consists in identifying variables and formulas that are similar and grouping them in what we will call a repeating sub-model.

We will first illustrate the development of a model without the use of a repeating sub-model to illustrate its complexity. The reader should keep in mind that this is not the proper modeling technique. We will then illustrate the proper use of the repeating sub-model and show how it simplifies the Formula Diagram and the Formula List.

Let’s consider a small example with Marco’s Widgets. Marco sells his widgets in three different regions: South, East and North. He wants a spreadsheet that will show him the profit per region as well as the total profit. To allocate the demand per region, he tells you that the demand has traditionally been 48%, 23% and 29% respectively for the South, East and North regions. Marco uses a Unit Cost composed of a Manufacturing Cost and a Delivery Cost. The Manufacturing Cost does not depend on the region and is equal to 120$. The Delivery Cost is equal to 50$, 80$ and 60$ respectively for the South, East and North regions. To calculate each region’s profit, the Fixed Cost will be allocated to each region with the same distribution as the demand. We now have all the information needed to design the model.

2.1 Building the model for the South region

*Figure 1 presents the Formula Diagram for the South region only. The corresponding Formula List is shown in Table 1.*

Now, if we continue on with the East region, we will notice that the variables and the formulas are similar. The only difference is that the variables will use the *East* suffix instead of *South*.

It will also be the same with the North-region and the *North* suffix. We would end up with a model that looks like Figure 2.
**Figure 1 - Formula Diagram for the South region**

**Table 1 - Formula List of the South region model**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>Average price of widgets</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>Profit South</td>
<td>Profit of the South region</td>
<td>Output</td>
<td>= Revenue South - Total Cost South</td>
</tr>
<tr>
<td>DemParA</td>
<td>First Demand function parameter</td>
<td>Parameter</td>
<td>367,000</td>
</tr>
<tr>
<td>DemParB</td>
<td>Second Demand function parameter</td>
<td>Parameter</td>
<td>1.0009</td>
</tr>
<tr>
<td>Fixed Cost</td>
<td>Fixed cost of manufacturing the widgets</td>
<td>Parameter</td>
<td>2,500,000$</td>
</tr>
<tr>
<td>Mfg Cost</td>
<td>Cost of manufacturing one widget</td>
<td>Parameter</td>
<td>120$</td>
</tr>
<tr>
<td>Dist South</td>
<td>Proportion of the Demand sold in the South region</td>
<td>Parameter</td>
<td>49%</td>
</tr>
<tr>
<td>Delivery Cost South</td>
<td>Cost of delivery of widgets in the South region</td>
<td>Parameter</td>
<td>50$</td>
</tr>
<tr>
<td>Demand</td>
<td>Demand of widgets, formula given by the market research specialist</td>
<td>Calculated</td>
<td>= DemParA * DemParB^-Price</td>
</tr>
<tr>
<td>Demand South</td>
<td>Portion of the Demand sold in the South region</td>
<td>Calculated</td>
<td>= Demand * Dist South</td>
</tr>
<tr>
<td>Total Cost South</td>
<td>Total Cost of selling widgets in the South region</td>
<td>Calculated</td>
<td>= Fixed Cost South + Variable Cost South</td>
</tr>
<tr>
<td>Fixed Cost South</td>
<td>Portion of the Fixed cost allocated to the South region</td>
<td>Calculated</td>
<td>= Fixed Cost * Dist South</td>
</tr>
<tr>
<td>Variable Cost South</td>
<td>Variable Cost of the widgets sold in the South region</td>
<td>Calculated</td>
<td>= Demand South * Unit Cost South</td>
</tr>
<tr>
<td>Unit Cost South</td>
<td>Unit cost of one widget in the South region</td>
<td>Calculated</td>
<td>= Mfg Cost + Delivery Cost South</td>
</tr>
<tr>
<td>Revenue South</td>
<td>Revenue of the South region</td>
<td>Calculated</td>
<td>= Demand South * Price</td>
</tr>
</tbody>
</table>
2.2 Creating a repeating sub-model

The model of Figure 2 has obvious shortcomings. The most important one is that it does not scale well. Imagine how it would look like if we had to expand it to cover more regions, like provinces or states in many countries. Canada has 10 provinces, India has 28 or more states, the USA has 50 states, France has 100 departments; expanding the model to cover the divisions of many countries is practically infeasible. Another important shortcoming is that any modification, like adding variables, has to be replicated many times. This increases the risk of introducing errors, which is something we want to avoid.

The key to the repeating sub-model lies in the variables that have a suffix. Instead of having one copy of a variable for each region, we will use one variable representing any region. The variables Delivery Cost South, Delivery Cost East and DeliveryCostNorth will be replaced by Delivery Cost. Thus, Delivery Cost is now a variable with multiple values: it is the set of 3 values \{50, 80, 60\}.

When the same variable name would appear more than once, we need to use an adjective to identify its role. For instance, we have two variables that represent the
demand: the variable Demand that represents a single value and the variable Demand that represents multiple values will be renamed Total Demand and Regional Demand respectively.

We will apply the same treatment to the other variables. The repeating variables are Distribution, Regional Demand, Delivery Cost, Unit Cost, Revenue, VariableCost, Regional Fixed Cost, Total Cost and Profit.

We represent the repeating sub-model with a box with a dash border in the Formula Diagram. We write the name of the repeating entity in the top right corner of the box. Now, we place all the variables that have multiple values, depending on the Region, inside the sub-model box and all the other variables outside the box. This is illustrated in Figure 3.

![Figure 3 - Formula Diagram with the region sub-model](image)

All the variables outside the box represent single values, and all the variables inside the box represent multiple values.

This method of representing a repeating sub-model takes care of the two major shortcomings we mentioned earlier. First, the model is now scalable as it does not get more complicated with an increase in the number of regions. In fact, the model does not change at all when we increase the number of regions. Naturally, the spreadsheet implementation will be bigger with more regions, but we will show a straightforward way of expanding the spreadsheet implementation to accommodate more regions.

Second, modifying the model is simplified by the fact that parts of the model are not repeated many times. Any change needs only to be made once, no matter how many regions there are.

We calculate the Total Profit from the Profit of each region. Since it is inside the repeating sub-model box, the variable Profit represents a set of values, and since it is outside the box, the variable Total Profit represents a single value. Thus, the function that calculates Total Profit must be a function that takes a set of values and returns a single value. There are a few such functions, called aggregate functions, and you are already familiar with some them: SUM,
AVERAGE, MINIMUM, MAXIMUM, VARIANCE, and STANDARD DEVIATION. There are others that are more specialized, like NPV (Net Present Value) and IRR (Internal Rate of Return) often used in Finance and Accounting models.

In our case, we will use the formula Total Profit = SUM(Profit). We add the following entry in our Table of Formulas:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>Average price of widgets</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>Profit</td>
<td>Profit of each region</td>
<td>Output, repeating</td>
<td>= Revenue - Total Cost</td>
</tr>
<tr>
<td>DemParA</td>
<td>First Demand function parameter</td>
<td>Parameter</td>
<td>367,000</td>
</tr>
<tr>
<td>DemParB</td>
<td>Second Demand function parameter</td>
<td>Parameter</td>
<td>1.00009</td>
</tr>
<tr>
<td>Fixed Cost</td>
<td>Fixed cost of manufacturing the widgets</td>
<td>Parameter</td>
<td>2,500,000$</td>
</tr>
<tr>
<td>Mfg Cost</td>
<td>Cost of manufacturing one widget</td>
<td>Parameter</td>
<td>120$</td>
</tr>
<tr>
<td>Distribution</td>
<td>Proportion of the Demand sold in each region</td>
<td>Parameter, repeating</td>
<td>48%, 23%, 29%</td>
</tr>
<tr>
<td>Delivery Cost</td>
<td>Cost of delivery of widgets in each region</td>
<td>Parameter, repeating</td>
<td>50$, 80$, 60$</td>
</tr>
<tr>
<td>Total Demand</td>
<td>Demand of widgets, formula given by the market research specialist</td>
<td>Calculated</td>
<td>= DemParA * DemParB^Price</td>
</tr>
<tr>
<td>Regional Demand</td>
<td>Portion of the Demand sold in each region</td>
<td>Calculated, repeating</td>
<td>= Total Demand * Distribution</td>
</tr>
<tr>
<td>Total Cost</td>
<td>Total Cost of selling widgets in each region</td>
<td>Calculated, repeating</td>
<td>= Regional Fixed Cost + Variable Cost</td>
</tr>
<tr>
<td>Regional Fixed Cost</td>
<td>Portion of the Fixed cost allocated to each region</td>
<td>Calculated, repeating</td>
<td>= Fixed Cost * Distribution</td>
</tr>
<tr>
<td>Variable Cost</td>
<td>Variable Cost of the widgets sold in each region</td>
<td>Calculated, repeating</td>
<td>= Regional Demand * Unit Cost</td>
</tr>
<tr>
<td>Unit Cost</td>
<td>Unit cost of one widget in each region</td>
<td>Calculated, repeating</td>
<td>= Mfg Cost + Delivery Cost</td>
</tr>
<tr>
<td>Revenue</td>
<td>Revenue of each region</td>
<td>Calculated, repeating</td>
<td>= Regional Demand * Price</td>
</tr>
<tr>
<td>Total Profit</td>
<td>Total profit of all regions</td>
<td>Output</td>
<td>= SUM(Profit)</td>
</tr>
</tbody>
</table>

Table 2 - Formula List with the repeating sub-model

3 THE REPEATING SUB-MODEL IMPLEMENTATION

Like the implementation of the simple model shown in [Mireault, 2015], the implementation of the repeating sub-model follows precise rules. These rules have been empirically designed to reduce the possibility of making errors during the initial implementation of the spreadsheet as well as during its maintenance. All definition formulas are identified with a bold font and a proper number format (currency or percentage) is used where applicable.

We will need five worksheets. As described in [Mireault, 2015], we have the Interface sheet that will be used by the spreadsheet’s user. We also need the
Parameters and the Model sheets for all the variables and constants that are outside the repeating sub-model box. We will also use two sheets for the repeating parameters and the repeating variables defined inside the repeating sub-model box. We will name them Parameters-Regions and Regions.

The interface sheet

In the Interface sheet (Figure 4), we put the input variables with their reasonable starting value. We then name the cells containing the input variables. Finally, we prepare the area where we will put the references to the output variables. In the case where some output variables are from the repeating sub-model, we will also reference the repeating entity.

We will come back to complete Output Variables section of the Interface sheet after we have finished with the model.

![Figure 4 - Step 1: Naming the sheets and setting up the Interface](image)

Step 2: The parameters sheets

We define the single value parameters by putting the labels in column A and the values in column B. As illustrated in Figure 5, we name the single value parameters by selecting the labels and the values and clicking on the Create from Selection button of the Formulas ribbon.

![Figure 5 - Step 2: Creating names for single value parameters](image)
The repeating entity is defined in the Parameters-Regions sheet with the column labels we will use to identify its instances. In column A, we first write the name of the repeating entity, Region in our case, and the names of the parameters appearing inside the repeating sub-model box. Then, starting in Column B, we write the values of the entity and of the parameters, as illustrated in Figure 6.

![Figure 6- Step 2: Parameters of the repeating entity](image1)

Finally, we name the multiple value parameters by first selecting the entire rows consisting of the labels all the values and all the blank cells following them, as shown in Figure 7. Naming the whole row like this will allow us to easily add new regions.

![Figure 7- Step 2: Naming the repeating entity parameters](image2)

**Step 3: The Model sheet**

*In the Model sheet, we need to put the definition formulas of all the calculated variables that are outside of the repeating sub-model box. In our case, there are only two variables: TotalDemand and TotalProfit.*

Figure 8 shows the block structure with the defining variables above the line and the defined variable below. It also shows that cell B2 has been named Total Demand. Since Total Profit is calculated using a variable defined in the repeating sub-model, we will defer defining it until we have completed the repeating sub-model.
Implementing a repeating sub-model is done in two phases. The first phase consists of implementing the model for only one instance of the repeating entity by following the usual block structure and variable naming operations. The second phase consists of one copy operation where the model we implemented for the first instance of the entity is copied for all the other instances.

We start by identifying the region corresponding to the first column of the model. As shown in Figure 9, we put the label Region in cell A3 and the formula =Region in B3. It is important to start the model in the same column we used in the Parameters sheet; otherwise the names will not work properly.

Figure 8 - Step 3: The Model sheet

Step 4: The repeating sub-model Sheet

Implementing a repeating sub-model is done in two phases. The first phase consists of implementing the model for only one instance of the repeating entity by following the usual block structure and variable naming operations. The second phase consists of one copy operation where the model we implemented for the first instance of the entity is copied for all the other instances.

We start by identifying the region corresponding to the first column of the model. As shown in Figure 9, we put the label Region in cell A3 and the formula =Region in B3. It is important to start the model in the same column we used in the Parameters sheet; otherwise the names will not work properly.
We then develop the model with the block structure described in [Mireault, 2015] with the exception that we use the label of column A to name the whole row instead of the cell on its right as shown in Figure 10.

