Software Engineering Methods in Spreadsheets

First International Workshop

Proceedings

Organised by the TU Delft Spreadsheets Lab

Co-located with EuSpRiG 2014

Delft, The Netherlands, 2 July 2014

Editors: Felienne Hermans, Richard F. Paige, Peter Sestoft

Preface

This volume contains the papers presented at SEMS-14: Workshop on Software Engineering Methods in Spreadsheets held on July 2, 2014 in Delft. The workshop was organised by Felienne Hermans (TU Delft), Richard Paige (University of York) and Peter Sestoft (IT University of Copenhagen) as a result of their observation that significant research was taking place on spreadsheets in software engineering, and that the time was ripe for a collective meeting bringing together people from different communities - e.g., the end-user programming community, modelling community, testing and verification community, etc. The intent was to have an open workshop, with a reviewing process and open-access proceedings, with the emphasis being on discussions and promoting collaboration. The event was organised by TU Delft and the Spreadsheets Team, as part of Eusprig.

The program committee reviewed and selected 15 papers (including short papers, long papers and tools papers) for presentation during the workshop. Each paper received 3 reviews.

The organisers would like to thank TU Delft for their support for organising Eusprig. As well, they would like to acknowledge use of Easychair in organising the program committee activities, and CEUR for publishing the post-workshop proceedings.

2 July, 2014
Delft, York and Copenhagen

Felienne Hermans
Richard F. Paige
Peter Sestoft

Table of Contents

Program Committee

Jácome Cunha	HASLab/INESC TEC & Universidade do Minho
Felienne Hermans	Delft University of Technology
Nicholas Matragkas	University of York
Richard Paige	University of York
Peter Sestoft	IT University of Copenhagen
Leif Singer	University of Victoria
Tijs Van Der Storm	Centrum Wiskunde & Informatica
Arie van Deursen	Delft University of Technology

Additional Reviewers

Fernandes, Joao
Saraiva, João

Tool-supported fault localization in spreadsheets: Limitations of current evaluation practice

Birgit Hofer
Graz University of Technology
8010 Graz, Austria
bhofer@ist.tugraz.at

Dietmar Jannach
TU Dortmund
44221 Dortmund, Germany
dietmar.jannach@udo.edu

Thomas Schmitz
TU Dortmund
44221 Dortmund, Germany
thomas.schmitz@udo.edu

Kostyantyn Shchekotykhin
University Klagenfurt, Austria
kostya@ifit.uni-klu.ac.at

Franz Wotawa
Graz University of Technology
8010 Graz, Austria
wotawa@ist.tugraz.at

ABSTRACT

In recent years, researchers have developed a number of techniques to assist the user in locating a fault within a spreadsheet. The evaluation of these approaches is often based on spreadsheets into which artificial errors are injected. In this position paper, we summarize different shortcomings of these forms of evaluations and sketch possible remedies including the development of a publicly available spreadsheet corpus for benchmarking as well as user and field studies to assess the true value of the proposed techniques.

Categories and Subject Descriptors

H.4.1 [**Information Systems Applications**]: Office Automation—*Spreadsheets*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging aids*

General Terms

Spreadsheets, Debugging, Fault Localization

1. INTRODUCTION

Locating the true causes why a given spreadsheet program does not compute the expected outcomes can be a tedious task. Over the last years, researchers have developed a number of methods supporting the user in the fault localization and correction (debugging) process. The techniques range from the visualization of suspicious cells or regions of the spreadsheet, and the application of known practices from software engineering like spectrum-based fault localization (SFL) or slicing, to declarative and constraint-based reasoning techniques [1, 3, 6, 7, 9, 11, 12, 16].

However, there is a number of challenges common to all these approaches. Unlike other computer science sub-areas, such as natural language processing, information retrieval or automated planning and scheduling, no standard benchmarks exist for spreadsheet debugging methods. The absence of commonly used benchmarks prevents the direct comparison of spreadsheet debugging approaches. Furthermore, fault localization and debugging for spreadsheets require the design of a user-debugger interface. An important question in this context is: what input or interaction can realistically be expected from the user? Finally, the main question to be answered is whether or not automated debugging techniques actually help the developer as discussed in [14] for imperative programs.

In this position paper, we discuss some limitations of the current research practice in the field and outline potential ways to improve the research practice in the future.

2. LACK OF BENCHMARK PROBLEMS

To demonstrate the usefulness of a new debugging technique, we need spreadsheets containing faults. Since no public set of such spreadsheets exists, researchers often create their own suite of benchmark problems, e.g., by applying mutation operators to existing correct spreadsheets [2]. Unfortunately, these problems are only rarely made publicly available. This makes a comparative evaluation of approaches difficult and it is often unclear if the proposed technique is applicable to a wider class of spreadsheets.

In some papers, spreadsheets from the EUSES corpus[1] are used for evaluations. As no information exists about the intended semantics of these spreadsheets, mutations are applied in order to obtain faulty versions of the spreadsheets. The spreadsheets in this corpus are however quite diverse, e.g., with respect to their size or the types of the used formulas. Often only a subset of the documents is used in the evaluations and the selection of the subset is not justified well. Even when the benchmark problems are publicly shared like the ones used in [10], they may have special characteristics that are advantageous for a certain method and, e.g., contain only one single fault or use only certain functions or cell data types.

A corpus of diverse benchmark problems is strongly needed for spreadsheet debugging to make different research approaches better comparable and to be able to identify shortcomings of existing approaches. Such a corpus could be incrementally built by researchers sharing their real-world and artificial benchmark problems. In addition, since it is not always clear if typical spreadsheet mutation operators truly correspond to mistakes developers make, insights and practices from the Information Systems field should be better integrated into our research. This in particular includes the use of spreadsheet construction exercises in laboratory settings that help us identify which kinds of mistakes users make and what their debugging strategies are, see, e.g., [4].

[1] `http://esquared.unl.edu/wikka.php?wakka=EUSESSpreadsheetCorpus`

3. USABILITY AND USER ACCEPTANCE

Spreadsheet debugging research is often based on offline experimental designs, e.g., by measuring how many of the injected faults are successfully located with a given technique, see, e.g., [5]. In some cases, plug-ins to spreadsheet environments are developed like in [1] or [11]. Similar to plug-ins used for other purposes, e.g., spreadsheet testing, the usability of these plug-ins for end users is seldom in the focus of the research. The proposed plug-ins typically require various types of input from the user at different stages of the debugging process. Some of these inputs have to be provided at the beginning of the process and some can be requested by the debugger during fault localization. Typical inputs of a debugger include statements about the correctness of values/formulas in individual cells [10], information about expected values for certain cells [1, 3], specification of multiple test cases [11], etc.

In many cases, it remains unclear, if an average spreadsheet developer will be willing or able to provide these inputs since concepts like test cases do not exist in the spreadsheet paradigm. Therefore, researchers have to ensure that a developer interprets the requests from the debugger correctly and provides appropriate inputs as expected by the debugger. One additional problem in that context is that user inputs, e.g., the test case specifications, are usually considered to be reliable and most existing approaches have no built-in means to deal with errors in the inputs.

Overall, we argue that offline experimental evaluations should be paired with user studies whenever possible as done, e.g., in [8] or [11]. Such studies should help us validate whether our approaches are based on realistic assumptions and are acceptable at least for ambitious users after some training. At the same time, observations of the users' behavior during debugging can be used to learn about their problem solving strategies and to evaluate whether the tool actually helped to find a fault.

Again, insights and practices both from the fields of Information Systems and Human Computer Interaction should be the basis for these forms of experiments.

4. FIELD RESEARCH

In addition to user studies in laboratory environments, research on real spreadsheets as suggested in [15] is required to determine potential differences between the experimental usage of the proposed debugging methods and the everyday use of such tools in companies or institutes. Error rates and types found in practice could differ from what is observed in user studies whose participants in many cases are students. In [13], e.g., a construction exercise with business managers was done to determine error rates. In addition, the user acceptance of fault localization tools could vary strongly because of different expectations of professional users with respect to the utilized tools. To ensure the usability for real users, existing spreadsheets can be examined and questionnaires with users can be made, as done, e.g., in [7].

5. CONCLUSIONS

A number of proposals have been made in the recent literature to assist the user in the process of locating faults in a given spreadsheet. In this position paper, we have identified some limitations of current research practice regarding the comparability and reproducibility of the results. As possible remedies to these shortcomings we advocate the development of a corpus of benchmark problems and the increased adoption of user studies of various types as an evaluation instrument. As experimental settings differ from real-life, we additionally propose to use field studies to obtain insights on how debugging methods are used in companies.

6. REFERENCES

[1] R. Abraham and M. Erwig. GoalDebug: A Spreadsheet Debugger for End Users. In *Proc. ICSE 2007*, pages 251–260, 2007.

[2] R. Abraham and M. Erwig. Mutation Operators for Spreadsheets. *IEEE Trans. on Softw. Eng.*, 35(1):94–108, 2009.

[3] R. Abreu, A. Riboira, and F. Wotawa. Constraint-based debugging of spreadsheets. In *Proc. CibSE'12*, pages 1–14, 2012.

[4] P. S. Brown and J. D. Gould. An Experimental Study of People Creating Spreadsheets. *ACM TOIS*, 5(3):258–272, 1987.

[5] C. Chambers and M. Erwig. Automatic Detection of Dimension Errors in Spreadsheets. *J. Vis. Lang. & Comp.*, 20(4):269–283, 2009.

[6] J. Cunha, J. a. P. Fernandes, H. Ribeiro, and J. a. Saraiva. Towards a catalog of spreadsheet smells. In *Proc. ICCSA'12*, pages 202–216, 2012.

[7] F. Hermans, M. Pinzger, and A. van Deursen. Supporting Professional Spreadsheet Users by Generating Leveled Dataflow Diagrams. In *Proc. ICSE 2011*, pages 451–460, 2011.

[8] F. Hermans, M. Pinzger, and A. van Deursen. Detecting and Visualizing Inter-Worksheet Smells in Spreadsheets. In *ICSE 2012*, pages 441–451, 2012.

[9] F. Hermans, M. Pinzger, and A. van Deursen. Detecting Code Smells in Spreadsheet Formulas. In *Proc. ICSM 2012*, pages 409–418, 2012.

[10] B. Hofer, A. Riboira, F. Wotawa, R. Abreu, and E. Getzner. On the Empirical Evaluation of Fault Localization Techniques for Spreadsheets. In *Proc. FASE 2013*, pages 68–82, 2013.

[11] D. Jannach and T. Schmitz. Model-based diagnosis of spreadsheet programs - A constraint-based debugging approach. *Autom. Softw. Eng.*, to appear, 2014.

[12] D. Jannach, T. Schmitz, B. Hofer, and F. Wotawa. Avoiding, finding and fixing spreadsheet errors - a survey of automated approaches for spreadsheet QA. *Journal of Systems and Software*, to appear, 2014.

[13] F. Karlsson. Using two heads in practice. In *Proc. WEUSE 2008*, pages 43–47, 2008.

[14] C. Parnin and A. Orso. Are Automated Debugging Techniques Actually Helping Programmers? In *Proc. ISSTA 2011*, pages 199–209, 2011.

[15] S. G. Powell, K. R. Baker, and B. Lawson. A critical review of the literature on spreadsheet errors. *Decision Support Systems*, 46(1):128–138, 2008.

[16] J. Reichwein, G. Rothermel, and M. Burnett. Slicing Spreadsheets: An Integrated Methodology for Spreadsheet Testing and Debugging. In *Proc. DSL 1999*, pages 25–38, 1999.

Toward Interactive Spreadsheet Debugging

Dietmar Jannach
TU Dortmund, Germany
dietmar.jannach@tu-dortmund.de

Thomas Schmitz
TU Dortmund, Germany
thomas.schmitz@tu-dortmund.de

Kostyantyn Shchekotykhin
University Klagenfurt, Austria
kostya@ifit.uni-klu.ac.at

ABSTRACT

Spreadsheet applications are often developed in a comparably unstructured process without rigorous quality assurance mechanisms. Faults in spreadsheets are therefore common and finding the true causes of an unexpected calculation outcome can be tedious already for small spreadsheets. The goal of the Exquisite project is to provide spreadsheet developers with better tool support for fault identification. Exquisite is based on an algorithmic debugging approach relying on the principles of Model-Based Diagnosis and is designed as a plug-in to MS Excel. In this paper, we give an overview of the project, outline open challenges, and sketch different approaches for the interactive minimization of the set of fault candidates.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Spreadsheets; D.2.8 [**Software Engineering**]: Testing and Debugging

1. INTRODUCTION

Spreadsheet applications are mostly developed in an unstructured, ad-hoc process without detailed domain analysis, principled design or in-depth testing. As a result, spreadsheets might often be of limited quality and contain faults, which is particularly problematic when they are used as decision making aids. Over the last years, researchers have proposed a number of ways of transferring principles, practices and techniques of software engineering to the spreadsheet domain, including modeling approaches, better test support, refactoring, or techniques for problem visualization, fault localization, and repair [2, 5, 9, 13, 15].

The Exquisite project [12] continues these lines of research and proposes a constraint-based approach for algorithmic spreadsheet debugging. Technically, the main idea is to translate the spreadsheet under investigation as well as user-specified test cases into a Constraint Satisfaction Problem (CSP) and then use Model-Based Diagnosis (MBD) [14] to find the diagnosis candidates. In terms of a CSP, each candidate is a set of constraints that have to be modified to correct a failure. In our previous works, we demonstrated the general feasibility of the approach and presented details of an MS Excel plug-in, which allows the user to interactively specify test cases, run the diagnosis process and then explore the possible candidates identified by our algorithm [12].

Using constraint reasoning and diagnosis approaches for spreadsheet debugging – partially in combination with other techniques – was also considered in [3, 4, 11]. While these techniques showed promising results in helping users to locate faults in spreadsheets, a number of challenges remain. In this paper, we address the question of how the end user can be better supported in situations when many diagnosis candidates are returned by the reasoning engine. We will sketch different interactive candidate discrimination approaches in which the user is queried by the system about the correctness of individual cells' values and formulas.

2. DIAGNOSING SPREADSHEETS

Algorithmic Approach.