Figure 10 – Step 4: Naming the whole row of a variable's definition

Figure 11 – Step 4: The completed model for one region

Figure 12 – Step 4: The Formula View of the completed model
Figure 11 shows the block structure for all the variables of the repeating sub-model. We can see in its corresponding Formula View (Figure 12) the structure of each definition block: the top part contains named references to the variables used in the calculation, and the bottom part is the actual definition formula. By using only the cells directly above, the definition formula is easy to audit.

Once the model for one region is completed we will copy it to the right for the other regions. We start by selecting column B by clicking on its column letter as illustrated in Figure 13. We then drag the copy handle two columns to the right. The final result is shown in Figure 14.

Finally, we return to the Model sheet to implement the definition of Total Profit. Since it uses a repeating variable in its calculation we will implement it with a variation of the block structure.

First, we set up the usual block structure, with the references in the top part, as shown in Figure 15. We also include references to the repeating entity as a visual reference.
Next, we write the formula in cell B10 as the SUM of the whole row above it. As shown in Figure 16, we name the single cell B10, not the whole row 10, but we put the top border on the whole row as a visual indication that cell B10 uses the whole row above it.

Step 5: Finishing the Interface sheet

The final step consists of putting the references to the output variables in the Interface sheet. In column B we write the reference formulas for the entity, Region, and the output variables, Profit, as illustrated in Figure 17.

4 CONCLUSION

We have shown how to model a problem involving a repeating entity in a way that creates a simple and elegant sub-model.

By separating the creative task of building the conceptual model from the mechanical task of implementing the physical model, we expect to reduce logical errors due to the constant interruptions of the brain’s creative activities.

By reducing and by tightly constraining the copy operations, we expect to reduce the number of errors that are due to manipulations.

Further research can demonstrate whether the use of the Structured Modelling and Implement methodology with repeating sub-model does indeed affect the probability of making errors in a spreadsheet.
REFERENCES


Auditing spreadsheets: With or without a tool?

Simone Schalkwijk, Felienne Hermans, Michiel van der Ven, Hans Duits
HU University of Applied Sciences Utrecht and Delft University of Technology
Simone.Schalkwijk@student.hu.nl, f.f.j.hermans@tudelft.nl, michiel.vanderven@hu.nl, hans.duits@hu.nl
Additional research material is available on:
http://figshare.com/authors/Simone_Schalkwijk/706743

ABSTRACT

Spreadsheets are known to be error-prone. Over the last decade, research has been done to determine the causes of the high rate of errors in spreadsheets. This paper examines the added value of a spreadsheet tool (PerfectXL) that visualizes spreadsheet dependencies and determines possible errors in spreadsheets by defining risk areas based on previous work. This paper will firstly discuss the most common mistakes in spreadsheets. Then we will summarize research on spreadsheet tools, focusing on the PerfectXL tool. To determine the perceptions of the usefulness of a spreadsheet tool in general and the PerfectXL tool in particular, we have shown the functionality of PerfectXL to several auditors and have also interviewed them. The results of these interviews indicate that spreadsheet tools support a more effective and efficient audit of spreadsheets: the visualization feature in particular is mentioned by the auditors as being highly supportive for their audit task, whereas the risk feature was deemed of lesser value.

1 INTRODUCTION

A little while after spreadsheets were created in the 1960s, researchers started to examine the high rate of mistakes in spreadsheets [Olson & Nilsen, 1987]. Panko [1998] states that 86% of all spreadsheets contain errors. Eusprig collects ‘horror stories’ caused by errors in spreadsheets [EuSpRIG Horror Stories, 2012]. One of those stories concerns an understatement of six million US dollar of Knox County Trustee’s Office’s cash on hand due to a bad linked spreadsheet. This resulted in an additional audit fee of 12,500 US dollar. This example provides an insight in the possible outcome of an error in a spreadsheet. In this case the auditor did find the error but what happens when the error is not caught? This could cause serious image and financial damage to the auditor. Many tools have been developed to reduce this risk and to provide a more effective and efficient audit. What is the added value of such tools? This paper determines the perceptions of auditors about the added value of spreadsheet tools in financial auditing and provides guidelines for the improvement of spreadsheet tools. In order to get a clear overview of the perceptions of auditors about the usefulness of a spreadsheet tool during the audit, auditors from one of the big four audit firms in the Netherlands were shown functionalities in a spreadsheet tool called PerfectXL. The auditors were questioned by means of semi-structured interviews.

This paper is structured as follows. Section 2 provides a theoretical background on the most common errors in spreadsheets. Section 3 describes the methodology that has been followed. Section 4 describes the interview results. Section 5 states a conclusion. Section 6 provides topics for future research.
2 THEORETICAL BACKGROUND

2.1 Overview of errors

For many years spreadsheets are known to be error-prone [Panko, 1998]. Researchers Powell, Baker and Lawson [2008b, 2009a, 2009b] focus their studies on errors in operational spreadsheets. Firstly they critically overlooked the existing literature on spreadsheet errors. They found that laboratory experiments provide evidence for high error rates measured by cell error rates or percent of spreadsheets with errors. Studies using field audits in general show the same as laboratory experiments but methods and results vary widely. Furthermore they examined errors in operational spreadsheets and the impact of those errors on financial performance. They found that 14 out of the 25 spreadsheets contained errors. In these spreadsheets 381 potential errors were found of which 117 were confirmed as errors by the developers of the spreadsheet. Among these confirmed errors, 47 had no quantitative impact on the results. But among the 70 confirmed errors, the largest error had an impact of 100 million US dollars.

Many scientists have researched the causes for the high error rate in spreadsheets and have defined a categorization of errors [Kreie, Cronin, Pendley, & Renwick, 2000; Panko, 2008; Panko & Aurigemma, 2010]. This categorization by scientists of the most common errors in spreadsheets differs. Powell, Baker and Lawson [2008b] discuss that there is no one single correct categorization of spreadsheet errors. We found the categorization by de Ruijter and Pjoter [2010] most suitable for our research because this categorization is derived from the viewpoint of a controller within an organization. They define seven categories:

1. Reference errors. This category includes errors like wrong references to other spreadsheet cells or incorrect summation of values.
2. Cells containing an incorrect formula according to financial principles. For example an incorrect formula for a discounted cash flow.
3. Logical errors in Excel. This category includes incorrect application of a formula function. For example an IRR (Internal Rate of Return) function is used instead of an XIRR (Internal Rate of Return for a Series of Cash Flows) function.
4. Interface errors. This category contains incorrect or incomplete references to external sources, other spreadsheets or Pivot tables that are not up to date.
5. Input errors. Typing errors and incorrect assumptions are included here.
6. User related errors. This category contains the incorrect use of copied values and formulas (instead of correct references) or incorrect use of filters and sorting.
7. Control environment errors. These kinds of errors are caused by a lack of controls within the spreadsheet. For example formulas that are accidentally overwritten by fixed numbers, unauthorized changes and the use of wrong versions.
2.2 Overview of spreadsheet tools

The usage of spreadsheets is widespread [Hermans, 2012], because they are flexible and easy to use. Hermans [2012] states that half of all spreadsheets that a company has are used as a basis for decision making. Spreadsheets are also very commonly used for financial reporting purposes. Those spreadsheets have to be audited by an auditor. The extent to which a spreadsheet will be audited substantively depends on several factors. For example if the client maintains an effective internal control system to keep the integrity of a spreadsheet at the desired level, this results in lesser substantive testing by the auditor. A spreadsheet representing a material account in the financial statements results typically in more substantive testing.

Auditors should take into consideration that there is an increased likelihood of a misstatement in information provided in the form of spreadsheets. If an error causes a material misstatement and the auditor does not detect this error before certifying the financial statements, the impact on the image and the financial position of the auditor can be huge when this becomes public. The impact of the increased likelihood of misstatements in spreadsheet on the audit of spreadsheet in the context of legislation is outside the scope of this research.

Various organizations have developed spreadsheet tools that support error correction in spreadsheets. A list of spreadsheet auditing tools can be found at Resources for Spreadsheet Analytics, https://sites.google.com/a/usfca.edu/business-analytics/development-management/checking-auditing.

Research on spreadsheet tools is rather limited and conclusions are contradictory. Abraham and Erwig [2007] tested the spreadsheet tool Ucheck. In this research, spreadsheets created by university students were used to examine the effectiveness of Ucheck. These spreadsheets were evaluated by high school teachers. The researchers found that Ucheck does support users in correcting unit errors. Unit errors are categorized as input errors in our taxonomy. Since this research did not examine spreadsheets created by companies, it provides limited evidence that this spreadsheet tool will be useful for auditors when auditing spreadsheets. Powell, Baker and Lawson [2008a] developed an auditing protocol to find errors in operational spreadsheets. They tested this protocol together with spreadsheet auditing tools XL Analyst and Spreadsheet Professional amongst current undergraduate and graduate students in business or engineering and recent alumni of these programs. They found that the auditing software generated a high percentage of false positives and false negatives. However in their believe the use of auditing software is far more effective in identifying errors than unassisted auditing. They also found that auditors developed skills that allowed them to understand the formal structure of a complex spreadsheet developed a sense of where errors were likely to occur. Other spreadsheet auditing tools such as Spreadsheet Detective, Excel Auditor and Operis Analysis Kit were also subjected to research [Anderson, 2004]. The researchers concluded that these spreadsheet tools were very effective in detecting mechanical errors. Mechanical errors are according to Panko and Halverson [1996] errors arising from typing or pointing errors, so in our research we classify them as reference and input errors. In the study of [Anderson, 2004] the spreadsheet tools detected values stored as text in 82% of the cases and incomplete ranges were detected in 55% of the cases. Despite these good rates of error detection, the tools were unable to correctly flag errors.
in logic like the omission of a variable or an operator precedence error. These errors were caught in 18% and 9% of the cases respectively. In our taxonomy these errors are likely the cause of cells containing an incorrect formula according to financial principles or logical errors in Excel.

The spreadsheet tools described previously focused on static analysis of spreadsheets. The spreadsheet tools discussed next provide broader functionalities, including risk analysis and visualization. Hermans [2012] concludes that tools containing risk analysis and visualization functionalities contribute to a more effective and efficient spreadsheet audit. Panko and Aurigemma [2010] find that two kinds of inspection auditing software (Excel Error Check and Spreadsheet Professional), which include these functionalities, were almost useless for correctly flagging natural human errors in spreadsheets. The human inspectors found 54 errors from the total of 97 errors, as opposed to 5 errors that were flagged by the spreadsheet tools.

Microsoft itself came up with an add-in in Microsoft Office Excel 2013, to better comprehend the issues of auditing spreadsheets. This add-in is called Spreadsheet Inquire and offers various improvements compared to earlier versions of Microsoft Office Excel. These improvements are along the lines of the needs specified by Hermans [2012], as described above. O’Beirne [2013] examined this add-in and states that the functionalities of Spreadsheet Inquire are undeveloped compared to current spreadsheet tools. He found that visualization of relations between sheets from complex spreadsheets is not possible. Risk analysis is also not possible in Excel Spreadsheet Inquire. So the improvements that the add-in Spreadsheet Inquire should offer do not seem to be helpful compared to current spreadsheet tools.

Since the previous spreadsheet tools in our opinion do not provide enough support for auditors in auditing spreadsheets, we are looking at a different spreadsheet tool called PerfectXL. Hermans is the founder of Infotron, which developed this spreadsheet tool according to the research she did between 2008 and 2012. This spreadsheet tool should support auditors by providing a risk analysis function and a visualization function. Hermans [2012] determined several situations that proved high risk for causing errors in spreadsheets. Risk analysis should therefore help the auditor to focus on calculations in spreadsheets that have a high risk of containing errors. The risk analysis functionality highlights risky areas in the spreadsheet as is shown in figure 1.
A visualization functionality should also help the auditor to understand relations between different sheets of a spreadsheet. The visualization functionality in PerfectXL is available on a spreadsheet level basis which shows dependencies between sheets. Figure 2 shows the visualization of a workbook. The blocks represent the worksheets within the workbook. The external sources (worksheets from other workbooks in Excel) linked to a sheet are recognizable by the orange colour. Hidden and very hidden sheets are presented as relatively light blue and grey blocks. Thick arrows between the blocks indicate large dependencies.

Figure 1: Risk analysis and refactoring tips as an overview for a whole spreadsheet.

Figure 2: Visualization of a part of a spreadsheet where external sources and hidden sheets are made visible.
The visualization functionality is also available on a sheet level basis (figure 3) whereas the content of the sheet is divided into different categories. Orange cells are labelled as text, yellow cells are labelled as single numbers, blue cells indicate that a formula is used. The purple lined boxes show the range of consistent formulas.

A full overview and explanation of PerfectXL is presented in appendix 1. A free trial of this tool is available online at www.infotron.nl.

3 METHODOLOGY

3.1 Research questions

This research aims to answer the following questions:

R1 Are common errors recognized by auditors?

R2 To what extent does PerfectXL contribute to a more efficient and effective audit?

R3 Which functionalities should PerfectXL possess in order to contribute towards a more efficient and effective audit?

3.2 Research design

This research is designed to determine the perceptions of auditors on the usefulness of PerfectXL. 8 auditors, all working for the same big four audit firm in the Netherlands, participated in this research. The auditors who were selected have a working experience of over four years. The auditors were interviewed in a semi-structured setting. The
interviews were performed over a period of four weeks, with two interviews each week. Each interview was recorded and transcribed verbatim. The results obtained in the analysis of these interviews were adapted in the questions for later interviews. A full overview of questions asked is included in appendix 2. Normally, the interview started with general questions about the length of service at the audit firm and the customers the auditors are working for. Then there were some more specific questions about common mistakes in Excel made by clients. The spreadsheet tool PerfectXL was then demonstrated to the auditors. Finally, the auditors were asked to give their opinion according to positive, missing and negative aspects of the tool.

The interviews with the auditors were semi-structured because this gave the auditors the ability to voice their opinions without being tied in to answering only structured questions. The tenure of work experience was set to ensure that the auditors had obtained enough experience auditing spreadsheets. Since the PerfectXL tool was not ready at the time the research took place, a presentation of the future tool of PerfectXL was given to the auditors to show the capabilities of PerfectXL. The sheets presented the functionalities visualization and risk analysis and gave an impression of the interface of PerfectXL. For an explanation of these functionalities, refer to section 2.2. Each of those sheets was carefully explained to the auditor by the interviewer, who had significant knowledge of the future tool of PerfectXL. A brief overview of this presentation is included in appendix 2. A full overview of this presentation is available online at figshare.

4 RESULTS

4.1 Most common errors

The auditors agree with the occurrence of errors in spreadsheets. The table below shows the scores from the interviews of the most common errors derived from literature.

<table>
<thead>
<tr>
<th>Most common errors</th>
<th>Yes</th>
<th>No</th>
<th>No answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incorrect range of formulas</td>
<td>75%</td>
<td>12.5%</td>
<td>12.5%</td>
</tr>
<tr>
<td>Typing errors</td>
<td>12.5%</td>
<td>12.5%</td>
<td>75%</td>
</tr>
<tr>
<td>External sources not up to date</td>
<td>12.5%</td>
<td>12.5%</td>
<td>62.5%</td>
</tr>
<tr>
<td>Copied formulas and values that lead to errors</td>
<td>50%</td>
<td>37.5%</td>
<td>12.5%</td>
</tr>
<tr>
<td>Incorrect negative numbers</td>
<td>62.5%</td>
<td>25%</td>
<td>12.5%</td>
</tr>
<tr>
<td>Incorrect input</td>
<td>12.5%</td>
<td>12.5%</td>
<td>75%</td>
</tr>
<tr>
<td>Unfixed error message from Excel</td>
<td>12.5%</td>
<td>12.5%</td>
<td>75%</td>
</tr>
</tbody>
</table>

Table 1: Most common errors noted by auditors from the sample

Most of the common errors that are recognized by auditors are incorrect range of formulas, copied formulas and values that lead to errors and incorrect negative numbers. From the interviews, there is no evidence that typing errors, out of date external sources, incorrect input or unfixed error messages from Excel are recognized common errors from an auditor’s perspective. This can be concluded because the auditors did not mention these errors when asked which errors they frequently encountered in spreadsheets prepared by the client. The last one – unfixed error messages from Excel – seems understandable because an error message from Excel is easily recognized by the client. It is therefore
plausible that the client corrects these mistakes before he provides the spreadsheet to the auditor.

4.2 Perceptions of functionalities of PerfectXL

The table below shows the answers of auditors on the basis of the demonstration of the demo tool.

<table>
<thead>
<tr>
<th>Opinion about the tool</th>
<th>Positive</th>
<th>Doubtful</th>
<th>Negative</th>
<th>No answer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Risk analysis</strong></td>
<td>37.5%</td>
<td>50%</td>
<td>12.5%</td>
<td>0%</td>
</tr>
<tr>
<td><strong>Visualization of spreadsheet logic in general</strong></td>
<td>62.5%</td>
<td>12.5%</td>
<td>0%</td>
<td>25%</td>
</tr>
<tr>
<td><strong>Analysis on spreadsheet level</strong></td>
<td>50%</td>
<td>0%</td>
<td>12.5%</td>
<td>37.5%</td>
</tr>
<tr>
<td><strong>Analysis on sheet level</strong></td>
<td>62.5%</td>
<td>0%</td>
<td>0%</td>
<td>37.5%</td>
</tr>
</tbody>
</table>

Table 2: The perceptions of the auditors from the sample about the usefulness of a spreadsheet tool

The attitude of auditors towards the different functionalities of a spreadsheet tool was mixed. Most auditors were favorable towards the visualization. The analysis on sheet level obtained the most favorable results. The auditors found this functionality helpful for quickly checking for the internal consistency of formulas. Some auditors thought that the analysis on spreadsheet level would help them auditing spreadsheets by better understanding the spreadsheet logic. The auditors were doubtful about the risk analysis.

The last questions were about the attitude in general towards a spreadsheet tool. Most auditors found that the tool indicated a direction for the audit. This could be dangerous if an auditor trusts the tool and overlooks further analysis if no risks are indicated. One respondent formulated this as follows:

“What you saw in the visualization with the colors, green, orange and red. That is very nice to have as a guideline. When you see a sheet that is green, then it will be correct. If it is red then it is time to have a look at what precisely is going on.”

A last observation is that two of the auditors specifically mentioned that they were doubtful that the spreadsheet tool would improve the efficiency and effectiveness of the audit. The following quote indicates that concern:

“What I find difficult is that you could lose yourself in such analysis. So the tool provides various issues and you could check every formula but I am doubtful how I could use this efficiently and effectively in my audit.”
5 CONCLUSION

The goal of this research was to determine the perceptions of the usefulness of PerfectXL in auditing spreadsheets. We defined these perceptions by answering three research questions. Most of the errors defined as common by previous researchers were recognized by the auditors. This indicates the importance of an effective audit by using a spreadsheet tool. Furthermore the auditors agreed that the visualization functionality supported a more effective audit. Especially the visualization on a sheet level basis as referred to in Section 2.2 provided high added value, in the opinion of the auditor. Despite these positive opinions from the auditors, some specifically mentioned that they were doubtful about the effectiveness and efficiency of using PerfectXL in their audit of spreadsheets. We do however believe that the majority of the auditors would appreciate using PerfectXL in their audit of spreadsheets.

More specifically, we give the following answers regarding the research questions.

R1 Are common errors recognized by auditors?
Four out of seven scientifically defined common errors are not recognized by the auditors. For the error messages from Excel, there could be an explanation from the auditor’s perspective because these errors are so obvious that they could easily be detected and corrected by the client before providing the spreadsheet to the auditor.

R2 To what extent do spreadsheet tools contribute to a more efficient and effective audit?
The results for this research question are broad because of the diversity of opinions on the spreadsheet tool from the respondents. The results do indicate, however, that some tool functionalities are helpful in the audit. The functionalities that we refer to in this research are visualization and risk analysis.

R3 Which functionalities should spreadsheet tools have in order to contribute to a more efficient and effective audit?
The results clearly indicate that the majority of the auditors had the perception that a visualization is useful in order to perform a more efficient and effective audit. There is doubt about the added value of a risk analysis. The respondents indicate that is not obvious that risk analysis results in a more efficient and effective audit.

Furthermore, this research sheds light on the usefulness of other spreadsheet tools, in addition to PerfectXL. Because the functionalities of PerfectXL are scientifically composed and more spreadsheet tools offer the same functionalities, a similar experiment could also apply to these spreadsheet tools. This research is also unique in examining the needs of auditors in auditing spreadsheets with spreadsheet tools.

Further research needs to be done in order to determine the improvement of efficiency and effectiveness of audits through the use of spreadsheet tools within the audit. The quantitative effect of using a spreadsheet tool could be determined by performing an experiment. This experiment could be done with spreadsheet tool PerfectXL, because this tool provides functionalities needed by auditors, according to Hermans [2012].
research by Aurigemma and Panko [2010] provides a good example for the research design of this experiment. The spreadsheets used to perform this experiment could be randomly chosen from spreadsheets provided by clients. As we know from the research by Aurigemma and Panko [2010], some errors are not correctly flagged by spreadsheet tools. The aim of this experiment should not be to look at correctly flagging errors by PerfectXL but should rather focus on differences in effectiveness and efficiency between groups of auditors who audit a spreadsheet with a tool and without a tool. This experiment thus has to measure the percentage of errors detected and the time it took to audit the spreadsheet. These measures, compared between the groups, would give an indication of the increase in effectiveness and efficiency through the use of a spreadsheet tool when auditing spreadsheets.

ACKNOWLEDGEMENT

The authors thank the HU Utrecht University of Applied Sciences/FAI and Delft University of Technology for their support in setting up this combined student research project.

6 REFERENCES


APPENDIX 1: BRIEF OVERVIEW OF PERFECTXL

The images below are fragments of the presentation that was shown to the auditors to give an overview of the design and functionalities of PerfectXL:

Figure 4: Risk analysis and refactoring tips as an overview for the whole spreadsheet.

The image above is an overview of the risk analysis of PerfectXL. The tool highlights the following risks:

- fixed numbers in formulas;
- unusual ranges which are detected through inconsistency in a column of consistent formulas;
- formulas that contain a large number of references to another sheet;
- multiple functions in one formula;
- many cell references in one formula;
- a long chain of formulas;
- copied formulas;
- references to empty cells;
- error messages from Excel.
Figure 5: Visualisation with an overview of the degree of risks in the different sheets.

The image above represents the structure of a part of sheets in a spreadsheet. The colours reflect the degree of risk in the separate sheets. A grey arrow shows that there is a link between two sheets. The colour grey stands for a reference to a cell from a sheet on the right hand side of the sheet from which the grey arrow originates. The opposite is the case for a purple arrow.

Figure 6: Visualization of a sheet using risk analysis.

This figure shows the risks that come out of the risk analysis. The results are now presented in the sheet. Yellow coloured cells have a low risk of containing an error.
Orange coloured cells have a moderate risk of containing an error. Red coloured cells have a high risk of containing an error.

Figure 7: Visualization of the whole spreadsheet where external sources and hidden sheets are made visible.

The above figure shows the same arrows as in figure 4. The link between external sources and sheets can be seen in this figure. These external sources are recognizable by the orange colour. Hidden and very hidden sheets are presented as relatively light blue and grey blocks.

Figure 8: Visualization of the content of a sheet including specifications of cells.

In the above figure, orange cells are labelled as text, yellow cells are labelled as single numbers, blue cells indicate that a formula is used. The purple lined boxes show the range of consistent formulas.
APPENDIX 2: FIXED QUESTIONS IN THE INTERVIEWS

General questions:

- How long have you been employed at the audit firm?
- What type of clients do you have?

Questions relating to the use of Excel by clients and auditors and the problems they have while using Excel:

- At what type of customers do you come across spreadsheets most often?
- For what specifications do they use spreadsheets?
- How do you audit spreadsheets? Do you use tools for the audit of spreadsheets?  
  Do you miss anything for this?
- What errors do you detect in spreadsheets?
- Do you sometimes have problems with understanding/auditing spreadsheets?
- Which of the following situations do you see in spreadsheets and how often do you see these situations?
  - Fixed numbers in formulas
  - Copied values and formulas
  - Negative numbers
  - Hidden cells
  - Overly long formulas
  - Incorrect ranges

Questions relating to the tool:

- What do you like about the tool?
- What would you like to see more in the tool?
- What do you think is bad/unnecessary in the tool?
Development and Experimentation of a Software Tool for Identifying High Risk Spreadsheets for Auditing

Mahmood H. Shubbak, Simon Thorne
Cardiff Metropolitan University
sthorne@cardiffmet.ac.uk

ABSTRACT

Heavy use of spreadsheets by organisations bears many potential risks such as errors, ambiguity, data loss, duplication, and fraud. In this paper these risks are briefly outlined along with their available mitigation methods such as: documentation, centralisation, auditing, and user training. However, because of the large quantities of spreadsheets used in organisations, applying these methods on all spreadsheets is impossible. This fact is considered as a deficiency in these methods, a gap which is addressed in this paper.

In this paper a new software tool for managing spreadsheets and identifying the risk levels they include is proposed, developed and tested. As an add-in for Microsoft Excel application, 'Risk Calculator' can automatically collect and record spreadsheets properties in an inventory database and assign risk scores based on its importance, use, and complexity. Consequently, auditing processes can be targeted to high risk spreadsheets. Such a method saves time, effort, and money.

1. BACKGROUND

Financial, commercial, industrial, service sector, government, construction, and educational institutions report high levels of spreadsheet use [SERP, 2006]. This high use is no surprise since spreadsheets tend ‘fill the gap’ between IT system capabilities and current business requirements [Chambers & Hamill, 2008]. However, this large scale uncontrolled use of spreadsheets exposes these organisations to significant risks.