In [14], Reiter proposed a domain-independent and logic-based characterization of the MBD problem. A diagnosable system comprises a set of interconnected components Comps, each of which can possibly fail. A system description SD specifies how components behave when they work correctly, i.e., given some inputs, the definitions in SD and Comps determine the expected outputs. In case the expected outputs deviate from what is actually observed, the diagnosis problem consists of identifying a subset of Comps, which, if assumed faulty, explains the observations.

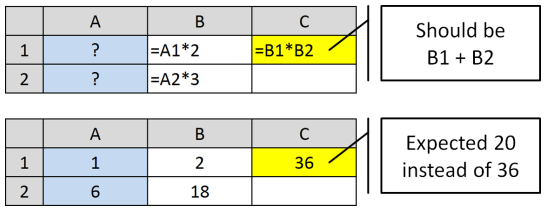


Figure 1: A spreadsheet with an unexpected output

The main idea can be transferred to the spreadsheet domain as follows [12]. In the example shown in Figure 1, the intended formula in cell C2 should be an addition, but the developer made a typo. When testing the spreadsheet with the inputs {A1=1, A2=6} and the expected output {C1=20}, the user notices an unexpected output (36) in $C1$. MBD reasoning now aims to find minimal subsets of the possibly faulty components – in our case the cells with formulas – which can explain the observed discrepancy. A closer investigation of the problem reveals that only two minimal explanations exist in our example if we only allow integer values: "C1 is faulty" and "B2 is faulty". The formula in cell B1 alone cannot be the sole cause of the problem with B2 and C1 being correct as 18 is not a factor of the expected value 20, i.e., there is no solution to the equation

$B1 \cdot 18 = 20, B1 \in \mathbb{N}$. Note that we assume that the constant values in the spreadsheet are correct. However, our approach can be easily extended to deal with erroneous constants.

In [12], we describe a plug-in component for MS Excel, which relies on an enhanced and parallelized version of this diagnostic procedure, additional dependency-based search space pruning, and a technique for fast conflict detection.

We have evaluated our approach in two ways. (A) We analyzed the required running times using a number of spreadsheets in which we injected faults (mutations). The results showed that our method can find the diagnoses for many small- and mid-sized spreadsheets containing about 100 formulas within a few seconds. (B) We conducted a user study in the form of an error detection exercise involving 24 subjects. The results showed that participants who used the EXQUISITE tool were both more effective and efficient than those who only relied on MS Excel's standard fault localization mechanisms. A post-experiment questionnaire indicated that both groups would appreciate better tool support for fault localization in commercial tools [12].

The Problem of Discriminating Between Diagnoses.

While we see our results so far to be promising, an open issue is that the set of diagnosis candidates can be large. Finding the true cause of the error within a larger set of possible explanations can be tedious and, finally, make the approach impractical when a user has to inspect too many alternatives. Since this is a general problem of MBD, the question of how to help the user to better discriminate between the candidates and focus on the most probable ones was in the focus of several works from the early days of MBD research [7]. In the next section, we will propose two possible remedies for this problem in the spreadsheet domain.

3. TOWARD INTERACTIVE DEBUGGING

Early works in MBD research were dealing with fault diagnosis of electrical circuits. In this domain, an engineer can make additional measurements, e.g., of the voltage at certain nodes. These measurements can then help to reduce the set of candidates because some observations may rule out certain explanations for the observed behavior.

Each measurement however induces additional costs or effort from the user. One goal of past research was thus to automatically determine "good" measurement points, i.e., those which help to narrow down the candidate space fast and thus minimize the number of required measurements. In [7], for example, an approach based on information theory was proposed where the possible measurement points were ranked according to the expected information gain.

3.1 Example

Figure 2 exemplifies how additional measurements (inputs by the user) can help us to find the cause of a fault. The example is based on the one from Figure 1. The user has corrected the formula in C1 and added a formula in D1 that should multiply the value of C1 by 10. Again, a typo was made and the observed result in D1 is 30 instead of the expected value of 200 for the input values {A1=1, A2=6}.

Given this unluckily chosen test case, MBD returns four possible single-element candidates {B1}, {B2}, {C1}, and {D1}, i.e., every formula could be the cause. To narrow down this set, we could query the user about the correctness of the intermediate results in the cells B1, B2, and C1.

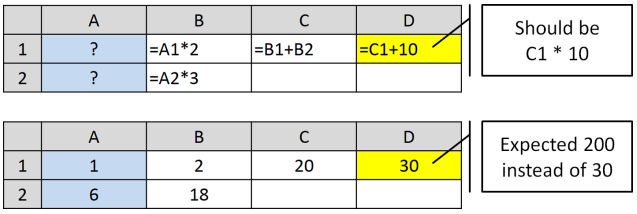


Figure 2: Another faulty spreadsheet

If we ask the user for a correct value of B1, then the user will answer "2". Based on that information, B1 can be ruled out as a diagnosis and the problem must be somewhere else. However, if we had asked the user for the correct value of C1, the user would have answered "20" and we could immediately infer that {D1} remains as the only possible diagnosis.

The question arises how we can automatically determine which questions we should ask to the user. Next, we sketch two possible strategies for interactive querying.

3.2 Querying for Cell Values

The first approach (Algorithm 1) is based on interactively asking the user about the correct values of intermediate cells as done in the example[1].

Algorithm 1: Querying cell values

```
Input: A faulty spreadsheet P, a test case T
S = Diagnoses for P given T
while |S| > 1 do
    foreach intermediate cell c ∈ P not asked so far do
        val = computed value of c given T
        count(c) = 0
        foreach Diagnosis d ∈ S do
            CSP' = CSP of P given T \ Constraints(d)
            if CSP' ∪ {c = val} has a solution then
                inc(count(c))
    Query user for expected value v of the cell c where
        count(c) is minimal
    T = T ∪ {c = v}
    S = Diagnoses for P given T
```

The goal of the algorithm is to minimize the number of required interactions. Therefore, as long as there is more than one diagnosis, we determine which question would help us most to reduce the set of remaining diagnoses. To do so, we check for each possible question (intermediate cell c), how many diagnoses would remain if we knew that the cell value val is correct given the test case $\mathcal{T}$. Since every diagnosis candidate d corresponds to a relaxed version CSP' of the original CSP, where the latter is a translation of the spreadsheet $\mathcal{P}$ and the test case $\mathcal{T}$, we check if CSP' together with the assignment $\{c = val\}$ has a solution. Next, we ask the user for the correct value of the cell for which the smallest number of remaining diagnoses was observed. The user-provided value is then added to the set of values known to be correct for $\mathcal{T}$ and the process is repeated.

To test the approach, we used a number of spreadsheets containing faults from [12], measured how many interactions

[1]Actually, a user-provided range restriction for C1 ($15 < C1 < 25$) would have been sufficient in the example.

Scenario	#C	#F	#D	ØRand	Min
Sales forecast	143	1	89	46.7	11
Hospital form	38	1	15	7.6	5
Revenue calc.	110	3	200	11.8	9
Example Fig. 3	170	1	85	50.3	12

Table 1: Results for querying cell values.

it requires to isolate the single correct diagnosis using Algorithm 1 and compared it to a random measurement strategy.

The results given in Table 1 show that for the tested examples the number of required interactions can be measurably lowered compared to a random strategy. The sales forecast spreadsheet, for example, comprises 143 formulas (#C) and contains 1 fault (#F). Using our approach, only 11 cells (Min) have to be inspected by the user to find the diagnosis explaining a fault within 89 diagnosis candidates (#D). Repeated runs of the randomized strategy lead to more than 45 interactions (ØRand) on average.

As our preliminary evaluation shows, the developed heuristic decreases the number of user interaction required to find the correct diagnosis. In our future work, we will also consider other heuristics. Note however that there are also problem settings in which all possible queries lead to the same reduction of the candidate space. Nonetheless, as the approach shows to be helpful at least in some cases, we plan to explore the following extensions in the future.

- Instead of asking for expected cell values we can ask for the correctness of individual calculated values or for range specifications. This requires less effort by the user but does not guarantee that one single candidate can be isolated.

- Additional test cases can help to rule out some candidates. Thus, we plan to explore techniques for automated test-case generation. As spreadsheets often consist of multiple blocks of calculations which only have few links to other parts of the program, one technique could be to generate test cases for such smaller fragments, which are easier to validate for the user.

3.3 Querying for Formula Correctness

Calculating expected values for intermediate cells can be difficult for users as they have to consider also the cells preceding the one under investigation. Thus, we propose additional strategies in which we ask for the correctness of individual formulas. Answering such queries can in the best case be done by inspecting only one particular formula.

1. We can query the user about the elements of the most probable diagnoses as done in [16], e.g., by limiting the search depth and by estimating fault probabilities.

2. In case of multiple-fault diagnoses, we can ask the user to inspect those formulas first that appear in the most diagnoses. If one cell appears in all diagnoses, it must definitely contain an error.

3. After having queried the user about the correctness of one particular formula, we can search for copy-equivalent formulas and ask the user to confirm the correctness of these formulas.

	...	G	...	L	M
1	...	=IF(A1>5, A13, A25)	...	=IF(F1>5, F13, F25)	=SUM(G1:L1)
2	...	=IF(A2>5, A14, A26)	...	=IF(F2>5, F14, F26)	=SUM(G2:L2)
...	...	...	...	...	...
12	...	=IF(A12>5, A24, A36)	...	=IF(F12>5, F24, F36)	=SUM(G12:L12)
13					=SUM(M1:M12)-1

Figure 3: A small extract of a faulty spreadsheet with structurally identical formulas

The rationale of this last strategy, which we will now discuss in more detail, is that in many real-world spreadsheets, structurally identical formulas exist in neighboring cells, which, for example, perform a row-wise aggregation of cell values. Such repetitive structures are one of the major reasons that the number of diagnosis candidates grows quickly. Thus, when the user has inspected one formula, we can ask him if the given answer also applies to all copy-equivalent formulas, which we can automatically detect.

In the example spreadsheet shown in Figure 3 the user made a mistake in cell M13 entering a minus instead of the intended plus symbol. A basic MBD method would in the worst case and depending on the test cases return every single formula as equally ranked diagnosis candidates. When applying the value-based query strategy of Section 3.2, the user would be asked to give feedback on the values of M1 to M12, which however requires a lot of manual calculations.

With the techniques proposed in this section, the formulas of the spreadsheet would first be ranked based on their fault probability. Let us assume that our heuristics say that users most probably make mistakes when writing IF-statements. In addition, the formula M13 is syntactically more complex as those in M1 to M12 and thus more probable to be faulty.

Based on this ranking, we would, for example, ask the user to inspect the formula of G1 first. Given the feedback that the formula is correct, we can ask the user to check the copy-equivalent formulas of G1 to L12. This task, however, can be very easily done by the user by navigating through these cells and by checking if the formulas properly reflect the intended semantics, i.e., that the formulas were copied correctly. After that, the user is asked to inspect the formula M13 according to the heuristic which is actually the faulty one. In this example, we thus only needed 3 user interactions to find the cause of the error. In a random strategy, the user would have to inspect up to half of the formulas in average depending on the test case. The evaluation of the described techniques is part of our ongoing work.

3.4 User Acceptance Issues

Independent of the chosen strategy, user studies have to be performed to assess which kinds of user input one can realistically expect, e.g., for which problem scenarios the user should be able to provide expected (ranges of) values for intermediate cells. In addition, the spreadsheet inspection exercise conducted in [12] indicates that users follow different fault localization strategies: some users for example start from the inputs whereas others begin at the “result cells”. Any interactive querying strategy should therefore be carefully designed and assessed with real users. As a part of a future work we furthermore plan to develop heuristics to select one of several possible debugging techniques depending on the users problem identification strategy.

4. ADDITIONAL CHALLENGES

Other open issues in the context of MBD-based debugging that we plan to investigate in future work include the following aspects.

Probability-Based Ranking.

Another approach to discriminate between diagnoses is to try to rank the sometimes numerous candidates in a way that those considered to be the most probable ones are listed first. Typically, one would for example list diagnoses candidates of smaller cardinality first, assuming that single faults are more probable than double faults. In addition, we can use fault statistics from the literature for different types of faults to estimate the probability of each diagnosis. In the spreadsheet domain, we could also rely on indicators like formula complexity or other spreadsheet smells [10], the location of the cell within the spreadsheet's overall structure, results from Spectrum-Based Fault Localization [11], or the number of recent changes made to a formula. User studies in the form of spreadsheet construction exercises as done in [6] can help to identify or validate such heuristics.

Problem Encoding and Running Times.

For larger problem instances, the required running times for the diagnosis can exceed what is acceptable for interactive debugging. Faster commercial constraint solvers can alleviate this problem to some extent. However, also automated problem decomposition and dependency analysis methods represent a powerful means to be further explored to reduce the search complexity.

Another open issue is that in works relying on a CSP-encoding of the spreadsheets, e.g., [3, 11] and our work, the calculations are limited to integers, which is caused by the limited floating-point support of free constraint solvers. More research is required in this area, including the incorporation of alternative reasoning approaches like, e.g., linear optimization.

User Interface Design.

Finally, as spreadsheet developers are usually not programmers, the user interface (UI) design plays a central role and suitable UI metaphors and a corresponding non-technical terminology have to be developed. In Exquisite, we tried to leave the user as much as possible within the known MS Excel environment. Certain concepts like "test cases" are, however, not present in modern spreadsheet tools and require some learning effort from the developer. The recent work of [8] indicates that users are willing to spend some extra effort, e.g., in test case specification, to end up with more fault-free spreadsheets.

How the interaction mechanisms actually should be designed to be usable at least by experienced users, is largely open in our view. In previous spreadsheet testing and debugging approaches like [1] or [2], for example, additional input was required by the user. In-depth studies about the usability of these extensions to standard spreadsheet environments are quite rare.

5. SUMMARY

In this paper, we have discussed perspectives of constraint and model-based approaches for algorithmic spreadsheet debugging. Based on our insights obtained so far from the Exquisite project, we have identified a number of open challenges in the domain and outlined approaches for interactive spreadsheet debugging.