1.1. Spreadsheet Risks

Spreadsheet risks can be categorised into: Data Loss; Errors; Ambiguity; Conflicts and Redundancy and Fraud.

1.1.1 Data Loss

When spreadsheets are used for critical decision-making, such as calculating the company’s budget, supporting strategic reports, or negotiating deals with clients, some important questions should be asked; what would be the case if some of these critical sheets were lost? What are the consequences on the organisation’s deals with clients and their confidence? What are the implications on the validity and accuracy of the decisions based on incomplete information?

1.1.2 Errors

According to Panko [1998]:


‘There has long been ample evidence that errors in spreadsheets are pandemic. Spreadsheets, even after careful development, contain errors in 1% or more of all formula cells. In large spreadsheets with thousands of formulas, there will be dozens of undetected errors’

Panko [1996] demonstrated that with even the simple task of calculating the costs of building a wall, 38% of participants from MIS background have made errors. Field studies of errors show that spreadsheets in industry carry a significant number of errors too, some studies reporting up to 90% of the models audited carrying at least one material error.

Errors can be introduced to spreadsheets at any time and in various different forms, they can occur during the manual entry of data, during internal or external linking, while writing formulas, or within copy-paste operations.

Spreadsheets are usually designed and used by employees with no formal training sessions [Baker et al, 2006]. Spreadsheets that support critical decision making are susceptible to the ‘errors risk’. Errors here mean wrong decisions at the operational level, which can affect the efficiency of organisation’s day to day activities. On the other hand, wrong estimations of assets, costs, or taxes can directly affect the strategic level.

1.1.3 Ambiguity

Using Excel without a systematic procedure of development and documentation can lead by time to ambiguity, especially when these files are used by many users. Misunderstanding of the sheet content, graphs, variable names and formulas can lead to handling them incorrectly, either at the data entry level or in generating reports. If these reports are then used by management in decision making, it will entail serious consequences.

1.1.4 Conflicts and Redundancy

Owing to their flexibility, it is likely that users prefer designing new Excel sheets than searching for similar archived ones. This attitude ends up, in many cases, with a large number of redundant sheets. Sometimes conflicts between those sheets exist; making it very difficult for users to find the most accurate information and integrate all these sheets together [Hermans, 2012].

This risk can delay the processing time and result in inaccurate reports. Moreover, these redundant sheets usually occupy high portions of the organisations’ servers, reducing their efficiency and information security as well.

1.1.5 Fraud

The uncontrolled large use of Excel sheets can negatively affect companies’ security, “They can be opening up opportunities for fraud” [Denizon, 2012], critical information can fall into the wrong hands, and large losses can incurred consequently.

In one of the ‘horror stories’, Allied Irish Bank (AIB) reported in March 2002 a large loss of approximately $700 Million in a fraud [Thorne, 2013]. The fraud committed was hidden by spreadsheets that were used to assess risk.
1.2. **Risk Mitigation Methods**

Existing approaches to mitigating spreadsheet risk begin usually with defining the main factors behind those risks. Accordingly, two different sets of solutions emerge in the literature either based on the view that spreadsheet risk arises from technical issues or that spreadsheet risk arises from management practices. Hence the solutions broadly fall into these two categories either reduce risk through technical or managerial solutions.

1.2.1 **Technical solutions**: relate to the software itself in terms of its interface and characteristics, under which the following solutions can be found:

a. Altering spreadsheet environments [Paine, Ensuring Spreadsheet Integrity with Model Master, 2001]


c. Centralisation; either in a common server [Informatica, 2012], or using Cloud Computing [Mell & Grance, 2011] however, this might encounter many risks, that sometimes exceed the spreadsheet risks [Armbrust, et al., 2009].

The main issue with technical solutions is that by significantly altering the spreadsheet environment, the solution is far less likely to be applicable to the artefacts in most organisations. Hence whilst inventing a new environment might be applicable in niche situations, it is unlikely to be universal before taking into consideration the cost and other barriers to adoption.

1.2.2 **Managerial approaches**: related to:

a. Internal policy and strategy [Chambers & Hamill, 2008]

b. Training sessions [Stott, 2013].

c. Best Practice [Colver, 2004]

d. Auditing [Panko, 1998].

Although the underlying process described by [Chambers and Hamill, 2008] may provide a framework for management solutions to spreadsheet risk in organisations, the detail of such plans is highly sensitive to the organisation. As [Colver, 2004] points out, ‘best practice’ in one situation can exacerbate problems in another. Hence the same issue arises with management approaches, a certain set of best practices might apply well in one organisation but are rarely universal.

1.3 The best defence

By far the most effective method of limiting spreadsheet errors and subsequently risk is through auditing and code inspection activities. Research shows this to be the most effective approach to catching and correcting errors, with group auditing catching two thirds of mistakes [Panko, 1998].

However, the spreadsheet problem is so large that it is not feasible to audit every single model in an organisation. Automated spreadsheet auditing tools can streamline and assist in such activities but this software still needs a skilled auditor to make decisions and to infer if mistakes have been made. OPERIS Analysis Kit [OPERIS, 2012], Spreadsheet Detective [Berglas, 2001], Spreadsheet Studio [McDaid, 2011], Rainbow Analyst [Shallcross, 2001], and SpACE [AuditWare, 2013], are examples of such tools. Whilst
these tools are designed to audit one spreadsheet at a time in depth, they do not generally offer a means of assessing the riskiness of a spreadsheet (or group of spreadsheets) to assist in deciding which spreadsheet models to audit.

What follows is an overview of the tool presented in this paper which attempts to answer the problem of sorting spreadsheets into risk categories so that auditing processes can be directed more efficiently. The software uses a number of separate and customisable metrics based on importance, complexity and usage characteristics.

2.0 RISK CALCULATOR

The risk calculator is an add in tool for Microsoft Excel that firstly scans and builds an inventory of spreadsheets on an organisation's server or network, calculates absolute and relative risk scores. The process is as follows:

1. Firstly, the software scans the main characteristics of the available spreadsheets within the organisation. This information is obtained from Excel and through user interaction.

2. Depending on the scan results, a filtering and sorting process will follow; in which some spreadsheets are suggested to be migrated into another software applications that are more suitable, while other spreadsheets will simply be ignored.

3. Finally a risk quantification procedure is calculated. Each spreadsheet is given two different risk scores; the first is an absolute score, while the second is a relative score comparing to other spreadsheets available in the organisation.

After that, the high risk spreadsheets can be identified and appropriate procedures for risk reduction can be instigated. Moreover, a curriculum for training sessions as well as special purpose templates can be designed in accordance with their nature and content based on the findings of the process.

2.1 Risk Indicators

The following section discusses the various risk indicators that are used to build a risk scorecard for the risk calculator application

2.1.1 Spreadsheet Nature

Inside a firm's servers and computers, one can find large numbers of Excel files that have already reached the end of their life cycle, i.e. they already achieved the purpose they have originally been created for. These old files are thus not being used anymore in the company's daily work, neither for maintaining data, nor for processing it. Identifying these files from the initial stages of the risk management policy is important in order to exclude them from any further consideration. This saves time and processing power in the following steps.

On the other hand, the risk level of a spreadsheet is directly related to the nature of its content. Spreadsheets can either be used for maintaining and observing data, or for processing and doing calculations on it. As calculating spreadsheets include data input, calculation, and output processes, it is obviously more susceptible to human errors either in designing the equations, or in the data entry stages [Panko, 1998].

Those two points are the main factors of consideration under the spreadsheet nature, however to complete the image, an additional piece of information regarding the using...
category of the sheet is also required. What we mean by category is the group to which
the main activity of the sheet is related, e.g. whether it is mainly related to employees,
customers, assets, projects, or products and services. Knowing the spreadsheet category
can facilitate filtering and other file management processes.

In order to understand and estimate the risk within a spreadsheet, we considered the
following three main factors;

2.1.2 Importance

Importance in Risk Calculator is measured via three different dimensions; financial,
strategic, and security related.

Financial importance is based on the approximate amount of money included in the
spreadsheet. Since each Excel sheet can contain more than one currency formatted cell,
which are sometimes totals and subtotals of other cells, the approach chosen for money
approximation here is to find the maximum value within all currency formatted cells. The
maximum here can give a more effective indication to the financial importance of the
sheet, avoiding the exaggerated results of the summation operations.

The second consideration is given to the strategic importance of the Excel sheet, whether
it is used for generating reports that support decision making processes or not. Such an
indicator should have a significant share in identifying critical spreadsheets in the firm’s
business.

The third and last importance consideration is given to the safety and security relatedness
of the spreadsheet. Excel sheets are sometimes used for holding some sensitive data, these
sheets however might not have high monetary or direct strategic decision making
relations, but containing such critical information, they definitely need to be handled
more carefully. For this reason, the risk calculator add-in takes this indicator into
consideration too.

2.1.3 Complexity

While the humans’ learning process is accumulative by its nature, human’s errors seems
to be accumulative too. The more you do, the more serious errors you can make.
Consequently, it can be predicted that large spreadsheets with more cells, formulas,
variables and charts, encounter more end-user programming risk. These factors are used
in the Risk Calculator add-in along with some other deeper measurements as spreadsheet
complexity indicators.

Very little research has been conducted on whether the use of VBA in spreadsheet models
is as error prone as actual spreadsheet modelling, i.e. do we find the same level of
mistakes in VBA as in the spreadsheet model itself. However, one can assume that the
inclusion of VBA is likely to make the model more complex and more difficult to audit.
Similarly, nested if statements are usually a source of errors and ambiguity risks
[Hermans, 2014].

The Solver add-in in Microsoft Excel is normally used to solve optimization problems.
Such problems are mainly related to operations research and decision making. Besides its
complexity indication it also shows the strategic importance of the sheet. Therefore, it can
be considered as a good indicator of spreadsheet risk.

2.1.4 Spreadsheet Use
As explained in section 1.1, when the sheet is used and modified by many users, it becomes more susceptible to errors and ambiguity risks. The more users using a spreadsheet, the larger risk it could have, because of the possible dynamic changes on the spreadsheet they can do.

Moreover, being used frequently, even for solely data reading purposes, can indicate that the sheet is important somehow for the firm’s daily operations.

Finally, spreadsheets that are transmitted externally are usually with a special importance. Such transmitting indicates that the sheet is accessed, used, and might be modified by different persons from different organizations, which increases the potential risk associated with it [Chambers & Hamill, 2008].

2.1.5 Summary of Risk Indicators

The following risk indicators are used in the calculation of risk of a spreadsheet. Some of this information is gathered automatically from Excel and others details are obtained through interaction with the user.

1. Spreadsheet Nature
   1. The current status of the spreadsheet; whether it is active (in use), or idle (out of use).
   2. Whether it is just for data maintaining and observing, or it incorporates input-calculations-output.
   3. The general category to which it belongs; does it hold data about people, assets, products, or any other categories.

2. Spreadsheet Importance
   1. The approximate amount of money related to it.
   2. The strategic importance; in terms of its role in reports generating for decision making purposes.
   3. Its security and safety importance.

3. Spreadsheet Complexity
   1. The spreadsheet size; number of used cells.
   2. The number of formulas.
   3. The number of names (variables).
   4. The advance tools used; such as Solver, Macros, If-statements, Nested-If, and Charts.
   5. Whether it contains comments.

4. Spreadsheet Use
   1. How old is it?
   2. How frequently is it used?
   3. When was the last time it was modified?
   4. The number of users or departments using it.
   5. Whether it is for internal use, or is transmitted externally [Chambers & Hamill, 2008].

Users are not asked to provide all of those properties, since it would be counterproductive to further burden an already busy employee. Fortunately, Microsoft Excel stores properties for each file from which the complexity questions can be answered programmatically.

To collect remaining data, users complete a short electronic survey, shown in Figure 3.
In designing this survey, it was taken into consideration that users usually do not prefer a lot of writing in documentation processes [Brace, 2013]. Therefore all the questions were designed as ‘selecting from alternatives’ and ‘check boxes’, except two pieces of information that are related to amount of money and number of users involved, where only numbers are required.

Once the user clicks the submit button, the approximate risk level for the spreadsheet will be calculated using the data entered and the automatically collected data. This data is then sent to a database called the ‘Spreadsheet Properties Database’ (SPD).

By generating this database, queries can be generated by different departments, in various priorities, and for several purposes, e.g. for archiving and making backups, spreadsheets can be sorted by their importance, while for auditing purposes the complexity will be taken into consideration.

Moreover, the SPD provides the flexibility to evaluate different quantification methods and decide for the best among them. For instance if the organisation thinks that the default quantification procedure does not value one aspect of their spreadsheets accordingly, the weighting of this factor can be changed to value or devalue this factor more.

The SPD can also be considered as a good indicator of the actual use of spreadsheets in the organisation’s business, e.g. it is very easy to know the total amount of money related to spreadsheets, by simply summing all the data in the money field. It also can clearly indicate the nature of spreadsheets use, i.e. what level of complexity is present in the programming, how many macros in total are there? by which departments? and in which categories? etc.

2.1.6 Skipping

In this step, the spreadsheet nature measurements will be used for excluding two types of spreadsheets from any further consideration; the out-of-use and the data maintaining spreadsheets. While the first type is suggested to be ignored, the second can be migrated into the databases of the concerned departments. In both cases, the software tool will have no further interaction with them, except holding records of their basic metadata in the properties database.

In practice, ignoring idle spreadsheets can be achieved by assigning negative risk scores to them. In this case, they will not appear among the highest risk sheets, but their risk levels will still be readable.

2.2 Risk Quantification

The suggested quantification method is to calculate a Graded Point Average GPA of the three factors; Importance, Complexity, and Use.

Similar to the grading method in universities, each factor will be given a specific weight along with a relative grade, from which the GPA will be calculated using this formula:

\[
GPA = \frac{\sum (\text{grade} \times \text{weight})}{\sum \text{weights}}
\]

Table 1 shows these factors, their suggested weights and marking criteria. For each factor two different risk scores are calculated;

2.2.1 Absolute Score:
This score is calculated for each spreadsheet based on its own properties, independently from other spreadsheets within the organisation.

To do so, pre-defined thresholds for each measurement are used to assign its risk score, as shown in table 1. Obviously users will be able to calibrate and modify these thresholds via the software tool’s settings in accordance with their actual needs.

The following example illustrates this quantification method:

Depending on the default definitions for the thresholds, the money measurement will be compared with two thresholds; the first one is defined as £1000, while the second threshold is £10,000. If the spreadsheet money is lower than the first threshold its risk score will be minimum (30%), if its value is between the two thresholds, the risk score will be medium (60%), and finally if its money amount is greater than the second threshold, the risk score will be maximum (100%).

Similarly, other measurements will be translated into percentage scores using specific thresholds. Each spreadsheet will have its own absolute risk score depending solely on its properties, regardless other spreadsheets available in the database. So this score does not change when new spreadsheets are added to the database, or when sheets are removed from it.

2.2.2 Relative Score:

On the other hand, the relative score is a dynamic figure that indicates the actual position of the spreadsheet measurement compared to all other spreadsheets in the database. Therefore, it is expected to get changed every time a new spreadsheet is added to the database, or even when any spreadsheet record is updated or deleted.

To find this score, some descriptive statistical functions are used instead of the user-defined thresholds. The chosen functions here are the quartiles;

The quartiles of a sorted set of values are defined as: the three values that divide the whole set into four equal groups [Journet, 1999].
### Table 1 Risk Quantification

<table>
<thead>
<tr>
<th>Factor</th>
<th>Weight</th>
<th>Marking Criteria</th>
<th>Grade**</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Absolute</td>
<td>Relative</td>
</tr>
<tr>
<td>Importance</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safety or Security</td>
<td>5</td>
<td>Yes</td>
<td>100%</td>
</tr>
<tr>
<td>Importance</td>
<td></td>
<td>No</td>
<td>30%</td>
</tr>
<tr>
<td>Decision-Making</td>
<td>5</td>
<td>Yes</td>
<td>100%</td>
</tr>
<tr>
<td>Importance</td>
<td></td>
<td>No</td>
<td>30%</td>
</tr>
<tr>
<td>Money Importance*</td>
<td>5</td>
<td>≥ 10,000</td>
<td>≥ Q3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in between</td>
<td>in between</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt; 1,000</td>
<td>&lt; Q2</td>
</tr>
<tr>
<td>Complexity</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size (# of cells)</td>
<td>2</td>
<td>≥ 1,000</td>
<td>≥ Q3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in between</td>
<td>in between</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt; 100</td>
<td>&lt; Q2</td>
</tr>
<tr>
<td># of Formulas*</td>
<td>3</td>
<td>≥ 100</td>
<td>≥ Q3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in between</td>
<td>in between</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt; 10</td>
<td>&lt; Q2</td>
</tr>
<tr>
<td># of Variables*</td>
<td>1</td>
<td>≥ 10</td>
<td>≥ Q3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in between</td>
<td>in between</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt; 5</td>
<td>&lt; Q2</td>
</tr>
<tr>
<td>Macros</td>
<td>3</td>
<td>Yes</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No</td>
<td>0%</td>
</tr>
<tr>
<td>Solver</td>
<td>2</td>
<td>same as Macros</td>
<td></td>
</tr>
<tr>
<td>IF statements*</td>
<td>1</td>
<td>same as Variables</td>
<td></td>
</tr>
<tr>
<td>Nested-IF</td>
<td>2</td>
<td>same as Macros</td>
<td></td>
</tr>
<tr>
<td>Charts*</td>
<td>1</td>
<td>≥ 5</td>
<td>≥ Q3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in between</td>
<td>in between</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt; 2</td>
<td>&lt; Q2</td>
</tr>
<tr>
<td>Comments</td>
<td></td>
<td>if comments are used, subtract 5% from the final complexity grade</td>
<td>Optional</td>
</tr>
<tr>
<td>Use</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>3</td>
<td>Always</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sometimes</td>
<td>60%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rarely</td>
<td>30%</td>
</tr>
<tr>
<td># of Users</td>
<td>3</td>
<td>same as Charts</td>
<td></td>
</tr>
<tr>
<td>Internal or External</td>
<td>2</td>
<td>External</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal</td>
<td>30%</td>
</tr>
</tbody>
</table>

* Whenever a measurement value is zero, its assigned score will also be zero.
** Idle (non-active) spreadsheets will have negative risk scores.
Up to here, each spreadsheet will have a specific record in the properties database, as well as a specific percentage risk level, from which three categories can be recognised as follows:

- Risk levels of 75% and above will considered as very high risk.
- Between 50% and 75% as high risk.
- Between 25% and 50% as medium risk.
- And finally, the remaining sheets, less than 25%, will be considered as low risk.

Table 2 below shows suggested risk mitigation and reduction plans for each category:

**Table 2 Risk Reduction Policy**

<table>
<thead>
<tr>
<th>Risk Grade</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>75% - 100%</td>
<td>Very High Risk</td>
<td>- Monitoring</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Auditing &amp; Error Checking</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Archiving &amp; Backup</td>
</tr>
<tr>
<td>50% - 75%</td>
<td>High Risk</td>
<td>- Auditing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Archiving</td>
</tr>
<tr>
<td>25% - 50%</td>
<td>Medium Risk</td>
<td>- Archiving</td>
</tr>
<tr>
<td>less than 50%</td>
<td>Low Risk</td>
<td>- Nothing</td>
</tr>
</tbody>
</table>

In the further applications, it is suggested that once a new high risk spreadsheet is created and saved anywhere in the network, an alert will be automatically sent by the software tool to the management and IT departments, so that further auditing and monitoring processes can be taken.

2.3 Summary

A summary of the potential risks and their mitigation methods is shown in Table 3.

**Table 3 The Relation between Risk Mitigation Processes and Potential Risks**

<table>
<thead>
<tr>
<th>Process</th>
<th>Potential Risks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Loss</td>
</tr>
<tr>
<td>Migration &amp; Skipping</td>
<td>✓</td>
</tr>
<tr>
<td>Training Sessions</td>
<td>✓</td>
</tr>
<tr>
<td>Documentation</td>
<td>✓</td>
</tr>
<tr>
<td>Auditing</td>
<td>✓</td>
</tr>
<tr>
<td>Monitoring</td>
<td>✓</td>
</tr>
<tr>
<td>Versions Archiving</td>
<td>✓</td>
</tr>
<tr>
<td>Backups</td>
<td>✓</td>
</tr>
<tr>
<td>Encryption</td>
<td>✓</td>
</tr>
</tbody>
</table>
It can be clearly noticed that developing this software tool can support most of the spreadsheet risk mitigation methods by identifying high risk sheets. Figure 1 shows a flowchart of the suggested spreadsheet risk management policy and the use of the software tool within it.

![Risk Management Policy Flowchart](image)

**Figure 1: Risk Management Policy Flowchart**

2.4 RiskCalculator Interface

Figure 2 shows the developed add-in after being installed and run in Excel.

![Risk Calculator Ribbon as Add-in for Excel](image)

**Figure 2: Risk Calculator Ribbon as Add-in for Excel**

The ‘Risk Calculator’ Ribbon consists of three main groups of functions;

2.4.1 ‘Spreadsheet Risk’ functions

This group gathers the functions that are solely related to each Excel workbook or file, which are: spreadsheet status regarding to the database, performing risk calculation, collecting properties for the active workbook, as well as automatically generating risk reports for the three main measurements; importance, complexity, and use. Moreover, an additional feature of representing the complexity properties graphically on the active workbook is also available via the ‘heat map’ command button.
Status Monitor

The first interaction between users and the add-in is the status monitor, with which they will be able to get a quick indication of the current situation of their spreadsheet in terms of its existence in the database, and to which extent its correlated record is accurate and up-to-date.

Calculate Risk

It collects the spreadsheet properties, calculates its risk levels accordingly, and stores the final outcomes along with the collected properties into a new record in the properties database. Once these processes are successfully done, the corresponding risk report will be generated and shown on the screen, and the spreadsheet status will be automatically updated. If the ‘defaults’ option (from the settings group) was not already chosen, the tool will ask the user to fill a short electronic survey (figure 3) once he clicks ‘calculate risk’.

2.4.2 Risk Report

Besides being automatically shown after a successful calculation process is executed, users can reopen the risk report anytime by clicking the risk report button. Figure 4 shows an example of this report.
The risk report contains general information about the spreadsheet such as file name, author, the date when it was created, last user, the nearest date it was modified on, its general category, and actual status.

In addition it includes general indications for the overall risk levels; both the absolute and relative risk scores, as well as suggested risk reduction plans that basically depend on the risk level.

The third part of this report is the detailed risk indicators for importance, use, and complexity. Users can also access a detailed report for each factor by clicking on their direct links.

Finally, besides reading the report from screen, users can also save it as a PDF file, in order to keep a soft copy of it, send it online as an email attachment, or print it out as a hard copy.

2.4.3 Heat Map

The ‘Heat Map’ function provides users with a graphical representation for the complexity measurements; by colouring cells background with various colours depending on the formulas they contain. Once the ‘heat map’ function is activated, a key for these colours and their corresponding meanings appears on the custom task pane in the left side of Excel’s window.

Moreover, the type of the ‘heat map’ button has not been chosen as a normal command button but rather as a ‘toggle button’, so when user clicks once again on it, the function will be deactivated, undoing all its changes on cells format.

Figure 5 illustrates the function of this button on a testing spreadsheet.
Figure 5: Heat Map Feature

2.5 ‘Spreadsheet Properties Database’ functions

The SPD or inventory functions group is designed mainly to give users the needed control on the specific record of the active spreadsheet in the properties database, such as updating this record with the most recent data, or remove it completely from the database. Moreover, it also enables users to access the entire database with all its records, in order to view their content, or even apply any processing operations on them; such as filtering, sorting, or running any specific queries they might need. Similar to risk reports, users are also able to export the entire database into a PDF file by clicking the ‘PDF Database’ button.

2.5.1 Options

This group can be considered as the control panel for the software tool in all aspects. Users can have a full control of the add-in settings and behaviour through the options available herein. Additionally, this group gives users the needed flexibility for calibrating the risk calculator parameters. The functions provided in this group can be mainly sorted into three types:

General Behaviour Functions; which includes two quickly accessible options:

Firstly, prompting users to calculate or update their spreadsheets risk records every time they save Excel files, and secondly, using the default values when calculating risk levels. Default values are pre-defined values for all the fields that users are usually asked to fill via the properties survey. When users enable the defaults checkbox, the defaults are used to automatically fill these fields without asking users to give any information. In other words, the feature of using defaults can act as a fast alternative for the properties survey. Such methods can be used with spreadsheets of similar use and importance properties, or when these data are unknown or unimportant.

Furthermore, by using defaults, the amount of money related to the active spreadsheet will be automatically calculated either by summing up all the cells that have currency
formatting, or by finding the maximum value among them. This choice can be made by users in the ‘advance settings’.

Obviously, some limitations can appear here due to the reliance on currency formatted cells; as users sometimes do not use the cell’s formatting properties. Testing the add-in for this particular issue, it was found that the importance risk score fell from 53% to 20% when cell formatting was changed from ‘currency’ to ‘general’. Therefore, users are recommended to correctly use the formatting properties for each cell before using the ‘defaults’ option.

When the add-in is installed into Excel for the first time, both checkboxes; the prompting and defaults are unchecked, and the default option for the money field is the maximum value by default.

2.6.1 Advanced Settings

The advanced settings are categorized as follows:

**Defaults Values** – users are able to define the default values to be used instead of the survey when this feature is enabled.

As shown in figure 6, these data are related to spreadsheet nature, use and importance properties. It is very similar to the survey, however, for the ‘money field’ users can choose either the summation or the maximum value among the currency-formatted cells to be used as a default value.