Acknowledgements

This work was supported by the EU through the programme "Europäischer Fonds für regionale Entwicklung - Investition in unsere Zukunft" under contract number 300251802.

6. REFERENCES

[1] R. Abraham and M. Erwig. AutoTest: A Tool for Automatic Test Case Generation in Spreadsheets. In *Proceedings VL/HCC 2006*, pages 43–50, 2006.

[2] R. Abraham and M. Erwig. GoalDebug: A Spreadsheet Debugger for End Users. In *Proc. ICSE 2007*, pages 251–260, 2007.

[3] R. Abreu, A. Riboira, and F. Wotawa. Constraint-based Debugging of Spreadsheets. In *Proc. CIbSE 2012*, pages 1–14, 2012.

[4] S. Außerlechner, S. Fruhmann, W. Wieser, B. Hofer, R. Spörk, C. Mühlbacher, and F. Wotawa. The Right Choice Matters! SMT Solving Substantially Improves Model-Based Debugging of Spreadsheets. In *Proc. QSIC 2013*, pages 139–148, 2013.

[5] S. Badame and D. Dig. Refactoring meets Spreadsheet Formulas. In *Proc. ICSM 2012*, pages 399–409, 2012.

[6] P. S. Brown and J. D. Gould. An Experimental Study of People Creating Spreadsheets. *ACM TOIS*, 5(3):258–272, 1987.

[7] J. de Kleer and B. C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[8] F. Hermans. Improving Spreadsheet Test Practices. In *Proc. CASCON 2013*, pages 56–69, 2013.

[9] F. Hermans, M. Pinzger, and A. van Deursen. Supporting Professional Spreadsheet Users by Generating Leveled Dataflow Diagrams. In *ICSE 2011*, pages 451–460, 2011.

[10] F. Hermans, M. Pinzger, and A. van Deursen. Detecting Code Smells in Spreadsheet Formulas. In *Proc. ICSM 2012*, pages 409–418, 2012.

[11] B. Hofer, A. Riboira, F. Wotawa, R. Abreu, and E. Getzner. On the Empirical Evaluation of Fault Localization Techniques for Spreadsheets. In *Proc. FASE 2013*, pages 68–82, 2013.

[12] D. Jannach and T. Schmitz. Model-based diagnosis of spreadsheet programs - A constraint-based debugging approach. *Autom Softw Eng*, to appear, 2014.

[13] D. Jannach, T. Schmitz, B. Hofer, and F. Wotawa. Avoiding, finding and fixing spreadsheet errors - a survey of automated approaches for spreadsheet QA. *Journal of Systems and Software*, to appear, 2014.

[14] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[15] G. Rothermel, L. Li, C. Dupuis, and M. Burnett. What You See Is What You Test: A Methodology for Testing Form-Based Visual programs. In *Proc. ICSE 1998*, pages 198–207, 1998.

[16] K. Shchekotykhin, G. Friedrich, P. Fleiss, and P. Rodler. Interactive ontology debugging: Two query strategies for efficient fault localization. *Journal of Web Semantics*, 12-13:788–103, 2012.

Improving Methodology in Spreadsheet Error Research

Raymond R. Panko
Shidler College of Business
University of Hawai`i
2404 Maile Way
Honolulu, HI 96821
001.808.377.1149
Ray@Panko.com

ABSTRACT
Too much spreadsheet research is unpublishable in high-quality journals due to poor methodology. This is especially a problem for computer science researchers, who often are untrained in behavioral research methodology. This position paper reflects the author's experiences in reviewing submissions to information systems and computer science journals.[1]

Categories and Subject Descriptors
K.8.1: Spreadsheets. D.2.5 Testing and Debugging.

General Terms
Experimentation, Verification.

Keywords
Methodology. Spreadsheet Experiments, Experiments, Inspection. Sampling, Statistics

1. INTRODUCTION

For a number of years, computer science journal editors have taken to sending me articles to review that involve experimental and other methodology. It is frustrating to review these studies because they often show a weak understanding of methodology. Fatal methodological errors are too common, and errors that hobble the use of results are even more frequent. In spreadsheet error research, methodological issues have been particularly common in papers by computer scientists. Based on my experience, this paper presents some prescriptions for improving spreadsheet error research. We will look at issues in inspections (audits) of operational spreadsheets, spreadsheet development experiments, and spreadsheet inspection experiments.

2. INSPECTIONS (AUDITS) OF OPERATIONAL SPREADSHEETS

Several studies have inspected corpuses of operational spreadsheets to look for errors. Many studies call this auditing, but auditing is a sample-driven statistical analysis method for developing an option about quality in development. Audits are not comprehensive error detection tools.

2.1 Respect Human Error Research

Inspection methodologies often fail to reflect the fact that software and spreadsheet error rates are similar. Consequently, spreadsheet methodologies tend to ignore the rather vast literature on code inspection. By code inspection standards, most spreadsheet inspection methodologies do look like mere audits. They lack the required initial understanding of the spreadsheet, are undertaken on whole spreadsheets instead of modules, use single inspectors, and so forth.

2.2 Don't Trust. Verify.

Spreadsheet inspection methodologies are rarely verified. Instead, they tend to be refined until the researchers "feel good" about them. To verify the effectiveness of a methodology, it is important to have multiple inspectors independently use the same methodology to inspect the same spreadsheets. Comparing errors from multiple inspectors can indicate relative effectiveness in finding different types of errors. If the methodology is strong, cross-analysis can even give an estimate of errors remaining.

2.3 Report Time Spent

Time spent in testing is important in assessing human error research. It is important to reveal inspection rates for individual spreadsheets—both time in total and time as a percentage of size expressed in multiple ways, such as all cells, all formula cells, unique formulas, and so forth. If a spreadsheet inspection method has multiple phases, time in each phase should be reported.

2.4 Understanding the Spreadsheet First

Spreadsheets are not self-documenting. It is important for inspectors to be given a thorough explanation of the spreadsheet's detailed logic before they begin testing.

2.5 Report Error Seriousness

The seriousness of errors—at least the most serious error found—should be assessed. Seriousness should be reported by size of each error on monetary or other scales, percentage size of the error relative to the size of the correct value, seriousness of the error in its context, and risk created for the organization. Context must be understood well. In annual budgeting, small errors can be very damaging, while in major one-off projects such as the purchasing of another company, errors would have to be large compared to the results variance caused by uncertainties in input numbers.

3. DEVELOPMENT EXPERIMENTS

In development experiments, participants create spreadsheet models based on requirements in a word problem. To date, we have done well in estimating cell error rate ranges during development. However, there is much more we need to do.

3.1 Use New Tasks

Spreadsheet development experiments have only used a few tasks. We need to do development experiments with more tasks to be confident about typical cell error rates. The widely used Wall and Galumpke tasks have different error patterns. We need to try new tasks to see if new patterns emerge. The Wall task is especially problematic because it was designed to be extremely simple and almost free of domain knowledge requirements. Participants make very few errors on the Wall task.

3.2 Have Adequate Task Length

Errors are rare in spreadsheet development. Tasks need to be relatively long or there will be too few errors to analyze. One way to address this is to have subjects do multiple tasks in a balanced design and to analyze errors in the total multitask sample.

3.3 Go Beyond Student Samples

We also need to do studies on people with different levels of experience in spreadsheet development to ensure that spreadsheet research does not suffer from being the science of sophomores.

3.4 Test Prescriptions for Safety and Effectiveness

We need to move beyond simply claiming that certain prescriptions (such as have a separate assumptions section) and certain tools are good ideas. We must test them to see if they really are "safe and effective." We cannot just build tools and make claims about why they will save the world. Prove it.

3.5 Go All the Way to Error Reduction

Showing that users like it or showing that a tool can help point to earlier cells is not enough. Does it reduce errors? If not, who cares?

3.6 Use Ample Sample Sizes

Sample sizes must be large—at least around 30 to 50 participants per condition. Otherwise, statistical analysis is unreliable. The minimum number should be determined empirically, by a power test.

3.7 Avoid Friends and Family Samples

We also need clean samples. Mixing highly experienced professionals with rank novices in the sample requires far larger samples for statistical validity.

3.8 Do Rigorous Random Assignment to Conditions

Doing rigorous random assignment to the control and treatment groups is mandatory and critical. This must be done on the basis of individuals. We cannot assign whole class sections to different treatments. Nor can we place earlier arrivers in one condition and later arrivers in another condition.

3.9 Use Nonparametric Statistics

It is important to use nonparametric statistics because errors do not follow the normal distribution even roughly. Transforming data so that they are pseudonormal and then applying traditional parametric statistics is not acceptable today.

3.10 Be Generous in Presenting Statistical Results

When giving results, do not just give bare minimum result numbers like means, medians, and standard deviations. Show the full results matrix generated by statistical analysis programs. Also, in comparisons, give overall numerical differences. Do not just say that a difference was statistically significant without giving the numerical differences or correlations.

4. INSPECTION EXPERIMENTS

Inspection experiments should follow the advice in both previous sections. It is wise to avoid seeded errors and go with data from actual development experiments. (The author has such a corpus.)

4.1 Higher Error Rates

One good thing is that human error detection rates are worse than error commission rates, so sample can be a little smaller and still generate enough errors. However, statistical analysis is misleading with less than about 30 subjects per group and rigorous subject randomization.

4.2 Test for Safety and Effectiveness

Again, we need to go beyond simply measuring error detection rates and move to testing alternative methods for finding errors. If we test only two methods—such as doing nothing and using a particular method, then we double the required sample size and must be extremely careful about random treatment assignment. Effects size is also critical in selecting sample sizes.

5. CONCLUSION

We need to stop touting untested prescriptions and tools if we are to put our field on a scientific footing. We must scrutinize prescriptions for safety and effectiveness, and we must do so with exemplary methodology. We also should be balanced in our presentation of results. Everything has strengths and weaknesses. Our results should be honest about weaknesses. Obscuring methodology is a professional sin.

Spreadsheets are Models Too [Position Statement]

Richard F. Paige, Dimitrios S. Kolovos and Nicholas Matragkas
Dept. of Computer Science
University of York, UK
[richard.paige, dimitris.kolovos, nicholas.matragkas]@york.ac.uk

ABSTRACT

Spreadsheets are among the most widely used tools in systems engineering, especially for documenting system requirements and tests, and for supporting tasks like impact analysis, traceability management and project planning. We argue for the treatment of spreadsheets as models, in order to support the systems engineering lifecycle, and to provide a suitable migration path for organisations to follow in maturing their use of modelling techniques.

1. INTRODUCTION

In Model-Driven Engineering (MDE) approaches to systems engineering, many different languages are used (e.g., UML, SysML, domain-specific languages). Usually such languages are designed and implemented by MDE specialists, who use metamodelling infrastructure (e.g., EMF/Ecore[1]) to define the abstract syntax of such languages, and thereafter exploit the infrastructure for the purposes of automation – for example, generating code or text, version control, validation, etc. Once languages with metamodels have been provided, automated model management tools and techniques can be used for systematically manipulating and modifying models across the engineering lifecycle. In particular, tools such as Obeo Designer[2], Epsilon [3], or ATL [1] can be applied to support different engineering tasks.

Systems engineering is expensive and complex, often involves multiple engineering disciplines (e.g., in avionics or aerospace, it can involve software, mechanical, materials and power engineering), and substantial communications overhead between skilled personnel with different vocabularies, practices and tools. Arguably, MDE as it is currently practiced (and supported by tools) is insufficient for supporting the full systems engineering lifecycle. In particular, it can very easily fall short in the early stages, when requirements are still being elicited. Often, early requirements are ambiguous and need to be described in unconstrained natural language: using a domain-specific language may place too many constraints (both conceptual or structural) on specification. As well, requirements often emerge from previous developments, and these requirements may have been specified in non-MDE languages. Combine this with the gradual increase in MDE skills, there is substantial benefit to be able to *interface* MDE languages and tools with non-MDE languages and tools.

In this position paper, we argue for interfacing spreadsheets with MDE languages and tools. We provide motivation for doing this, and briefly touch on some the important technical challenges of, and alternatives for doing so.

2. MOTIVATION

There have been a number of contributions made related to integrating spreadsheets into an engineering process. Much of this work focuses on using software engineering practices to improve the quality of spreadsheets. This includes work on bad smell detection and visualisation in spreadsheets [6], and other analytic approaches that exploit assertions to identify formula errors [8], or that provide testing techniques for spreadsheets [7]. Constructive approaches such as [4, 2] focus on generating high quality spreadsheets using transformation approaches. None of this research has taken the perspective of treating spreadsheets as models.

We have hinted at a number of motivations for treating spreadsheets as models, and for supporting the use of model management operations (such as model transformations) on spreadsheets. We briefly summarise key motivations.

- *Early stages of engineering.* MDE operates most efficiently on well-defined languages (that do not change frequently, or at least, not in significant ways) and models with limited uncertainty. In the early stages of requirements engineering, the concepts of interest in our models may change frequently; they may be imprecisely defined; and the languages that we use to express these concepts may need to evolve. MDE techniques may not be the most useful or appropriate in early stages. Natural language with some restrictions is widely used for early requirements engineering, as are tables of natural language requirements. These can easily be expressed using spreadsheets, which also enable traceability and (in later stages) requirements coverage analysis. Being able to treat spreadsheets

[1] http://www.eclipse.org/emf
[2] http://www.obeodesigner.com/download

as models thus enables defining bridges between early stages of systems engineering, and later stages, where more precise languages are needed.

- *Support for legacy models.* Industry uses spreadsheets, and many large organisations have legacy spreadsheets that can play critical roles, such as in project configuration and monitoring/measurement, requirements capture for product lines, etc. Being able to use such legacy spreadsheets as-is with new engineering processes, practices and tools makes it easier to change processes and practices while reducing risk of bad effects on the bottom line.

- *Tabular problems need tabular solutions.* Some modelling problems are inherently tabular in nature, and benefit from being able to specify data (models) in columns and rows (with constraints amongst them) without requiring relational solutions. Specification of control laws, or parameters used to configure product lines, simple requirements capture, and test suite specification are all problems that lend themselves to tabular specifications, where spreadsheets can conceivably provide support. Providing MDE support for such idioms allows engineers who need to use such concepts to benefit from automated processing support.