![Figure 6: Advance Settings - Defaults](image)

The ‘Risk Calculator Settings’; which includes the weights of risk factors, the score levels, and the thresholds to be used for calculating absolute risk scores. Figure 7 shows a screenshot of the risk calculator settings window.
Figure 7: Advance Settings - Risk Calculator

Through this screen, users can calibrate the weights of risk factors depending on their actual spreadsheets’ use. For example, in logistics and supply chain management departments, where operations research calculations are usually done for finding optimal production decisions, the use of the Solver add-in is considered more significant. Therefore, users from such departments can easily increase the ‘Solver’ weight as a complexity risk factor via this screen. In this case, spreadsheets that contain Solver operations will have more complexity scores than those which do not.

‘Database Settings’; which contains:

- The time period threshold for a spreadsheet to be considered as ‘out-of-date’.
- The Database security options, such as setting and changing its password, and controlling its opening options; making it open as a webpage instead of an Excel file.
- The option of updating the entire database and recalculating the risk scores for every single record it contains.
- The option of removing all the records of files that are deleted from the hard drive.

Figure 8 shows all these options.
2.7 Testing and Analysis

A sample of the EUSES spreadsheet corpus was used for testing processes. The results of these tests have been analysed and used for calibrating and improving the software tool functions.

Having a brief look into the corpus, it contains 690 functional spreadsheets that are related to financial operations, which have all been used for testing the ‘Risk Calculator Add-in’.

Since these Excel files have not been designed by the researcher, most of the use and importance measurements could not be determined, such as the number of users, frequency of use etc. For this reason the ‘default values’ approach have been used to deal with these measurements. This made the testing process faster, and its results were basically built on the complexity and money measurements.

Table 4 summarizes the test outcomes:

Table 4: Summary of Test Outcomes

<table>
<thead>
<tr>
<th>Number of Excel files</th>
<th>690</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Time needed</td>
<td>38.74 minutes</td>
</tr>
<tr>
<td>Average Time per file</td>
<td>3.4 seconds</td>
</tr>
<tr>
<td>Average Money</td>
<td>$74,944,542.72</td>
</tr>
<tr>
<td>Average Spreadsheet Size (number of cells)</td>
<td>775.3</td>
</tr>
<tr>
<td>Oldest file created in</td>
<td>1995</td>
</tr>
<tr>
<td>----------------------</td>
<td>------</td>
</tr>
<tr>
<td>Newest file created in</td>
<td>2004</td>
</tr>
<tr>
<td>Average Relative Risk Score</td>
<td>34.9%</td>
</tr>
<tr>
<td>Min. Relative Risk Score</td>
<td>26.8%</td>
</tr>
<tr>
<td>Max. Relative Risk Score</td>
<td>69.5%</td>
</tr>
<tr>
<td>Average Absolute Risk Score</td>
<td>30.5%</td>
</tr>
<tr>
<td>Min. Absolute Risk Score</td>
<td>21.3%</td>
</tr>
<tr>
<td>Max. Absolute Risk Score</td>
<td>62.1%</td>
</tr>
</tbody>
</table>

This testing operation had many benefits, it helped in detecting and fixing some bugs in the add-in, it also supported the calibration of the risk factors’ weights, and it opened the door for further capabilities and analysis possibilities.

Moreover, some compatibility issues have been discovered and solved. It gave us some indications to the differences in the software architecture and internal definitions used in the old and new versions of Microsoft Excel.

### 2.8 Further Analysis Capabilities

One of the most interesting outcomes of this research is the wide possibilities that the ‘risk calculator’ add-in along with the ‘properties database’ can provide.

With its valuable numerical data sets, the ‘properties database’ can open the door to further study and analysis of the spreadsheets nature as well as the human use of it. Such analyses can lead to interesting and sometimes unexpected outcomes. It also reveals some hidden relations between factors and aspects that we have never studied before, or were not able to observe.

This subsection introduces examples for such data analysis for the 690 financial Excel sheets that are tested in this stage.

#### 2.8.1 The relation between spreadsheet size and number of formulas:

As shown in the following chart (figure 9) these two factors are directly correlated; as the largest the spreadsheet is, the highest possibility to have formulas it has.

![Figure 9: Cells vs. Formulas Graph](chart.png)
Since this direct correlation between spreadsheet number of cells and number of formulas can be considered as self-evident, it has never been studied in more detail before. However, understanding such relations might lead to a deeper understanding of the actual interaction between users and spreadsheet applications, and thus, it might help in improving the efficiency of these applications and reducing their risks. Based on the results shown in figure 10, a further step towards understanding such relation as well as studying the human factors of using and designing spreadsheets can be taken, as shown in the graph, formulas occupies 25% of spreadsheet’s cells.

2.8.2 The relation between the spreadsheet size and the tools used within it:

Interestingly, the data analysis of this test operation shows the following outcomes for the relation between Excel sheet size and the use of If statements and Charts in it.

![Figure 10: Cells vs. Charts and If-statements Graph](image)

As shown in figure 10, ‘If-statements’ are mostly used within the mid-size spreadsheets, that have between 1,500 and 5,500 cells. On the other hand, charts can be found in the small-size spreadsheets (of 10 to 2,500 cells size). It was also observed that both tools are not usually used in the large-size spreadsheets (of more than 5,500 cells size). Considered as unaddressed research perspective before, such analysis can give a deeper understanding of the actual human use of spreadsheets, which can support the scientific research on human interaction with end-user computing applications, spreadsheets errors’ classification, and complexity taxonomies. It can also draw the attention to the powerfulness of the add-in developed here in collecting and building multidimensional models of several spreadsheet complexity factors, studying their relative trends, keeping records of them, as well as giving users the desired flexibility to tune up their weights in order to form an overall risk score.

2.8.3 The relation between spreadsheets size and amounts of money they contain:

The following graph (figure 11) shows that the large money amounts are usually included in the small-sized spreadsheets (of cells less than 2,500), while no money amounts can be found in the large spreadsheets. Based on these observations, it can clearly be seen the effectiveness of having different risk measurements for both spreadsheet’s complexity and its financial importance. As large money amounts are related to small spreadsheets, the importance score of such sheets will be relatively high, while their complexity scores
might be low. Having two different measurements, both risk indicators can be easily detected, and thus properly dealt with.

Point X in the graph below is a good example. As the size of this spreadsheet is only 360 cells, with no macros, no solver or any if statements involved, its relative complexity score is 33.7% (low compared to the max. 86.7%). On the other hand, containing large money amounts of about $469 Millions, its relative importance score was at the maximum level of 53.3% (this is the maximum importance level because no decision-making or safety importance is considered by default). Consequently, this spreadsheet had a high overall risk score of 50.5%. The powerfulness of the developed add-in can be obviously seen here, since the high risk incorporated with this spreadsheet was detected, despite its low complexity.

Figure 11: Cells vs. Money Graph

3 CONCLUSIONS

End-User Computing Applications can be a nightmare for any organisation, as their uncontrolled use encounters many potential risks. However, a good understanding of these risks, and applying suitable policies for managing them, can convert spreadsheets into reliable tools that aid the organisation in reaching its goals, benefiting from all the advantages they have.

This research has introduced and developed a software tool that automates spreadsheets risk management, by recording their properties in a special database, and assigning risk scores to them based on their importance, use, and complexity. Consequently, further auditing processes can be applied on the high risk sheets only. Such a method saves much time, effort, and money, and can be considered as bridging the gap in the other risk mitigation methods.

4 FUTURE WORK

Further improvements and research fields can be built on this research, such as:

- Centralizing the developed tool into having a single database within the entire organisation, using a Microsoft SharePoint server.
- Developing a mechanism for generating and sending alerts to management once any high risk spreadsheet is saved or modified within organisation’s servers.

- Improving the complexity factors by differentiating between Excel’s built-in functions, and user-made formulas.

- Using the developed add-in for deeper studying of the relations between spreadsheet internal properties and used tools, as well as the human attitudes in designing and using spreadsheets.

REFERENCES & BIBLIOGRAPHY


ABSTRACT
Research on spreadsheet errors is substantial, compelling, and unanimous. It has three simple conclusions. The first is that spreadsheet errors are rare on a per-cell basis, but in large programs, at least one incorrect bottom-line value is very likely to be present. The second is that errors are extremely difficult to detect and correct. The third is that spreadsheet developers and corporations are highly overconfident in the accuracy of their spreadsheets. The disconnect between the first two conclusions and the third appears to be due to the way human cognition works. Most importantly, we are aware of very few of the errors we make. In addition, while we are proudly aware of errors that we fix, we have no idea of how many remain, but like Little Jack Horner we are impressed with our ability to ferret out errors. This paper reviews human cognition processes and shows first that humans cannot be error free no matter how hard they try, and second that our intuition about errors and how we can reduce them is based on appallingly bad knowledge. This paper argues that we should reject any prescription for reducing errors that has not been rigorously proven safe and effective. This paper also argues that our biggest need, based on empirical data, is to do massively more testing than we do now. It suggests that the code inspection methodology developed in software development is likely to apply very well to spreadsheet inspection.

1 INTRODUCTION
Research on spreadsheet errors is substantial, compelling, and unanimous. As we will see later, all empirical studies, without exception, have found errors at frequencies that few would call acceptable. Actually, if we had done no spreadsheet research at all, results from human error research in other fields would have given us all the data we need to treat spreadsheet error risks seriously.

Yet few developers or corporations have responded to the error research by the only means that has proven effective in other human cognitive domains: massive testing. The problem seems to be that the error rate results do not “feel right”. In our experience, we do not make as many errors as the literature says we do, we catch far more errors than the literature says we do, our spreadsheets are rarely materially incorrect, and the steps we take to make our spreadsheets error free are effective. In the words of American comedian Stephen Colbert, the results that suggest otherwise lack the feeling of “truthiness.” This is true despite the fact that every empirical study has found overconfidence in the face of many errors.

At the root of this disconnect between research findings and our error beliefs, this paper argues, are the fundamental ways in which human attention and cognition work. We will begin the discussion by looking at how our cognition presents the world to us. We will see that our cognition edits out anything that is unimportant, including most errors. This distorts our “experience” of errors, and this distorted experience is what drives our intuition about error risks. Furthermore, our cognition makes
us confident that our “constructed reality” is accurate. Pilots learn that when they are flying through a cloud, they need to trust the data from their instruments, not their perceptions. Business professionals arguably need to do the same when they deal with how they should build and test spreadsheets.

If we do accept spreadsheet error research as being correct, then what should we do in response? We will argue that serious error reduction is likely to be predominantly about testing but not testing as we now do it. Testing based on research will be far more expensive. The “bottom line” is that spreadsheet programs are not error-prone. People are error prone. To use spreadsheet programs is not wrong, but using those that are not tested by research-based methods is.

2 THE EVIDENCE

We will not go over the evidence for the prevalence of spreadsheet errors in detail. Tables of results can be found at panko.com, under both human error and spreadsheet error research. Spreadsheet error research is a tiny corner of the human error research field, so much research on spreadsheet errors has focused on seeing if human error research results apply to spreadsheet error. The brief answer is that they apply very well.

2.1 Error Rates during Development

In the 1980s, there was a kind of Grand Unification in human error research. Researchers in different domains found that their error rates were nearly identical for tasks of comparable cognitive complexity. These error rates were reported as base error rates (BERs), which are long-term average error rates quantified as the number of errors per 100 actions.

BERs obviously depend on complexity. For simple but nontrivial cognitive actions such as writing, calculating, and writing program statements, BERs are usually in the range of 1% to 5%. This is found in both experiments and data collection in real organizations. Programming BERs are based on particularly large corpuses of actual software testing in corporations. Table 1 shows four such corpuses totaling around 10,000 code inspections in industry. These data come from the Panko human error website, which also has data on smaller corpuses. The average error rates of the four corpuses ranged from 1.9% to 3.7%.

Table 1: Error Rates in Program Statements and Spreadsheet Cells

<table>
<thead>
<tr>
<th>Study Description</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>More than 6,000 code inspections in industry, per line of code.</td>
<td>1.9% Weller [1993]</td>
</tr>
<tr>
<td>National Software Quality Experiment, per line of code.</td>
<td>2.0% O’Neill [1994]</td>
</tr>
<tr>
<td>2,500 inspections at Cisco Systems, per line of code.</td>
<td>3.2% Cohen [2006]</td>
</tr>
<tr>
<td>AT&amp;T. 2.5 million lines of code over 8 software releases, per line of code.</td>
<td>3.7% Graden and Horsley [1986]</td>
</tr>
<tr>
<td>14 laboratory studies of spreadsheet development, 967 participants working alone on a variety of tasks. Cell error rate</td>
<td>3.9% Panko Spreadsheet Research Website</td>
</tr>
</tbody>
</table>

Based on human error research, we would expect spreadsheet cell error rates to in the range of 1% to 5%. Table 1 summarizes results from 14 laboratories spreadsheet development studies involving 967 individuals working alone on a variety of tasks. These numbers come from the Panko spreadsheet research website. The average across these studies is a cell error rate of 3.9%.
These per-cell error rates are small. However, spreadsheets contain many cells, and the probability of an error increases rapidly when there are many calculations that depend on precedent cells. Figure 1 shows the importance of error compounding over chains of calculations. The top line in the figure is for a spreadsheet with 100 cells in cascades. This might be a single cascade of 100 cells, 25 cascades of 4 cells, 10 cascades of 10 cells, or any other combination. This might be 100 root (non-copied) formulas, although copying is not error-free, especially when the formula changes in “copied” cells in complex ways.

**Figure 1: Cell Error Rates and Probabilities of a Bottom-Line Error**

Spreadsheets with 100 or more cells in cascades are plausible to consider given the size of corporate spreadsheets. Hermans and Murphy-Hill [2015] analyzed spreadsheets obtained during the legal discovery process at Enron. This corpus included 9,120 Enron spreadsheets with formulas. In these spreadsheets, they found 20,277,835 cells with formulas. Other censuses of spreadsheets have also revealed massive numbers of large spreadsheets. For example, McDaid, et al. [2011] found 65,806 spreadsheets on the department servers in two organizations. These spreadsheets averaged over 4,000 formulas. Even with extensive copying, the number of root formulas must have been extremely large in both corpuses.

Figure 1 shows that for cell error rates of 1% to 5%, the probability of a calculation error is overwhelming. In fact, even error rates an order of magnitude lower would give error probabilities that any company would view as unacceptable. A second line gives the probability of a bottom line error in a spreadsheet with a mere 25 cells in cascades. Here too, likely cell error rates give lower but still unacceptable error rates.

Are bottom-line error rates in real spreadsheets consistent with Figure 1? Table 2 shows that the answer is yes. The table shows the results of 85 intensive inspection studies that took several days per spreadsheet. Most were done by commercial spreadsheet “auditing” firms. One was done by a government tax collection unit using a strong methodology that included substantial cell-by-cell inspection. Hicks [1991] used software code inspection methodology with a team of three inspectors to examine a spreadsheet module. Collectively, these studies found errors in 94% of the spreadsheets studied.
Table 2: Errors Found in Operation Spreadsheets during Intensive Inspection

<table>
<thead>
<tr>
<th>Authors</th>
<th>Spreadsheets</th>
<th>Percent with Errors</th>
<th>Cell Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hicks [1995]</td>
<td>1</td>
<td>100%</td>
<td>1.2%</td>
</tr>
<tr>
<td>Coopers and Lybrand [1997]</td>
<td>23</td>
<td>91%</td>
<td>Fagan Code Inspection with a Team of Three</td>
</tr>
<tr>
<td>KPMG [1998]</td>
<td>22</td>
<td>91%</td>
<td>Commercial “Audit”</td>
</tr>
<tr>
<td>Lukasic [1998]</td>
<td>2</td>
<td>100%</td>
<td>2.2%, 2.5%</td>
</tr>
<tr>
<td>Lawrence and Lee [2001]</td>
<td>30</td>
<td>100%</td>
<td>Audit of Spreadsheet Submitted with Taxes in the United Kingdom</td>
</tr>
</tbody>
</table>

Total / Weighted Average 85 94%

Note also that two studies reported cell error rates. These are similar to those expected from human error research and similar to those found in spreadsheet development experiments.

2.2 Errors Detected During Inspection

Given likely error rates in operational spreadsheets, extensive testing will be needed—more extensive than the testing that was done on operational spreadsheets.

Effective testing is likely to be difficult and extremely expensive. Although people are 95% to 99% accurate when they do calculations, write, code, or enter spreadsheet cells, human error research has shown that humans are much worse at finding errors that have occurred.

Data from the Panko human error website on proofreading for spelling shows that detection rates are 81% for simple spelling errors but only 66% for more complex spelling errors. Detection rates plummet for longer words [Healey, 1980] and more complex material [Riefer, 1991].

In software development, error detection rates are only 20% to 40% [Basili and Selby, 1986; Johnson and Tjahjono, 1997; Myers, 1978; Porter, Votta, and Basili, 1995; Porter and Johnson, 1997; Porter, Sly, Toman, and Votta, 1997; Porter and Votta, 1994] when single inspectors examine a code module to look for errors. This is why software code inspection is always done in teams of three to five or more [Fagan, 1976 1986; Gilb and Graham, 1993; Cohen, 2006].

In spreadsheet inspection, nine experiments using over 1,000 participants in total have found average error detection rates of 60%. However, detection rates depend heavily on error complexity. Even simple and obvious errors are not detected with close to 100 percent accuracy. Anderson [2004] published data for different types of error in his study, which was done by experienced spreadsheet developers. Of 14 errors with different ranges of complexity, three had detection rates below 10%, and half had detection rates below 40%. Even the simplest errors were caught by fewer than 60% of the participants, and no participant found all errors.

What type of errors do people make when they build spreadsheets? Panko and Halverson [1996] divided spreadsheet errors into logical (mathematical and domain knowledge) errors, mechanical errors (such as pointing to the wrong cell and typing errors), and omission errors (leaving out a necessary component). In one experiment, Panko and Halverson [2001] Found that 45% of errors that participants made were logic errors, 23% were mechanical errors, and 31% were omission errors. In another study using a different task, Panko and Sprague [1998] found that 44% of the total errors were logic errors, 3% were mechanical errors, and a whopping 53% were omission errors. The large number of omission errors in this study was an anomaly of the task statement wording. The statement
specified that “Both walls will be built by crews of two. Crews will work three eight-hour days to build either type of wall.” Almost all omission errors occurred when the participants omitted the number of members per crew, the number of days per wall, or the number of hours per day. Their memory was overloaded. This type of error is called a lapse in human error research. In an inspection of an operational spreadsheet using the Fagan [1976, 1986] methodology, Hicks [1995] found that most errors in the spreadsheet his team examined were logic errors. Omission errors were also widely seen in spreadsheet studies. In the Brown and Gould [1987] experiment, when omission errors were excluded, 44% of the spreadsheet models had errors. When omission errors were included, this rose to 63%. Omission errors are especially dangerous because their detection rate tends to be very low, especially if the omission error is due to a misinterpretation of the situation [Reason, 1990; Woods 1984].

2.3 Inexperience, Rushing, and Exacerbaters

Rushing, inexperience, code complexity, stress, and other factors can exacerbate error rates. However, as Figure 2 illustrates, reducing exacerbaters does not eliminate errors. There usually is a floor beyond which error reduction is minimal. Under extremely unchallenging conditions, error rates may even increase.

Table 3 shows how errors relate to stress [Swain and Guttman, 1983]. It shows multipliers of “normal” BERs under different conditions. If the BER under normal conditions is 2.1%, and if the multiplier is 2, then a BER of 4.2% is likely. The table shows that error rates actually increase under very low stress.

Table 3: Multipliers of "Normal" BERs by Experience, Stress, and Task Type

<table>
<thead>
<tr>
<th></th>
<th>Very Low Stress</th>
<th>Optimum Stress</th>
<th>Moderately High Stress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step-by-step task</td>
<td>x2</td>
<td>x1</td>
<td>x1</td>
</tr>
<tr>
<td>Dynamic task</td>
<td>x2</td>
<td>x2</td>
<td>x2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step-by-step task</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic task</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step-by-step task  x2  x4  2:1
Dynamic task  x5  x10  2:1

Source: Swain and Guttman [1983], Table 18-1.

Notes:
The nominal human error probability for a task is multiplied by the modifiers. For instance, if the normal human error probability is 1.2% for a dynamic task under optimum stress done by an experienced practitioner, the estimated human error probability to use for novices under these conditions will be 1.2% x 2, or 2.4%.

Novice workers have been doing the task less than six months, while skilled workers have been doing the task more than six months. The task is not done full time during this period, simply to a normal degree.
Step-by-step tasks are guided by formal procedures. Dynamic tasks require diagnosis and decision-making, keeping track of multiple functions, and so forth.

Many believe intuitively that experienced developers make hardly any errors. Table 3 compares error multipliers for inexperienced practitioners with less than six months of experience and those with more. Beyond about six months, error rates do not go down appreciably [Swain and Guttman, 1983]. Note that only as stress increases do inexperienced versus experienced error ratios reach 2:1. It does not go higher. This limited error benefit of experience surprises many. However, an error ratio of about 2:1 for inexperienced and experienced practitioners is commonly seen in other studies comparing error rates for inexperienced and experienced people [Grudin, 1993; Hayes et al., 1985; Ledgard, 1980; Lesgold, et al., 1988; and Reisner 1975]

Experience increases many aspects of performance, although performance growth often plateaus for long periods or permanently [Bereiter and Scardamalia, 1993]. Figure 3 shows this type of performance growth. It also shows error rates falling initially but quickly reaching a floor. Experience, even with diligent analysis and appraisal, does not decrease errors beyond some level.

Figure 3: Experience, Performance, and Error Rates

We have been looking at expertise in spreadsheet development. For spreadsheet error detection, in turn, Galletta et al. [1993] found that spreadsheet experience had no impact on error yield, although higher experience did decrease the time spent in inspection. Studies of error detection listed at the Panko Human Error Website often used the same spreadsheet detection task and methodology. They had approximately the same results regardless of who was doing the inspection.
3 PERCEIVING AND THINKING

We will begin by looking at human cognition. Humans intuitively feel that if we work carefully and check our work carefully, we will make hardly any errors. Although this is not seen in reality, it is a fundamental belief. Where does this belief come from?

Cognition involves two stages: perceiving and thinking. Perceiving encompasses our brain sensing and selecting what we will pay attention to. Thinking is how we use this information to decide what we will do. Both perceiving and thinking, we will see, are subject to extensive processing that allows us to be fast and flexible but never quite perfect. Valiant [2013] has summarized this by saying that the brain’s goal is to be “probably approximately correct.” This is sufficient for most situations in life, but there are situations in which error tolerances are far smaller. In these situations, we need to work very differently, and our intuition often does not see the need to do so.

3.1 Perceiving: Constructed Reality

Our brains do not simply pull data from our senses and draw results on a large screen in our brain for us to look at. Instead, it goes through several processing steps to develop a consistent and cohesive picture from the dizzying flood of sensations entering our bodies.

Figure 4 shows various steps that our brain uses to process the world around us based on vision, drawing primarily from Panko [2013]. Our eyes can only focus a tiny area directly in front of our gaze direction, so our gaze jerks around constantly in quick small jumps called saccades. Our brains govern these saccades, directing our gaze to things that the brain judges we need to see more clearly. It is like having a flashlight with a narrow beam and focusing it quickly in different directions. Our eyes make about 100,000 saccades per day [Mozlin, 2012]. To make the final picture coherent, however, our brains hide all of this complexity from us. We do not notice that most of what we see around us is out of focus. In addition, even if we try to pay attention to our saccades, we cannot.

The next gate is attention. We can only pay attention to a tiny fraction of what is around us, so our brain filters out whatever it judges to be irrelevant. If we close our eyes and are asked questions about what we just saw, we will be able to recall only a few features of our environment [Chabris and Simons 2010]. However, our brain gives us the comforting illusion that we are taking in what is around us in considerable detail.

Fortunately, our brain does not merely filter out; it also enhances. For example, if we are a batter and watch a pitcher in baseball throw a ball at us, it takes our brain quite a while to get information from our eyes. To compensate, the brain adjusts for this. What we “see” in this constructed reality is the ball further along its trajectory, so that we can swing appropriately [Chow, 2013]. The brain’s editing is artful editing. Again, such adjustments are hidden from us.
3.2 Thinking: System 1 and System 2

Constructed reality is not created for our aesthetic enjoyment. It exists to give us the ability to think about the world. It is the “data” we use in our mental computations. Unfortunately, just as constructed reality gives us a vision of the world that is probably approximately correct, while our thinking is enormously fast, it also adds to our limitations.

System 1 Thinking

Kahneman [2011] has noted that we appear to have two different thinking “systems”—System 1 (S1) and System 2 (S2).

System 1 is our default thinking method. Suppose that we are told that revenues are $1,000, that expenses are $800, and that we should calculate income. We will quickly say, “$200,” and that is the correct answer. For obvious reasons, Kahneman [2011] calls S1 thinking “thinking fast.” Thinking fast is not only fast. We are confident in its conclusions. When you thought “$200,” you were not tentative in your answer.

Consider another example of thinking fast [Kahneman 2011]. Suppose that a bat costs a dollar more than a ball, and that the bat and ball together cost $1.10. How much does the ball cost? About half of us immediately give the answer ten cents. We give this answer as quickly as we did in the first task. In addition, we feel equally good about the answer.

System 2 Thinking

However, the correct answer is five cents. Those who realize that five cents is the correct answer probably also had the initial answer ten cents. However, they then turned on System 2 thinking. S2 thinking takes S1 answers as mere suggestions to be tested. If the ball is really ten cents, the bat must be a dollar. A dollar is not a dollar more than ten cents, of course. The correct answer is that bat is a dollar and five cents, and the ball is five cents.