- *Supporting existing skillsets.* Not every organisation has, or can quickly acquire, expertise in MDE and model management. Most organisations do have expertise and skills with spreadsheets. Providing means for organisations to transition gradually to use of MDE and model management, and allowing those organisations to maximise the use of their current skillset, could reduce the risks associated with adopting MDE.

- *Catching repeated errors.* Substantial research has been carried out in MDE in terms of automated support for identifying and repairing repeated errors in modelling and model management. For example, updating models or evolving models after changes in a modelling language are problems for which good automated or semi-automated solutions exist. These are problems with spreadsheets as well (e.g., bad smell detection). By interfacing spreadsheets with MDE, it may be that spreadsheet users can exploit MDE solutions.

3. MECHANISMS

There are several plausible ways to interface spreadsheets and MDE.

- Build *injectors* which generate models (with metamodels) from spreadsheets, thus allowing MDE languages and tools to be applied to spreadsheets indirectly. Additionally, extractors from models to spreadsheets may also be needed in order to return results to a form amenable to processing by spreadsheet tools. In both the injection and extraction, *specification blow-up* may be an issue (i.e., encoding or decoding spreadsheets as or from models may lead to less than optimal spreadsheet or model sizes or structures).

- Provide equivalents of MDE and model management operations on spreadsheets, e.g., update-in-place transformations, validation/constraint checking, transformations, text generation. These would need to be encoded using any scripting languages provided by a spreadsheet tool. For example, for Google Spreadsheets, these operations might be encoded using the Spreadsheet Service[3]. However, such encodings would need to be reimplemented for each spreadsheet tool.

- Provide spreadsheet drivers for model management tools, so that these tools can directly manipulate spreadsheets like any other form of models. This is the approach we have taken in Epsilon [5]. A driver must be implemented for each spreadsheet tool - though some abstraction is possible (specifically, a spreadsheet interface is provided that needs to be implemented for each spreadsheet tool). Arguably, implementing an interface for querying and changing spreadsheets via an API is less expensive than implementing model management operations for each spreadsheet tool.

4. CONCLUSIONS

Spreadsheets are models: a less constrained and less expressive form of model than those permitted by full-blown MDE languages and tools. By treating spreadsheets as models, we can provide ways to bootstrap the MDE process, to enable automated and powerful tool support for legacy models, and a way to maximise use of current skillsets while personnel are educated in using MDE and model management techniques. Arguably, MDE and model management tools should support more model/data representation formats and techniques like spreadsheets, which allow more flexible and less constrained styles of specification and design.

5. REFERENCES

[1] Atlas Transformation Language, official web-site. http://www.sciences.univ-nantes.fr/lina/atl/.

[2] J. Cunha, J. P. Fernandes, H. Ribeiro, and J. Saraiva. MDSheet: A framework for model-driven spreadsheet engineering. In *Proc. ICSE*, 2012.

[3] Dimitrios S. Kolovos, Louis M. Rose, Antonio Garcia Dominguez and Richard F. Paige. *The Epsilon Book*. 2013. http://www.eclipse.org/epsilon/doc/book/.

[4] G. Engels and M. Erwig. Classsheets: automatic generation of spreadsheet applications from object-oriented specifications. In *Proc. ASE'05*, ASE '05. ACM, 2005.

[5] M. Francis, D. S. Kolovos, N. Matragkas, and R. F. Paige. Adding spreadsheets to the MDE toolkit. In *Proc. MoDELS*. LNCS 8107, Springer-Verlag, 2013.

[6] F. Hermans, M. Pinzger, and A. van Deursen. Detecting and visualizing inter-worksheet smells in spreadsheets. In *ICSE*, pages 441–451. IEEE, 2012.

[7] G. Rothermel, M. Burnett, L. Li, C. Dupuis, and A. Sheretov. A methodology for testing spreadsheets. *ACM Trans. Softw. Eng. Methodol.*, 10(1):110–147, Jan. 2001.

[8] J. Sajaniemi. Modeling spreadsheet audit: A rigorous approach to automatic visualization. *Journal of Visual Languages & Computing*, 11(1):49 – 82, 2000.

[3] `https://developers.google.com/apps-script/reference/spreadsheet/`

On the Usage of Dependency-based Models for Spreadsheet Debugging

Birgit Hofer
Graz University of Technology
Inffeldgasse 16b/II
8010 Graz, Austria
bhofer@ist.tugraz.at

Franz Wotawa
Graz University of Technology
Inffeldgasse 16b/II
8010 Graz, Austria
wotawa@ist.tugraz.at

ABSTRACT

Locating faults in spreadsheets can be difficult. Therefore, tools supporting the localization of faults are needed. Model-based software debugging (MBSD) is a promising fault localization technique. This paper presents a novel dependency-based model that can be used in MBSD. This model allows improvements of the diagnostic accuracy while keeping the computation times short. In an empirical evaluation, we show that dependency-based models of spreadsheets whose value-based models are often not solvable in an acceptable amount of time can be solved in less than one second. Furthermore, we show that the amount of diagnoses obtained through dependency-based models is reduced by 15 % on average when using the novel model instead of the original dependency-based model. The short computation time and the improved diagnostic accuracy enable the usage of model-based debugging for spreadsheets in practice.

Categories and Subject Descriptors

H.4.1 [**Information Systems Applications**]: Office Automation—*Spreadsheets*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging aids*

Keywords

Spreadsheets, Debugging, Model-based Fault Localization

1. INTRODUCTION

Even for small spreadsheets, the localization of faults can be time consuming and frustrating. Thus, approaches supporting fault localization in spreadsheets are needed. Some fault localization techniques developed for the software engineering discipline have been adapted to the spreadsheet domain, for example [1, 9, 10, 8]. Model-based software debugging (MBSD) [16, 18, 13] is one of these techniques. So far, researchers have only focused on methods that use value-based models [3, 4, 12, 11]. Value-based models compute a small set of possible explanations (i.e., diagnoses) for an observed misbehavior. This small set of diagnoses is helpful for users when debugging. Unfortunately, value-based models have high computation times and they do not scale: the underlying solving mechanisms have problems when dealing with variables with large domains and real numbers. In an empirical evaluation, Außerlechner et al. [5] showed the limitations of different constraint solvers and SMT (satisfiability modulo theories) solvers when using these models.

To the best of our knowledge, dependency-based models have not been used for localizing faults in spreadsheets. The reason for this may be their inaccuracy. Dependency-based models compute a significantly higher number of diagnoses than value-based models. In this paper, we propose a novel type of dependency-based model which uses equivalence instead of the implication to model the dependency relation between cells. This allows improvements on the diagnostic accuracy while keeping the computation times short.

In order to demonstrate the differences between the value-based, the original dependency-based, and our improved dependency-based model, we make use of a running example. This example is a simplified version of the "homework/budgetone" spreadsheet taken from the EUSES spreadsheet corpus [6]. We manually injected a fault into the spreadsheet in cell `D5`. Figure 1a shows the normal (or value) view of this faulty spreadsheet variant. Figure 1b shows the formula view of the same spreadsheet. The output cells[1] of the spreadsheet are shaded in gray. The faulty cell `D5` is highlighted with a red box. The fault manifests in the value of the output cell `D7`. The expected value for this cell is 78,6 %.

	A	B	C	D
1	**Item**	**1st Qtr**	**2nd Qtr**	**Total**
2	**Units Sold**	1000	1500	2500
3	**ASP/Unit**	$ 20	$ 21	$ 20,6
4	**Sales Revenue**	$ 20.000	$ 31.500	$ 51.500
5	**Expenses**	$ 5.000	$ 6.000	$ 5.000
6	**Operating Income**	$ 15.000	$ 25.500	$ 46.500
7	**Op Income in %-Sales**	75,0 %	81,0 %	90,3 %

(a) Normal view

	A	B	C	D
1	**Item**	**1st Qtr**	**2nd Qtr**	**Total**
2	**Units Sold**	1000	1500	=SUM(B2:C2)
3	**ASP/Unit**	20	21	=D4/D2
4	**Sales Revenue**	=B3*B2	=C3*C2	=SUM(B4:C4)
5	**Expenses**	5000	6000	=SUM(B5:B5)
6	**Operating Income**	=B4-B5	=C4-C5	=D4-D5
7	**Op Income in %-Sales**	=B6/B4	=C6/C4	=D6/D4

(b) Formula view

Figure 1: Running example

When using model based debugging, the faulty cell of this example spreadsheet can be detected independent of the underlying model. There are however differences with respect to runtime and solution size. The value-based model and our novel dependency-based model identify three cells that

[1] An output cell is a cell that is not referenced by any other cell.

could explain the observed misbehavior, while the original dependency-based model identifies six cells as possible explanations. When using the constraint solver MINION [7], both dependency-based models require only one third of the computation time compared with the value-based model.

This work is related to the work of Mayer and Stumptner [15] and Wotawa [17]. Mayer and Stumptner evaluated models for model-based debugging in the software domain. Wotawa discussed the relationship of model-based debugging and program slicing.

In the remainder of this paper, we will explain the different types of models and show the model representations for the above described running example (see Section 2). In Section 3, we will empirically compare the different models with respect to efficiency and effectiveness. The novel dependency-based model reduces the number of computed diagnoses by 15 % compared to original dependency-based model. Furthermore, the empirical evaluation shows that the dependency-based models can be solved in less than one second even for those spreadsheets whose value-based models require more than 20 minutes solving time.

2. MODEL-BASED DEBUGGING

In model-based software debugging (MBSD), the cells of a spreadsheet and the given observations (i.e., the test case[2]) are converted into constraints. As the given test case is a failing test case, this constraint system results in a contradiction. In order to determine which cells could resolve this contradiction, MBSD introduces abnormal variables (AB) for each cell. These abnormal variables represent the "health" state of the cells. If a cell c is not abnormal, the formula of the cell must be correct [18]:

$$\neg AB(c) \rightarrow \text{constraint}(c). \tag{1}$$

This logic expression can be transformed to

$$AB(c) \vee \text{constraint}(c). \tag{2}$$

Having such a constraint system, we are able to use a constraint or SMT solver to determine which abnormal variables have to be set to `true` to eliminate the contradiction. We refer the interested reader to [18, 14] for more information about the principles of MBSD. MBSD can be performed using either dependency-based or value-based models. We discuss these models and our novel dependency-based model in the following subsections.

2.1 Value-based models

When using value-based models, the values of the cells are propagated. A value-based constraint system contains (i) the input cells and their values, (ii) the output cells and their expected values, and (iii) all formulas concatenated with their abnormal variable. The constraint representation handles the formulas as equations instead of assignments. This allows to draw conclusions on the input from the output of a formula. Such a value-based model for spreadsheets is proposed by Abreu et al. [3, 4]. The running example from Figure 1b is converted into the following constraints:

[2]A test case is a tuple (I, O), where I are the values for the input cells and O the expected values for the output cells. A test case is a failing test case if at least one computed output value differs from the expected value.

Input:			*Output:*		
$B2$	==	1000	$D3$	==	20.6
$C2$	==	1500	$B7$	==	0.75
$B3$	==	20	$C7$	==	0.81
...			$D7$	==	0.786

Formula constraints:

$$\begin{aligned}
AB(\text{cell}_{D2}) &\vee D2 == B2 + C2 \\
AB(\text{cell}_{D3}) &\vee D3 == D4/D2 \\
AB(\text{cell}_{B4}) &\vee B4 == B3 \times B2 \\
AB(\text{cell}_{C4}) &\vee C4 == C3 \times C2 \\
AB(\text{cell}_{D4}) &\vee D4 == B4 + C4 \\
AB(\text{cell}_{D5}) &\vee D5 == B5 \\
&\dots \\
AB(\text{cell}_{D7}) &\vee D7 == D6/D4
\end{aligned}$$

Solving this constraint system leads to three possible solutions: Either cell D5, D6 or D7 must contain the fault.

2.2 Original dependency-based models

When using dependency-based models, only the information about whether the computed values are correct is propagated. Therefore, all cell values are represented as Boolean instead of Integer or Real values. All variables representing input cells are initialized with `true`. All variables representing correct output cells are also initialized with `true`. The variables representing erroneous output cells are initialized with `false`. Instead of using the concrete formulas in the constraints, only the correctness relation is modeled. If the formula of cell c is correct and the input values of a formula are correct then cell c must compute a correct value:

$$AB(\text{cell}_c) \vee \bigwedge_{c' \in \rho(c)} c' \rightarrow c \tag{3}$$

where $\rho(c)$ is the set of all cells that are referenced in c. Details about this modeling for software written in an imperative language can be found e.g. in [17]. The dependency-based constraints for our running example are as follows:

Input:			*Output:*		
$B2$	==	true	$D3$	==	true
$C2$	==	true	$B7$	==	true
$B3$	==	true	$C7$	==	true
...			$D7$	==	false

Formula constraints:

$$\begin{aligned}
AB(\text{cell}_{D2}) &\vee (B2 \wedge C2 \rightarrow D2) \\
AB(\text{cell}_{D3}) &\vee (D2 \wedge D4 \rightarrow D3) \\
AB(\text{cell}_{B4}) &\vee (B2 \wedge B3 \rightarrow B4) \\
AB(\text{cell}_{C4}) &\vee (C2 \wedge C3 \rightarrow C4) \\
AB(\text{cell}_{D4}) &\vee (B4 \wedge C4 \rightarrow D4) \\
AB(\text{cell}_{D5}) &\vee (B5 \rightarrow D5) \\
&\dots \\
AB(\text{cell}_{D7}) &\vee (D4 \wedge D6 \rightarrow D7)
\end{aligned}$$

Solving this constraint system leads to six possible solutions: Either cell B4, C4, D4, D5, D6 or D7 must contain

the fault. This dependency-based model computes more diagnoses because of the implication. In the value-based model, the cells B4, C4, and D4 can be excluded from the set of possible diagnoses because B4 and C4 are used to compute D4, and D4 is used to compute D3, which is known to compute the correct value. Unfortunately, this information gets lost when using the implication because the implication allows conclusions only from the input to the output but not vice versa. This problem will be solved with the novel dependency-based model that is explained in the following subsection.