In the bat and ball task, our brain did a disturbing thing. Rather than solving the actual problem, which would involve algebraic thinking, it solved a different problem. Kahneman [2011] notes that this frequently happens. Our brain often solves a simpler problem than the one we wish to solve. It then interpolates the answer to the actual problem. Yet when this happens, our brain gives us no indication that this has happened. More broadly, System 1 thinking gives us no indication of how well founded its conclusion is. As Kahneman has said, it does not exactly lie to us. Rather, giving us an assessment of its conclusion is not part of what it does. However, confidence in System 1 thinking cannot be used as an excuse for believing everything it suggests.

It seems like we should suspect every conclusion to be wrong. We should always use S2 thinking. There are, however, several reasons why we do not and cannot.

- First, if we actually applied S2 thinking all the time to question everything, it would drive us crazy [Gilovich 1991]. It would be an extreme form of obsessive-compulsive disorder.
- Second, we cannot sustain S2 thinking for more than a brief period. It is literally exhausting.
- Third, even when we do turn on S2 thinking, S2 thinking is difficult and certainly does not always give the correct answer. The bat and ball problem is best approached by algebra, and we all expect to make mistakes when we do algebra.

The fourth and most subtle reason we do not always use S2 thinking is that we usually do not see a need to do so. In the revenue-and-expense problem, you were probably very comfortable in your answer. As noted earlier, whenever S1 gives us an answer, we tend to be comfortable about it, whether it is right or wrong. We are seldom prodded to suspect the answer. Sometimes we are of course, but Allwood [1984], who observed problem solving behavior, found that only some of these instances lead to successful problem resolution. Many undetected errors generated no suspicion at all.
Allwood [1984], however, also noticed that participants sometimes simply paused to check their recent calculations. This is how they caught many of the errors that had generated no specific indications or even vague unease. This was also how they caught many of the errors that had generated unease but had not been caught earlier. Kellog [1994] found that when people write, they spend about a third of their time pausing to plan or check their work. Allwood called this activity a “standard check.” We will call it unprompted error checking. Allwood noted that while this type of error checking is good it is far from being a cure-all for errors.

4 IMPROVING SPREADSHEET ACCURACY

In the first section of this paper, we saw that there are substantial, compelling, and unanimous research findings that spreadsheets are at high risk of spreadsheet errors. In the next section, we saw how human cognition, in both perception and thinking, give us an approximate view of the world that works most of the time but that hides information from us, giving us a distorted experience of what errors we make and how good we are at finding errors.

If the research is true, and if we wish to act on empirical findings like a pilot trusting his or her instruments in darkness instead of physical sensations about an airplane’s orientation in the sky, what should we do? Figure 5 may give us some guidance. To create a program or spreadsheet, the creator usually begins with modules that are later integrated into larger units and finally into whole programs or spreadsheets. In software development, module testing is called unit testing. Spreadsheet development experiments create spreadsheet models that are similar in size to software modules. The figure shows that error rates are similar at the end of the module stage (but before extensive unit testing) for spreadsheets and programs.

Figure 5: Error Rates at the Module and Final Stages of Development

![Error Rates Chart]

We saw this similarity earlier. The figure also shows final error rates in delivered programs, based on errors found after delivery. Note that roughly 90% of the errors in a program are removed during development.

We saw earlier that commercial spreadsheet inspections (audits) and testing of similar intensity found cell error rates of 1.2%, 2.2%, and 2.5%. These inspections averaged about one person-week apiece. Powell, Baker, and Lawson [2008] only found a cell error rate of 0.9% when hardcoding, which all other studies treat as a qualitative error, was removed. However, the average time spent per spreadsheet was only 3.25 hours. Clermont, Hanin, and Mittermeier [2000] found a quantitative cell
error rate of only 0.4%. However, it only used static analysis auditing software. Anderson [2004] and Aurigemma and Panko [14] compared errors discovered with this type of software to actual errors in spreadsheets and found that they were not effective in finding errors. Anderson’s study found that the programs he used only caught 27% of the errors he seeded in the spreadsheet used in the experiment. Aurigemma and Panko’s seven testers found far fewer errors made by actual developers (natural errors instead of seeded errors). Aurigemma and Panko, however, did not use the mapping functions available in this software, although the spreadsheets did not have a strong block structure.

Of course, while cell error rates (CERs) at the module and final stage are similar, they should not be. Inspections of final spreadsheets should have undergone such extensive testing that few errors should remain—as in the case of software development. Furthermore, while the spreadsheets developed in experiments have known correct solutions, all errors can be found. In turn, inspections of operational spreadsheets only report CERs based on discovered errors. Given human limitations in discovering errors, the inspection CERs are likely to be larger than the three numbers shown. Unfortunately, none of the inspections of operational spreadsheets had multiple inspectors individually and report their errors. This would have allowed a statistical estimate of remaining errors, so that error detection effectiveness could be assessed.

What is different between software development and spreadsheet development? Initially, programmers attempted to reduce errors by writing structured programs and using languages with such features as strong data typing. While this helped somewhat, the results of improved development practices were modest. Brooks [1987] summarized this modest progress succinctly. He said that there were no silver bullets in software development.

In the end, commercial software developers addressed the problem of errors by focusing primarily on testing. They accepted that errors would occur and would continue to occur in acceptable numbers even when good development practices are used. Over time, they had to keep increasing testing until they reached acceptable final error rates. Jones (1998) looked at 84 projects in 27 organizations. Testing time ranged from 27% to 34% of total development resources. In every case, subjects reported that not enough time was spent testing. Later, Kimberland (2004) reported that Microsoft software development teams spent 40% to 60% of their total working time in testing.

For software, dynamic testing based on well-chosen test cases with oracles (correct answers) is the most common form of testing. This is good when there is simple final behavior. For instance, if the software is designed to copy changed authentication credentials to all authentication servers, it is relatively easy to see if the software does this under various plausible scenarios. In spreadsheets, however, results are derived from chains of calculations. For each calculation, test cases for a dozen or more equivalence classes may have to be undertaken. This would result in a combinatorial explosion in test cases to compute a final value. Furthermore, spreadsheet software does not have native functionality for implementing this.

For the unit testing of new or modified modules, commercial software companies often turn to static analysis, in which the code is examined without running it. There are several informal methods for static inspection, but firms that use static analysis typically use a specific formal method called code inspection. The code inspection methodology was originally developed by Fagan [1976, 1986] and was later developed by others. Today, it is normal to replace the final meeting in which errors are reported, assessed, and accepted or not accepted with an online process to do these things [Cohen, 2006].

Fagan [1976, 1986] grounded code inspection in empirical analysis. A prime tenet was the obligation to report on the broad results of code inspections, at least in the aggregate. Consequently, code inspection has produced data on tens of thousands of commercial code inspections. This resulted in refinements as different approaches were actually proven to produce better results. People who argued that things should be done in a certain way because they were experts often found themselves wrong. Good practice emerged based on merit, not on the authority of the proposer.
Although Fagan [1976 1986] and later developers did not base their work on human error research, the code inspection is deeply consistent with this research.

A code inspection is always done by teams, never by individuals. Teams typically have three to eight members. This is consistent with the fact that humans are only modestly effective at finding errors. Even with team inspection, code inspections typically only find 60% to 80% of all errors in a module[Boehm and Basili 2001, Eick, et al. 1992, Fagan 1986, Hall 1996, Jones 1986, Jones 1998]. The rest must be found in later testing. Based on data provided from code inspections, single-person code inspection would only catch 20% to 40% of all errors. Panko [1999] conducted an experiment applying code inspection methodology to spreadsheet inspection. He found that individuals only caught 63% of all errors in the spreadsheet they examined. In teams of three, the average was 83%. This is a modest percentage increase, but produced large jumps in detection rates for the most difficult-to-find errors. Inspection by individuals would be difficult to justify in spreadsheet developers.

Code inspection always begins with an initial meeting in which the ground rules are laid out, roles are assigned, and, most importantly, the team is taken through the program line by line to ensure that all inspectors understand the purpose of the module, the purpose of each part, and how the code works. During this process, some errors may be noted by the inspectors, but that is not the purpose of the initial meeting. The initial meeting is important because program code is not self-documenting, even when comments are used. (As developers sometimes remark, “That’s why they call it code.”). In their in-depth interviews with spreadsheet developers, both Nardi and Miller [1991] and Hendry and Green [1994] noted that this problem is serious with spreadsheets. While the rectangular structure of worksheets is ideal for some models, it sometimes creates difficult challenges. Difficult challenges tend to lead to complex precedent structures. Joseph [2000] conducted a code inspection experiment on spreadsheets. He confirmed the importance of the initial meeting, especially for complex logic flows.

After the initial meeting, the inspectors work on the code individually. Initially, the only goal was to develop a very good understanding of the code. Over time, it became more common to make this the stage for doing most error discovery and recording. This stage also has been the most studied and analyzed empirically.

Due to limits of human vigilance, individual inspections must be limited to about an hour. Inspection is mentally exhausting, and longer inspection periods produce a sharp drop in error identification. The statement inspection rate also must be limited to about 100 lines of code per hour for best results. Faster inspection rates again give lower error detection yield. Taken together, these limitations mean that code inspection can only be applied to software modules, not entire programs. Give the data on how rapidly error yield decreases over longer periods and correct inspection rates, and given the importance of understanding what one is inspecting, inspecting an entire program makes no sense.

In the review meeting, the team meets again. Initially, this was the stage in which errors were sought by active inspection. It gradually became the time to gather reports, determine if they are legitimate errors, rate the severity of errors, and create a list of errors to be removed. The team does not attempt to recommend fixes for these errors. Unfortunately, the need to have a review meeting often delayed the results of code inspections. Today, this stage is done online using application software designed for this work. In fact, the software now manages all stages.

After the review meeting, the code inspection is documented. Based on this documentation and data collected by the application, patterns for meetings can assessed. In addition, there needs to be error tracking to verify that errors are corrected.

In spreadsheet development, in contrast, all studies since the 1980s have found that detailed inspection is rare. When it does exist, it typically uses single inspectors and covers chunks of code that are very large with fast inspection rates.
We argue, based on human error research and specific software and spreadsheet research, that code inspection should be used to test spreadsheets. Empirical analysis will be needed to develop size and speed metrics for spreadsheet inspection, but we are likely to find that we will need to spend 20% to 40% of our development resources on testing. Doing this will be expensive. Not doing it is likely to be unprofessional.

5. CONCLUSION

In this paper, we saw that there is substantial, compelling, and unanimous evidence that spreadsheet errors are occurring at a rate that corporations should see as unacceptable. Humans have many intuitions about errors, but the way our cognition works means that we are conscious of very few of our errors. Consequently, we tend to believe, among other things, that few spreadsheets have material errors, that we make few errors, that anyone can make only rare errors and catch almost all errors if they just tried harder and spent more time, and that we know how to reduce spreadsheet errors through good practice.

If we accept that our “experiences” regarding errors is impaired, so that we must turn to human error research, software industry experience, and spreadsheet error research, then we need to do far more testing. Testing is not merely one of many error controls. It is likely to be by far the most important error control. When asked if their spreadsheet contains an error, developers in one unpublished study often said something to the effect that they did not see how that would be possible because they spent “so much time” testing it—although they always added that, of course, everyone makes mistakes. When asked if they were spending 20% to 40% of their time doing testing, their typical response was something to the effect that that would be crazy. If spreadsheet development is to be regarded as a true profession, the biggest change is likely to be a practice requirement for very extensive and expensive testing.

What about the many prescriptions that are touted for reducing spreadsheet errors? Many begin with a statement that, “the problem with spreadsheets is X.” X may be having end users do development instead of professionals, a failure to use a program with strong data typing, or many other things. Although proposers know that they are correct because of their “experience,” this knowledge should not be viewed as reliable. In addition, almost all prescriptions are for development practices, and if there are testing prescriptions as part of these practices, they are usually rudimentary. Senders and Moray [1991], in their review of human error research, noted that many error reduction prescriptions have bad side effects. For example, requiring strong typing could put more burden on developers who are already overloaded and making errors as a result. Given limits in human attention and other resources, adding burdens could increase errors much more than it reduces the type of error it was intended to reduce. Given that our experience of errors in unreliable, intuitions about how to reduce errors should not be taken seriously unless they are rigorously tested. As in the case of code inspection, we can draw from human error literature and proven practices in other areas and test whether it is feasible and effective to apply them to spreadsheets. Medicines must be tested to prove that they are safe and effective before they are accepted. We should require the same rigor for spreadsheet development prescriptions.

In software development, many students and professionals have adopted Watts Humphrey’s Personal Software Process (PSP), in which the person keeps detailed information on their software development activities. The goal of PSP is to counteract intuitive ideas about software errors with actual data. The results are typically a shock to the programmer using it. However, the goal is not to shock but to help the programmer develop better estimation skills for time, resources, and the work needed to reduce errors.

When people are confronted with their actual error rates they are typically shocked. For example, one study examined how frequently people hit the wrong pedal when they drive. The project director was worried that the practice would be so rare that measuring it would be extremely difficult. When the team did the study, they found to their amazement that the frequency of hitting the wrong pedal was
about once per hour [Spiegel 2010]. Self-knowledge is the most potent force for navigating through work.

REFERENCES


Hicks, L. (1995). NYNEX, personal communication with the first author via electronic mail.