2.3 Novel dependency-based models

In order to eliminate the previously described weakness of dependency-based models, we use bi-implication (equivalence) instead of the implication. The formula constraints for our running example from Figure 1b are as follows:

$$\begin{aligned}
AB(\text{cell}_{D2}) &\vee (B2 \wedge C2 \leftrightarrow D2)\\
AB(\text{cell}_{D3}) &\vee (D2 \wedge D4 \leftrightarrow D3)\\
AB(\text{cell}_{B4}) &\vee (B2 \wedge B3 \leftrightarrow B4)\\
AB(\text{cell}_{C4}) &\vee (C2 \wedge C3 \leftrightarrow C4)\\
AB(\text{cell}_{D4}) &\vee (B4 \wedge C4 \leftrightarrow D4)\\
AB(\text{cell}_{D5}) &\vee (B5 \leftrightarrow D5)\\
&\dots\\
AB(\text{cell}_{D7}) &\vee (D4 \wedge D6 \leftrightarrow D7)
\end{aligned}$$

Solving this constraint system leads to the same 3 diagnoses as when using a value-based model.

The bi-implication cannot be used in case of coincidental correctness. Coincidental correctness might occur for example in the following situations:

- usage of conditional function (e.g., the IF-function),
- abstraction function like MIN, MAX, COUNT,
- usage of Boolean,
- multiplication with a 0-value, and
- power with 0 or 1 as base number or 0 as exponent.

Please note, that this list gives only examples. It is not a complete list, because the size of the list depends on the functions that are supported by the concrete spreadsheet environment (e.g. Microsoft Excel, iWorks'Number, OpenOffice's Calc). All formulas where coincidental correctness might happen still have to be modeled with the implication instead of the bi-implication.

3. EMPIRICAL EVALUATION

This section consists of two major parts: the empirical setup (discussing the prototype implementation, the used platform, and the evaluated spreadsheet corpora) and the results showing that dependency-based models are able to compute diagnoses within a fraction of a second even for spreadsheets whose value-based models require more than 20 minutes of solving time. In addition, this empirical evaluation shows that the number of diagnoses obtained by the novel dependency-based model is reduced by 15 % on average compared to the original dependency-based model.

We developed a prototype in Java for performing the empirical evaluation. This prototype uses MINION [7] as a constraint solver. MINION is an out-of-the-box, open source constraint solver and offers support for almost all arithmetic, relational, and logic operators such as multiplication, division, and equality over Boolean and Integers.

The evaluation was performed on an Intel Core2 Duo processor (2.67 GHz) with 4 GB RAM and Windows 7 as operating system. We used the MINION version 0.15. The computation time is the average time over 10 runs.

We evaluated the models by means of spreadsheets from the publicly available Integer spreadsheet corpus[3] [5]. This corpus contains 33 different spreadsheets (12 artificially created spreadsheets and 21 real-life spreadsheets), e.g., a spreadsheet that calculates the lowest price combination on a shopping list or the winner of Wimbleton 2012. These spreadsheets contain both arithmetical and logical operators as well as the functions SUM and IF. The spreadsheets contain on average 39 formula cells, the largest spreadsheet contains 233 formulas. Faulty versions of the spreadsheets (containing single, double and triple faults) were created by randomly selecting formulas and applying mutation operators [2] on them. The corpus contains in total 220 mutants. In the empirical evaluation, we used only the spreadsheets which contain a single fault, i.e. 94 spreadsheets.

Table 1 compares the three types of models with respect to fault localization capabilities and runtimes. The fault localization capabilities are expressed by means of the number of cells that are single fault diagnoses. The runtime is measured by means of MINION's average solving time in milliseconds. The spreadsheets are divided into two subgroups: spreadsheets whose value-based models are solved by MINION in less then 20 minutes and spreadsheets whose value-based models could not be solved within 20 minutes (i.e. 31 from 94 spreadsheets). For these 31 spreadsheets, the dependency-based models are solved in less than one second. These runtime results indicate that dependency-based models are better suited for debugging large spreadsheets than value-based models.

Table 1: Evaluation results

Model	Single fault diagnoses	Solving time (in ms)
63 spreadsheets		
Value-based	4.0	56818.8
Original dep.-based	13.2	32.0
Novel dep.-based	11.0	31.6
31 spreadsheets		
Value-based	-	$>$ 20 minutes
Original dep.-based	45.0	187.4
Novel dep.-based	38.6	164.8

Considering the diagnostic accuracy, the value-based model yields better results. It computes only one third of the diagnoses of the original dependency-based model. The improved dependency-based model computes on average 15 % less diagnoses than the original dependency-based model.

Table 2 gives an overview of the reduction that can be achieved when using the novel instead of the original dependency-based model. The reduction is expressed by means of the following metric:

$$\text{REDUCTION} = 1 - \frac{|\text{Diagnoses in the novel model}|}{|\text{Diagnoses in the original model}|}. \quad (4)$$

[3] https://dl.dropbox.com/u/38372651/Spreadsheets/Integer_Spreadsheets.zip

Table 2: Summary of the achieved reduction when using the novel model instead of the original dependency-based model

Reduction	Number of spreadsheets
0 %	64
]0 %;10 %]	0
]10 %;20 %]	1
]20 %;30 %]	1
]30 %;40 %]	2
]40 %;50 %]	5
]50 %;60 %]	0
]60 %;70 %]	4
]70 %;80 %]	2
]80 %;90 %]	7
]90 %;100 %]	8

For 64 spreadsheets, no reduction in the number of diagnoses was achieved when using the novel dependency-based model instead of the original model. However, for 15 spreadsheets, a reduction of more than 80 % was achieved.

4. DISCUSSION AND CONCLUSIONS

Locating faulty formulas in spreadsheets can be time consuming. This paper addresses the fault localization problem by means of model-based diagnosis. Our most important contribution is the introduction of a novel dependency-based model. This novel dependency-based model improves previous work in two ways: (1) Compared to the original dependency-based model, it reduces the amount of diagnoses that have to be manually investigated by 15 %. (2) Compared to the value-based model, it reduces the required solving time and allows the computation of diagnoses in real-time where the value-based model cannot compute solutions within 20 minutes. The savings in computation time can be explained by the reduction of the domain: The dependency-based model requires only Boolean variables instead of Integers and Real numbers.

The reduction of the domain comes with additional advantages: (1) An arbitrary solver can be used, because all solvers support at least Boolean variables. Even spreadsheets containing Real numbers can be debugged with any solver when using dependency-based models. (2) The user does not need to indicate concrete values for the erroneous output variables. The information that an output cell computes the wrong value is sufficient.

In the description of the model, we listed all types of coincidental correctness occurring in the spreadsheets used in the empirical evaluation. This list is not exhaustive. For using this model in practice, the list has to be extended.

We are convinced that the model presented improves the state of the art in model-based diagnosis. Further work includes a user study and the adaptation to other types of programs, e.g. programs written in imperative or object-oriented languages.

Acknowledgments

The research herein is partially conducted within the competence network Softnet Austria II (www.soft-net.at, COMET K-Projekt) and funded by the Austrian Federal Ministry of Economy, Family and Youth (bmwfj), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

5. REFERENCES

[1] R. Abraham and M. Erwig. GoalDebug: A Spreadsheet Debugger for End Users. In *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007)*, pages 251–260, 2007.

[2] R. Abraham and M. Erwig. Mutation Operators for Spreadsheets. *IEEE Transactions on Software Engineering*, 35(1):94–108, 2009.

[3] R. Abreu, A. Riboira, and F. Wotawa. Constraint-based debugging of spreadsheets. In *CibSE'12*, pages 1–14, 2012.

[4] R. Abreu, A. Riboira, and F. Wotawa. Debugging of spreadsheets: A CSP-based approach. In *3th IEEE Int. Workshop on Program Debugging*, 2012.

[5] S. Außerlechner, S. Fruhmann, W. Wieser, B. Hofer, R. Spörk, C. Mühlbacher, and F. Wotawa. The right choice matters! SMT solving substantially improves model-based debugging of spreadsheets. In *Proceedings of the 13th International Conference on Quality Software (QSIC'13)*, pages 139–148. IEEE, 2013.

[6] M. Fisher and G. Rothermel. The EUSES Spreadsheet Corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. *SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.

[7] I. P. Gent, C. Jefferson, and I. Miguel. Minion: A fast, scalable, constraint solver. In *Proceedings of the 7th European Conference on Artificial Intelligence (ECAI 2006)*, pages 98–102, 2006.

[8] B. Hofer, A. Perez, R. Abreu, and F. Wotawa. On the empirical evaluation of similarity coefficients for spreadsheets fault localization. *Automated Software Engineering*, pages 1–28, 2014.

[9] B. Hofer, A. Riboira, F. Wotawa, R. Abreu, and E. Getzner. On the Empirical Evaluation of Fault Localization Techniques for Spreadsheets. In *Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering (FASE 2013)*, pages 68–82, Rome, Italy, 2013.

[10] D. Jannach, A. Baharloo, and D. Williamson. Toward an integrated framework for declarative and interactive spreadsheet debugging. In *Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2013)*, pages 117–124, 2013.

[11] D. Jannach and U. Engler. Toward model-based debugging of spreadsheet programs. In *JCKBSE 2010*, pages 252–264, Kaunas, Lithuania, 2010.

[12] D. Jannach and T. Schmitz. Model-based diagnosis of spreadsheet programs: a constraint-based debugging approach. *Automated Software Engineering*, pages 1–40, 2014.

[13] C. Mateis, M. Stumptner, D. Wieland, and F. Wotawa. Model-based debugging of Java programs. In *Proceedings of AADEBUG*, 2000.

[14] W. Mayer and M. Stumptner. Model-based debugging – state of the art and future challenges. *Electronic Notes in Theoretical Computer Science*, 174(4):61–82, May 2007.

[15] W. Mayer and M. Stumptner. Evaluating models for model-based debugging. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2008.

[16] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[17] F. Wotawa. On the Relationship between Model-Based Debugging and Program Slicing. *Artificial Intelligence*, 135:125–143, February 2002.

[18] F. Wotawa, M. Nica, and I.-D. Moraru. Automated debugging based on a constraint model of the program and a test case. *The journal of logic and algebraic programming*, 81(4), 2012.

A Spreadsheet Cell-Meaning Model for Testing

Daniel Kulesz
Institute of Software Technology
University of Stuttgart
daniel.kulesz@informatik.uni-stuttgart.de

ABSTRACT

Most attempts to test spreadsheets use the spreadsheet-internal model to automatically detect input, intermediate and output cells. In this paper, we discuss an example which shows why this approach is problematic even for simple spreadsheets. Following this, we derive a number of requirements for more feasible spreadsheet cell-meaning models and describe a first prototype we have designed.

1. INTRODUCTION

Regardless of the hazards which can arise when using spreadsheets, most businesses today regard them as indispensible tools for supporting their processes. This fact, together with the billions of existing spreadsheets [7], indicates a strong need for finding faults in spreadsheets.

There are many approaches for finding faults in spreadsheets [6]. One of them is testing, where the spreadsheet's input cells are populated with values and the values in the spreadsheet's output cells checked for certain criteria (e.g. [5, 3]). To accomplish this, knowing which of the spreadsheet's cells are input cells and which are output cells is mandatory. However, virtually no testing approaches use a designated model for this purpose. Instead, they rely on the model that spreadsheet execution environments use internally for (re)calculation purposes, and which can easily be extracted when considering the dependencies between cells:

- Input cells: Non-formula cells referenced by formula cells
- Intermediate cells: Formula cells referenced by other cells and referencing other cells themselves
- Result cells: Formula cells not referenced by other cells but referencing other cells themselves

We will refer to this model as the 'naive model' throughout this paper. While this model is certainly correct from a technical point of view, looking at it from the perspective of a spreadsheet user's domain can lead to imminent conflicts in a number of cases. In this paper we discuss these cases and propose an explicit model which is more difficult to extract but which we believe is better suited for testing spreadsheets.

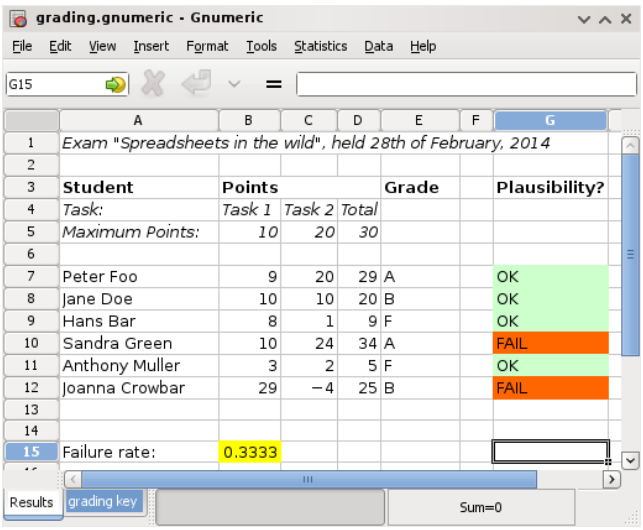


Figure 1: Example for a grading spreadsheet

Since the model is concerned with the type of cells from the perspective of what the cells mean to users, we could refer to it as "cell-type model". Unfortunately, the term "cell-type" is usually already used for describing the data type of a spreadsheet cell's contents. To avoid confusion, we use the notion of "cell-meaning model" instead.

2. ISSUES WITH THE NAIVE MODEL

Figure 1 shows a spreadsheet for managing grades of a study course. The spreadsheet is filled with data by a course instructor and later passed to a secretary for transferring the grades to a different system. Furthermore, the spreadsheet is used for statistical purposes by the manager of the study course this exam belongs to. We want to use this spreadsheet as a showcase with counter examples, arguing why the naive model can lead to a biased perception of the actual input cells and output cells of a spreadsheet:

- The total points (D7 to D12) could be output values

for a secretary who has to process these grades further (e.g. write letters to students), but these cells are referenced by the grade cells (E7 to E12). Thus, the cells would be regarded as intermediate cells by naive models and not as output cells.

- The study program manager might not be interested at all in the total points of the particular students but only in the failure rate (B15). Thus, he would not see the grades as output cells.
- Spreadsheet authors sometimes use defensive programming techniques and introduce checks themselves. The plausibility column (G) is such an example: It checks whether any of the grades for Task 1 or Task 2 are outside acceptable limits (zero to maximum points for the task). The naive model would treat the cells in the plausibility column (G7 to G12) as output as well.
- The second worksheet named 'grading key' is referenced by VLOOKUP functions in the grade column's cells (E7 to E12). The naive model would therefore interpret these referenced cells as input cells. However, none of the users of this spreadsheet is supposed to change the contents of these cells as they contain merely static data.

3. REQUIREMENTS

We are convinced that biased perceptions can be reduced if a spreadsheet is tested using a model explicitly designed for this purpose. From the discussion in the previous section, we derive the following requirements for such a model:

- **User-specifiable:** It must be possible for users to specify the cells themselves. If automatic extraction is used, users must be able to change it.
- **Support for views:** Since users have different perceptions and needs of the same spreadsheet, we either need one testing model with different views or it must be allowed that more than one testing model instance per spreadsheet exists.
- **Input cell types:** The model must separate input cells at least into two types: data cells (which contain data that rarely changes or is fed from another system) and actual decision variables which are supposed to be manipulated by the user (of this model).
- **Output cell types:** Apart from intermediate cells, the model must support cells which provide the data with final results (which the user is looking for) as well as support for plausibility and other additional cells.

Apart from these rather theoretical requirements, we identified two major practical requirements for the success of the implementation of such a model:

- **Understandability:** It must be easy for spreadsheet users to understand the model with no or little training, so that users can identify cells correctly in the sense of the model.
- **Acceptance:** Even if the model would be easily understandable for spreadsheet users, its benefits must be striking so that spreadsheet users will be motivated to take the effort connected with using the model. (Basically this requires a proper attention investment model as described by Blackwell and Burnett [2] [1]).

It seems infeasible to expect spreadsheet users to identify all cell-meanings manually, especially for huge spreadsheets. As already discussed, fully automated cell-meaning detection seems impossible — but it might be beneficial to consider assisting users by *proposing* cell-meanings based on auto-detection techniques.

A promising starting point could be the work of Hermans [4] which tries to identify plausibility cells automatically by inspecting result cells for two additional constraints: the formula starting with the IF operation and containing at least one branch which can result in a string. While this certainly works in many cases (including our small example), we have already seen spreadsheets which use numeric outputs for plausibility cells so this approach would fail. Yet, since such cases are pretty rare, asking users to just validate auto-detected cell-meanings instead of asking them to specify cell-meanings themselves might result in lower overall effort and thus higher acceptance.

4. PROTOTYPE

We prototyped a cell-meaning model which takes into account the requirements stated in the previous section. The model is illustrated in Figure 2 as a standard UML class diagram.

We provide a partial implementation of our prototype in our tool 'Spreadsheet Inspection Framework' which is available as open source software from GitHub[1]. The implementation allows users to manually mark cells and use them later for specifying test cases, but does not support all proposed cell-meaning types yet and lacks auto-detection capabilities.

5. FUTURE WORK

To assess the feasibility of the model, further research is required. A crucial aspect for evaluation will be the question whether the cell types can be communicated clearly to users, so users will tag existing spreadsheet cells according to our proposed model.

Another important aspect will be the acceptance of the model. Since we believe that acceptance might be very low without a reasonable level of automated assistance, it seems worthwhile to address this point first.

Although we explained the theoretical limitations of the naive model in this work, it must be explored whether the additional manual effort connected with applying our model yields enough benefits in terms of its ability to detect faults in spreadsheets more accurately.

[1]https://github.com/kuleszdl/Spreadsheet-Inspection-Framework

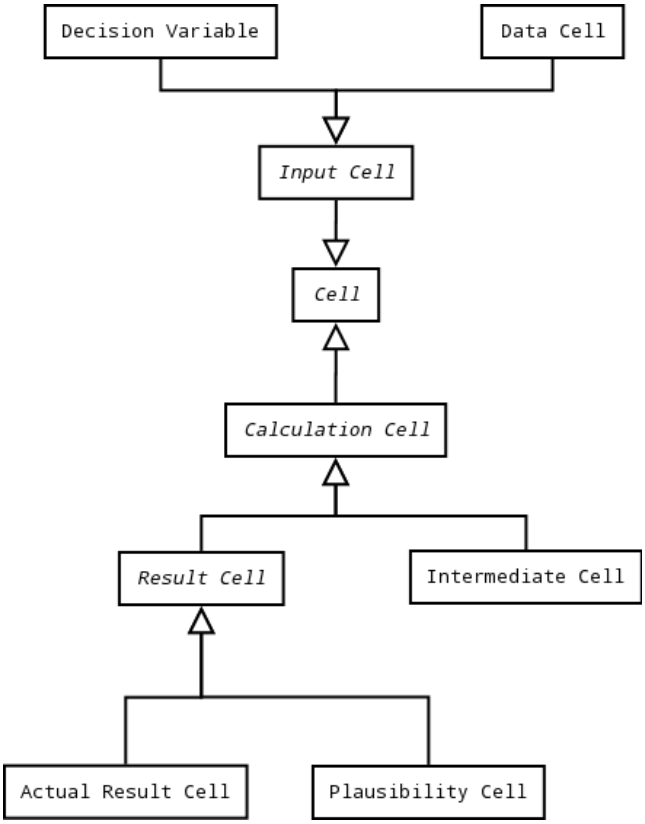


Figure 2: The cell-meaning model we propose

6. ACKNOWLEDGEMENT

We would like to thank Zahra Karimi, Kornelia Kuhle, Mandy Northover, Jochen Ludewig and Stefan Wagner for their constructive feedback on earlier versions of this position paper. Furthermore, we received many good hints and comments from the reviewers for which we are very thankful.

7. REFERENCES

[1] A. Blackwell and M. Burnett. Applying attention investment to end-user programming. In *Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on*, pages 28–30. IEEE, 2002.

[2] A. F. Blackwell. First steps in programming: A rationale for attention investment models. In *Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on*, pages 2–10. IEEE, 2002.

[3] M. Fisher II, G. Rothermel, D. Brown, M. Cao, C. Cook, and M. Burnett. Integrating automated test generation into the wysiwyt spreadsheet testing methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(2):150–194, 2006.

[4] F. Hermans. Improving spreadsheet test practices. *Center for Advanced Studies on Collaborative Research, CASCON*, 2013.

[5] D. Jannach, A. Baharloo, and D. Williamson. Toward an integrated framework for declarative and interactive spreadsheet debugging. In *Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 117–124. SciTePress, 2013.

[6] D. Jannach, T. Schmitz, B. Hofer, and F. Wotawa. Avoiding, finding and fixing spreadsheet errors–a survey of automated approaches for spreadsheet qa. *Journal of Systems and Software*, 2014.

[7] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 207–214. IEEE, 2005.

SBBRENG: Spreadsheet Based Business Rule Engine

Pablo D. Palma

Incentings
Latadia 4623, Santiago, Chile
+562 2207 7158
pablo.palma@incentings.com

ABSTRACT
We developed a software product to replace the use of spreadsheets as a data processing solution within a specific business area. This paper explains the characteristics of the tool and the findings, both resulting from a process of 3 years real life refinement inside the ICM domain, and that we postulate can be valid in other business domains.

General Terms
Documentation, Design, Security, Human Factors, Languages. Verification

Keywords
Spreadsheet, Business Rules Engine, DDD, SEmS'14

1. INTRODUCTION

1.1 Use of spreadsheets for ICM solutions
The last four years our company has been working in the field of Incentive Compensation Management –ICM-. Current solutions, based on calculating performance-based payment for employees, are complex and highly dynamic.

Worldwide, "only a 10% of sales organizations with more than 100 payees deploy prepackaged sales ICM applications" [1]. Almost all the remainder uses Excel. This is a reaction to the combination of factors: high rate of change, short time available for implementation, and typically long cycles in IT development. However, Excel introduces its own limitations. It requires a lot of human intervention that results in overpayment, and user-generated errors that could be reduced "by more than 90%" [1] (see subsection 5.2). In addition, there is "dissatisfaction with the reliability of spreadsheets in adequately supporting compensation processes" [1]. Excel also does not accomplish auditing, accounting and regulation requirements.

In our market, the most important features of ICM software are flexibility, security, auditing capabilities, and allowing the end users to update the product themselves by including changes in business rules.

1.2 Goals for a new ICM software
Some or the issues of pre-existing ICM solutions are:

- World Class ICM software solutions are costly and demand long implementation processes
- In-house developments are slow[1] and rigid[2]
- Excel-based solutions are fragile, difficult to audit and error prone[3]
- Currently available solutions don't attempt to improve problem representation[4] beyond conventional system documentation
- Excel formulas are one-line expressions and are thus difficult to read (e.g. nested if statements)

1.3 Importance and state of our work
There are two elements we consider important. First, we are putting in practice some ideas (see Section 8) that may be useful in other areas of enterprise software development. Second, we want to determine how well our selection of functionalities succeeds in creating a tool that best takes advantage of a mental model of spreadsheets.

Customized ICM applications developed with SBBRENG have been in use for more than a year in several companies from different areas: car dealerships, banks, retail, etc. This happens in the Chilean market where we use the product name IM4

2. BUSINESS RULES ENGINE
Business rules engines aren't a new product category, they started around the 80s [2]. Since then, many products and companies have undergone a cycle of creation, development, merging and death. We will use two currently successful solutions as comparison standards: Drools (see Drools Guvnor Knowledge Base)[5] [3] -a component of the open source platform JBoss BRMS- and ODM [3, 4] by IBM. Both are much larger systems than SBBRENG, sharing the same global objective: make the application more *business agile*.

Drools is a low level programming environment oriented to efficiently manage a large quantity of conditions of any type. It provides APIs for integration with other languages, tools and processing environments. On the other hand, Operational Decision Management –ODM- is a more business oriented solution that conceptually splits systems into two different components, talking to each other under a data contract. One is a traditional Data Processing System for storing, updating and reporting information related to some business domain, and the other is a specialized system for managing and executing the business rules of the same business domain.

[1] in the range of 2 hrs per 2.000 transactions

[2] no provisions for isolation or special management of business logic

[3] http://eusprig.org/horror-stories.htm

[4] *problem representation* has impact on the maintenance agility same as on the ability to preserve application coherence

[5] http://drools.jboss.org/drools-guvnor.html

SBBRENG is closer to ODM with some big differences: domain model is not Object Oriented and input/output documents are simple shared folders for storing interchanging files

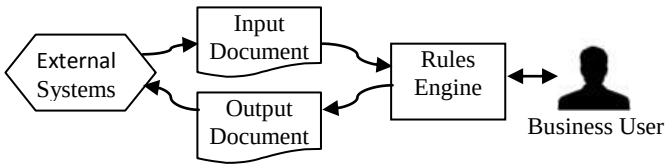


Figure 1: ODM high level view

3. THE SPREADSHEET-LIKE SIDE OF SBBRENG

3.1 The *ApplyRules* operation

A *WorkSheet* -WS- is a set of files and columns such as each column has a unique name and each cell stores an immutable value (current implementation does not yet force this immutability). A SBBRENG Process is a specific sequence of steps that modifies a WorkSheet. There is an operation *ApplyRules* (•) for implementing SBBRENG Processes, following statements of Business Rules. A Business Rule is -in the context of SBBRENG- a directive detailing how to calculate the numeric values used to run the business.

We represent a SBBRENG Process using the formula

$BR_k \bullet WS_p \Rightarrow (WS^k_p, OF_p)$

Where:
- • is the *ApplyRules* operation
- BR_k is a subset of the Business Rules comprising the Application
- WS_p is a current WorkSheet that is part of the Application
- WS^k_p is a new WorkSheet that will be part of the Application
- OF_p is an Output File that consists of a subset of WS^k_p

Operation • adds new columns at the right of WS_p. Business Rules define how to calculate the immutable values of the new cells. New columns can reference any column located at its left (Figure 2).

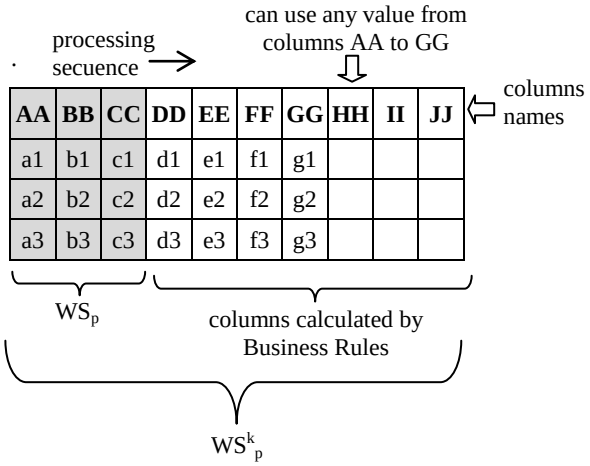


Figure 2: A SBBRENG Process

A SBBRENG Application is a sequence of Processes as seen below:

Process 1: $BR_1 \bullet WS_a$
Process 2: $BR_2 \bullet WS_b$
……
Process k: $BR_k \bullet WS_m$

3.2 The *Assemble* operation

An Input File becomes a WS when it contains all the information referenced by one or more Business Rules. When Business Rule references are contained in several Input Files, it is necessary to build a WorkSheet by assembling several Input Files. We use the *Assemble* operation (**+**) for this purpose, as shown in the following formula.

$$IF_i(p) + IF_k(q) \Rightarrow WS_a$$

Where **p** is a column of IF_i and **q** is a column of IF_k, and they provide a mechanism for matching rows of the Input Files.

Operation **+** produces a WorkSheet out of all the columns of both Input Files. The WorkSheet contains all the IF_i (p) rows and for each of them, only one matching IF_k (q) row. The matching logic is the same of an Excel Table Lookup operation, in which **p** is a column in the data and **q** represents the first column of the table.

The **+** operation is associative but not commutative.

The **+** operation can also be applied to a WorkSheet. In such cases we have a precedence of Processes. Figure 3 shows an example in which $BR_1 \bullet WS_a$ ***precedes*** $BR_2 \bullet WS_b$.

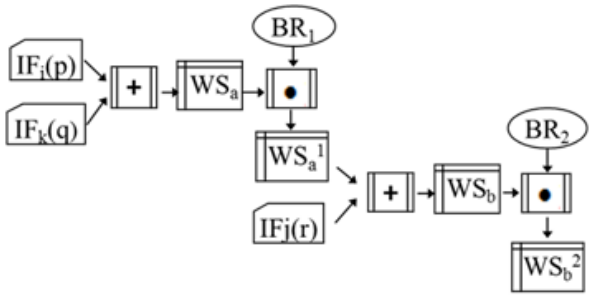


Figure 3: Process precedence

There are situations where it is necessary to assemble the same file more than one time, using different column keys. In such a case SBBRENG adds a prefix to column names to avoid collisions. In the following example, IF_k is applied twice:

$$(IF_i(p) + IF_k(q))(r) + IF_k(s) \Rightarrow WS_b$$

4. NON COMPATIBLE SPREADSHEET FUNTIONALITY

The most important difference with Spreadsheets is spreadsheet interactivity, because SBBRENG follows a batch processing model. Other examples of incompatible features are Table Lookup, Dynamic Tables, external links, totals and other

aggregated values in the same column as the original data, macros, different formulas in the same column, and the programming language.

5. SOME ADITIONAL FEATURES OF SBBRENG

5.1 Referential Transparency

A SBBRENG application offers "referential transparency", which is the base for providing reproducible results. In order to achieve that goal, it is necessary to replace links to external sources (other spreadsheets, Databases, etc.) by static Input Files containing the external linked information.

5.2 Separation of Data and Parameters

In the context of SBBRENG, Parameters are a special type of data: input files are produced by other systems, but parameters are maintained by users. Parameters represent a high level system abstraction, which is required to adapt the system behavior. Data is stored in Files and WorkSheets, and Parameters are stored in a special repository. SBBRENG's IDE provides the means for Parameter editing.

5.3 Iteration over Data

The calculus performed on each column follows a *cycle*. Rows are filtered by conditions and grouped by some column values. The logic applied to the cells belonging to a group, is repeated for each group until reaching the last. Some SBBRENG *core functions* offer aggregated operations over groups, e.g. count, sum, average, max.

5.4 Domain Model

5.4.1 Introduction

Five objects support Domain modeling: *Matrix, Classifier, List, Rules and Files*. *Files* are input/output files. *Rules* are pieces of code that have some specific properties (i.e. name, filter, sequence and granularity). The three remaining objects are the most important, because they store in their structure the values of the Parameters of the application. This allows a direct user interaction with the Domain Model representation, when adjusting Parameters values.

Parameters directly represent elements of the *ubiquitous language* [5] Those elements appear in several real life working documents: memos, agreements, contracts, regulations, etc. The shape of the Parameters as used in SBBRENG mimics its representation in documents. Therefore, business users understand them without requiring further explanations.

As needed, some Parameters may have embedded logic that is executed every time they are used in a Rule.

5.4.2 Matrixes

Matrixes are bi-dimensional arrays of values and conditions that return a value (or several values) based on the evaluation of its embedded logic.

Matrixes have two headings, X and Y. Each heading represents a tree of conditions: sibling nodes make an OR and parent-child nodes make an AND. It is very easy to see the tree as a set of adjacent boxes with the outermost boxes of the headings matching columns or rows of the Matrix. Each box has a label that makes apparent its associated condition.

		New Products			Old Products	
		Spot Clients	Recurrent Clients	Premiun Clients	Recurrent Clients	Premiun Clients
Salesmen type A		5%	6%	7%	7%	8%
Salesmen type B	North Region	4%	5%	6%	0%	0%
	South Region	3%	4%	5%	0%	0%
Salesmen Type C	North Region	5%	6%	7%	7%	8%
	Central Region	4%	5%	5%	6%	6%
	South Region	3%	4%	5%	0%	0%

Figure 4: A Matrix

Matrixes change the way in which complex nested conditions are visualized (see figure 5)

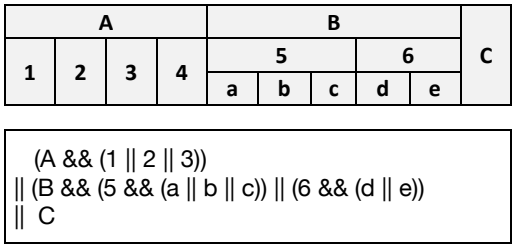


Figure 5: Equivalence of nested conditions

Matrixes are self-explanatory for anyone familiar with the Business Domain of the application. Their behaviour doesn't depend on the context in which they are used; it only depends on the values of some of the input data in a clear and explicit manner. Matrixes provide a powerful mechanism of Domain representation, because of its expressivity and because of the way they isolate behavior.

5.4.3 Classifiers

Classifiers are Boolean expressions whose value is automatically set based exclusively on the input data and remain immutable until the input data change. They represent business concepts, mostly corresponding to nouns in the *ubiquitous language*. Regardless of how many relationships input fields have in the system they comes from, Classifiers implements only those conditions required by our application. Classifiers are used by Matrixes to build its embedded logic.

Classifiers create a conceptual layer for mapping a SBBRENG application Domain with the Domain of systems where the input data were generated. Classifiers are used by Matrixes to build its embedded logic. Classifiers increase program readability and improve our ability to adapt to changes in the Input Files.

5.4.4 List and Constants

A List is a Dictionary where a value associated to an entry can be simple or complex. Constants are Lists that use a special syntax.

5.5 Programming Language

We use JavaScript to replace spreadsheets' functions. To improve productivity, we developed a library of *"core functions"*

frequently used in our Domain of applications. It is easy to add new core functions.

We also provide a graphic block language, similar to MIT's Scratch [6] and others [7]. Blocks automatically generate the equivalent JavaScript instructions. Blocks are very well suited to SBBRENG because each Rule is made of a few instructions. Blocks were initially implemented for the *Assemble* operation, and we have plans to extend it to the Rules.

5.6 Auditing

A *Run* is a complete execution of a SBBRENG Application. Each Run is stored as a *backup document* containing all inputs, outputs, parameters and logic utilized. SBBRENG automatically assigns a unique ID to each Run. Later, a Run can be opened as read-only for revision, but it cannot be modified. It is possible to reprocess a backup document, generating a new backup document with a different ID.

Additionally, there is a log of the changes made to the parameters, the input files and the logic, indicating old and new values, the user involved and date/time of all changes.

5.7 IDE

There is a special IDE -Integrated Development Environment- to support all tasks: application development, documentation, design, testing, etc. It also has functions for running applications, for reviewing previous Runs and for downloading results.

The IDE offers two views: a conventional nested folders type and an advanced *mental map* type [8, 9]. The latter is the base for some advanced visualization options that ease the understanding of an Application (pending development).

5.8 Documentation

Documentation is a part of a broader content we call *problem representation.* It includes parameters, code, blocks, ad-hoc descriptions, etc. Additional to the content, there are tools for filtering information, displaying information, and displaying information relationships. Some of this functionality is currently in use; some is pending development. Because documentation is supported by the IDE, it is always available on line when working with the application.

6. SOFTWARE STRUCTURE

On the Server side, there is a Web application than runs on IIS using .NET and SQL Server.

On the Client side, there is the IDE running in any modern browser.

7. RESULTS

Security and auditability of the applications were improved in relationship to spreadsheets, as a result of some new specific functionality (see Subsections 5.1, 5.2 and 5.6).

Documentation was improved when compared to conventional solutions, because of the integration of different types of information into one common repository (see Subsections 5.7, 5.8) and the availability of new capabilities based on the use of a Mental Map.

The use of the Domain Model (see Subsection 5.4) enhanced productivity of development and maintenance, because less code is required to implement the same business logic compared to solutions using spreadsheet (See Sub Subsection 5.4.2)

Performance is good. We were expecting 10 min per 2.000 transactions and 4 hrs per 3.000.000 transactions, but real numbers were 4 min and 1 hr 45min, respectively. We were using a conventional entry level server.

8. KEY LESSONS LEARNED FROM WORKING WITH SBBRENG

Looking at one of the components of the productivity equation, we think we successfully tried some new ideas, like a new approach for representing the Business Rules Domain, a method for avoiding complex nested conditions, an IDE based in a Mental Map, a graphic replacement for the programming language of spreadsheets, some mechanisms to improve security and auditability, etc.

But looking at the other component -the process of getting and agreeing to specifications for building the application- we think it is necessary to achieve important improvements. The ubiquitous language requires more elaboration[6]. The cognitive process that ends with a working application can probably take advantage of the impressive new findings in neuroscience. Focus, resources, new instruments and new methodologies are moving the limits. “Constant development of more sensitive and accurate neuroimaging and data analysis methods creates new research possibilities” [10].

9. FUTURE DEVELOPMENTS

We are interested in two areas for future development. The first is improving automatic analysis capabilities used during the testing phase, and the other is improving visualization capabilities for mental maps in the IDE.

10. REFERENCES

[1] Dunne, M. 2010. *MarketScope for Sales Incentive Compensation Management Software*. Gartner MarketScope Series (March 2010)

[2] Bosh 2010. *The Past, Present, and Future of Business Rules*. Bosch Software Innovations GmbH. (March 2010)

[3] Craggs, S. 2012. *Competitive Review of Operational Decision Management*, Lustratus Research (October 2012)

[4] IBM 2012. *Why IBM Operational Decision Management?* Software. Thought Leadership White Paper (June 2012)

[5] Evans, E. 2003. *Domain-Driven Design*. Addison Wesley; E (August 2003)

[6] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai Y. *Scratch: Programming for all*. Communications of the ACM (November 2009)

[6] it has been apparent that some additional concepts are necessary

[7] Hosick, E. 2014. Visual Programming Languages - Snapshots. (February 2014) http://blog.interfacevision.com/design/design-visual-progarmming-languages-snapshots/

[8] Eppler, M. 2006. A comparison between concept maps, mindmaps, conceptual diagrams, and visual metaphors as complementary tools for knowledge construction and sharing. Faculty of Communication Sciences, University of Lugano (USI), Lugano, Switzerland

[9] Novak, J., and Cañas, A. 2008. *The Theory Underlying Concept Maps and How to Construct and Use Them.* Florida Institute for Human and Machine Cognition (IHMC)

[10] Jääskeläinen, L. 20012. Cognitive Neuroscience: Understanding the neural basis of the human mind . Jääskeläinen & Ventus Publishing ApS (November 2012)| http://bookboon.com/

.

End-user development via sheet-defined functions

Peter Sestoft* Jonas Druedahl Rask Simon Eikeland Timmermann

ABSTRACT

We have developed an implementation of *sheet-defined functions*, a mechanism that allows spreadsheet users to define their own functions, using only spreadsheet concepts such as cells, formulas and references, and no external programming languages. This position paper presents the motivation and vision of this work, describes the features of our prototype implementation, and outlines future work.

Keywords

Spreadsheets, end-user development, functional programming

1. INTRODUCTION

Spreadsheet programs such as Microsoft Excel, OpenOffice Calc, Gnumeric and Google Docs are used by millions of people to develop and maintain complex models in finance, science, engineering and administration. Yet Peyton Jones, Burnett and Blackwell [9] observed that spreadsheet programs lack even the most basic abstraction facility — a way to encapsulate an expression as a reusable function — and proposed a design for such a mechanism.

We have implemented this idea in the form of *sheet-defined functions*. A user may define a function `F` simply by declaring which (input) cells will hold `F`'s formal parameters and which (output) cell will compute `F`'s result, as a function of the input cells. The function definition may involve arbitrary additional cells, spreadsheet formulas, calls to built-in functions, and calls to other sheet-defined functions. Figure 1 shows an example. In the example, values are referred to by cell (such as B25). A mechanism that allows for symbolic names (such as "periods") instead could be added, but Nardi speculates that end user developers would not necessarily find that better [7, page 44].

Augustsson et al. from Standard Chartered Bank provide further support for the utility of such abstraction mechanisms, saying about the traditional combination of Excel and externally defined functions that "change control, static type checking, abstraction and reuse are almost completely lacking" [1].

*sestoft@itu.dk, IT University of Copenhagen, Denmark

E34	A	B	C
22	=DEFINE("nominal", B26, B23, B24)		
23	'effective =	0.053543	
24	'npery =	4	
25	'periods =	=FLOOR(B24, 1)	
26	'nominal =	=((B23+1)^(1/B25)-1)*B25	
27			
28	=NOMINAL(0.053543, 4)		

Figure 1: A sheet-defined function implementing Excel's `NOMINAL` built-in. Cells B23 and B24 are input cells, cell B26 is the output cell, and B25 holds an intermediate result. The call to `DEFINE` in cell A22 creates the function. Cell A28 contains a call to the defined function. It takes around 200 ns to execute it, of which 80 ns is due to exponentiation (^). As shown in cell A28, a sheet-defined function is called just like a built-in or VBA function.

2. THE VISION

The ultimate goal of this work is to allow spreadsheet users themselves to develop and evolve libraries of user-defined functions to support sophisticated spreadsheet models. Defining a function requires only well-known spreadsheet concepts such as cell, cell reference and function, and no external programming languages. Therefore experimentation and adaptation of user-defined functions remain under the control of the spreadsheet users and domain experts, who need not wait for an IT department to understand, describe, implement and test the desired changes.

Any spreadsheet computation can be turned into a sheet-defined function. This ensures conceptual and notational simplicity. Moreover, it means that new user-defined functions may arise by refactoring of a spreadsheet model as it evolves. As a spreadsheet model becomes more refined and complex, it may be observed that the same cluster of formulas appears again and again. Such a cluster of formulas may then be encapsulated in a sheet-defined function, and each formula cluster replaced by a call to that function. This both simplifies the spreadsheet model and improves its

Table 1: Time to compute the cumulative distribution function of the normal distribution $N(0,1)$.

Implementation	Time/call (ns)
Sheet-defined function	118
C#	47
C (gcc 4.2.1 -O3)	51
Excel 2007 VBA function	1925
Excel 2007 built-in `NORMSDIST`	993

maintainability, because a bug-fix or other improvement to the function will automatically affect all its uses, unlike the traditional situation when there are multiple copies of the same cluster of formulas.

Sheet-defined functions may be shared with other users in the same department or application domain, without preventing them from making their own improvements — because the domain knowledge is not locked into the notation of a "real" programming language, but one that presumably is familiar to users and that they are (more) comfortable experimenting with.

Sheet-defined functions support end-user "tinkering" to develop models and workflows that are appropriate within their application domain [7]. Clearly not all spreadsheet users will be equally competent developers of sheet-defined functions, and clearly not all software should be developed in this way. However, judging from the huge popularity of spreadsheets within banks, finance, management, science and engineering, the immediate response and the user control offered by spreadsheets are attractive features. Also, from anecdotal evidence, structured use of spreadsheets is a flexible, fast and cheap alternative to "big bang" professional IT projects.

3. THE FUNCALC PROTOTYPE

We have created a prototype implementation of sheet-defined functions, called Funcalc. The implementation is written in C#, is quite compact (12,000 lines of code) and compiles sheet-defined functions to .NET bytecode [3] at run-time. As shown by Table 1 execution efficiency is very good; this is due both to local optimizations performed by our function compiler and to Microsoft's considerable engineering effort in the underlying .NET just-in-time compiler.

Funcalc features include:

- a "normal" interpretive spreadsheet implementation;
- a compiled implementation of sheet-defined functions;
- recursive functions and higher-order functions;
- functions can accept and return array values in addition to numbers and string;
- automatic specialization, or partial evaluation [12];
- facilities for benchmarking sheet-defined functions.

Because Funcalc supports higher-order functions, the value contained in a cell, say A42, may be a function value. This value may be called as `APPLY(A42,0.053543,4)` using built-in function `APPLY`.

Table 2: Time to call a square root function; includes recalculation time.

Calling	Time/call (ns)
Sheet-defined function from Funcalc	400
Excel built-in from Excel	160
.NET function from Excel/Excel-DNA	4,900
VBA function from Excel	12,000

Function values are built by applying a sheet-defined function to only some of its arguments, the absent arguments being given as `NA()`; the resulting function value will display as `NOMINAL(#NA,4)` or similar.

Such a function value may be specialized, or partially evaluated, with respect to its available (non-`#NA`) arguments. The result is a new function value with the same behavior but potentially better performance because the available argument values have been inlined and loops unrolled in the underlying bytecode. For more information, see [5] and [12]. Specialization provides some amount of incremental computation and memoization, and we do not currently have other general mechanisms for these purposes.

A forthcoming book [13] gives many more details of the implementation, more examples of sheet-defined functions, and a manual for Funcalc. A previously published paper [15] presents a case study of reimplementing Excel's built-in financial functions as sheet-defined functions.

A comprehensive list of US spreadsheet patents is given in a forthcoming report [14].

4. INTEGRATION WITH EXCEL

In ongoing work [10] we integrate sheet-defined functions with the widely used Microsoft Excel spreadsheet program, rather than our Funcalc prototype, as illustrated in Figures 2 and 3. This enables large-scale experimentation with sheet-defined functions because they can be defined in a context that is familiar to spreadsheet users and provides charting, auditing, and so on.

The main downside is that calling a sheet-defined function from Excel is much slower than from the Funcalc implementation (yet apparently faster than calling a VBA function); see Table 2. However, the sheet-defined function itself will execute at the same speed as in Funcalc. This work uses the Excel-DNA runtime bridge between Excel and .NET [4].

5. FUTURE WORK

So far we have focused mostly on functionality and good performance. We emphasize performance because we want sheet-defined functions to replace not only user-defined functions written in VBA, C++ and other external languages, but to replace built-in functions also. Domain experts in finance, statistics and other areas of rather general interest should be able to develop well-performing high-quality functions themselves and not have to rely on Microsoft or other vendors to do so.

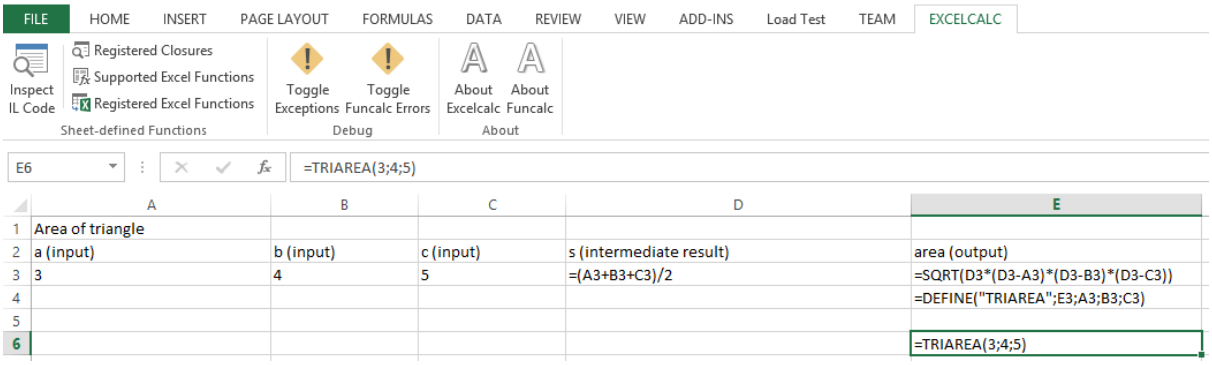


Figure 2: Funcalc as Excel plug-in, showing formulas of sheet-defined function `TRIAREA` with input cells A3, B3 and C3, intermediate cell D3, and output cell E3. The call to `DEFINE` in cell E4 creates the function. Through the new "Excelcalc" menu one can interact with the underlying Funcalc implementation and the Excel-Funcalc bridge (mostly for development purposes).

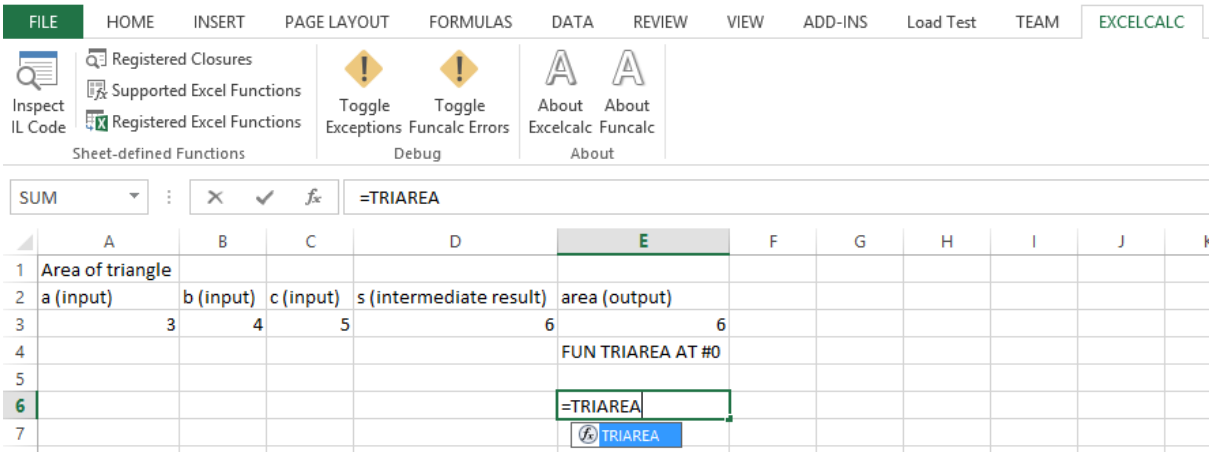


Figure 3: Same sheet as in Figure 2, here showing values rather than formulas. Note the editing in progress of a call to sheet-defined function `TRIAREA` in cell E6.

However, a well-performing implementation of sheet-defined functions is just the beginning: one should investigate additional infrastructure and practices to support their use. For instance, how can we extend the natural collaboration around spreadsheet development [8] in a community of users to cover also libraries of sheet-defined functions; how can we support versioning and merging of such libraries in a way that does not preclude individual users' tinkering and experimentation; how can we support systematic testing; and so on.

Our concept of sheet-defined functions should be subjected to a systematic usability study; the study conducted by Peyton-Jones, Blackwell and Burnett [9] assumed that functions could not be recursive, whereas ours can.

Finally, sheet-defined functions lend themselves well to parallelization, because they are pure (yet strict, an unusual combination) so that computations can be reordered and performed speculatively, and often exhibit considerable explicit parallelism. In fact, they resemble dataflow languages such as Sisal [6]. Presumably some of the 1980es techniques used to schedule dataflow languages [11] could be used to perform spreadsheet computations efficiently on modern multicore machines. The result might be "supercomputing for the masses", realizing Chandy's 1984 vision [2].

6. CONCLUSION

We have presented a prototype implementation of sheet-defined functions and outlined some future work. Our hope is that such functionality will become available in widely used spreadsheet programs, or via a full-featured version of the plugin described in Section 4, and will enable spreadsheet users to develop their own computational models into reusable function libraries, without loss of computational efficiency and without handing over control to remote IT departments or software contractors. Moreover, there seems to be a technological opportunity to harness the power of multicore machines through spreadsheet programming.

7. REFERENCES

[1] L. Augustsson, H. Mansell, and G. Sittampalam. Paradise: A two-stage DSL embedded in Haskell. In *International Conference on Functional Programming (ICFP'08)*, pages 225–228. ACM, September 2008.

[2] M. Chandy. Concurrent programming for the masses. (PODC 1984 invited address). In *Principles of Distributed Computing 1985*, pages 1–12. ACM, 1985.

[3] Ecma TC39 TG3. *Common Language Infrastructure (CLI). Standard ECMA-335*. Ecma International, sixth edition, June 2012.

[4] Excel DNA Project. Homepage. At http://exceldna.codeplex.com/ on 28 February 2014.

[5] N. D. Jones, C. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993. At http://www.itu.dk/people/sestoft/pebook/pebook.html on 9 June 2013.

[6] J. McGraw et al. Sisal. Streams and iteration in a single assignment language. Language reference manual, version 1.2. Technical report, Lawrence Livermore National Labs, March 1985.

[7] B. A. Nardi *A small matter of programming. Perspectives on end user programming*. MIT Press, 1993.

[8] B. A. Nardi and J. R. Miller Twinkling lights and nested loops: distributed problem solving and spreadsheet development. *International Journal of Man-Machine Studies*, 34:161–184, 1991.

[9] S. Peyton Jones, A. Blackwell, and M. Burnett. A user-centred approach to functions in Excel. In *ICFP '03: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, pages 165–176. ACM, 2003.

[10] J. D. Rask and S. E. Timmermann. Integration of sheet-defined functions in Excel using C#. Master's thesis, IT University of Copenhagen, 2014. (Expected June 2014).

[11] V. Sarkar and J. Hennessy. Compile-time partitioning and scheduling of parallel programs. In *ACM SIGPLAN '86 Symposium on Compiler Construction*, pages 17–26, June 1986.

[12] P. Sestoft. Online partial evaluation of sheet-defined functions. In A. Banerjee, O. Danvy, K. Doh, and J. Hatcliff, editors, *Semantics, Abstract Interpretation, and Reasoning about Programs*, volume 129 of *Electronic Proceedings in Theoretical Computer Science*, pages 136–160, 2013.

[13] P. Sestoft. *Spreadsheet Implementation Technology. Basics and Extensions*. MIT Press, 2014. ISBN 978-0-262-52664-7. (Expected August 2014). 313 pages.

[14] P. Sestoft. Spreadsheet patents. Technical Report ITU-TR-2014-178, IT University of Copenhagen, 2014. ISBN 978-87-7949-317-9. (To appear).

[15] P. Sestoft and J. Z. Sørensen. Sheet-defined functions: implementation and initial evaluation. In Y. Dittrich et al., editors, *International Symposium on End-User Development, June 2013*, volume 7897 of *Lecture Notes in Computer Science*, pages 88–103, 2013.

Dependence Tracing Techniques for Spreadsheets: An Investigation

Sohon Roy
Delft University of Technology
S.Roy-1@tudelft.nl

Felienne Hermans
Delft University of Technology
F.F.J.Hermans@tudelft.nl

ABSTRACT
Spreadsheet cells contain data but also may contain formulas that refer to data from other cells, perform operations on them, and render the results directly to show it to the user. In order to understand the structure of spreadsheets, one needs to understand the formulas that control cell-to-cell dataflow. Understanding this cell-to-cell inter-relation or **dependence tracing** is easier done in visual manners and therefore quite a few techniques have been proposed over the years. This paper aims to report the results of an investigative study of such techniques. The study is a first step of an attempt to evaluate the relevance of these techniques from the point of view of their benefits and effectiveness in the context of real world spreadsheet users. Results obtained from such a study will have the potential for motivating the conception of newer and better techniques, in case it is found that the need for them is still not fully catered.

Categories and Subject Descriptors
H.4.1 [**Information Systems Applications**]: Office Automation – *Spreadsheets*

General Terms
Design, Experimentation, Human Factors

Keywords
End-user computing, Dependence tracing, Spreadsheet visualizations

1. INTRODUCTION

1.1 Background
Spreadsheets offer the end-users an interface that is incomparable in its simplicity and flexibility. However it is mostly beneficial for performing rapid calculations and quick simple analyses. This interface is not helpful at all in understanding the design logic behind a spreadsheet, especially the type of understanding that is necessary in order to make modifications to existing spreadsheets. Modification becomes harder in the case where it is done by a user different from the creator. This situation is fairly common in the industry as the average lifespan of spreadsheets have been found to be 5 years [3] which can often prove too long for the possibility that the original creator will be always available whenever some modifications are required. When understanding spreadsheets, the visual structure that is perceived from just looking at the cells is referred to as spreadsheet *surface structure* [2] comparable to the anatomical structure of the human body. However calculations are performed based on formulas and the formulas connect the cells to form another kind of structure called the *computational/deep structure* that is comparable to the nervous system of the human body. These two structures are often not similar and at times can be radically different. The deep structure reflects the data flow in the spreadsheet and is basically the cell-to-cell inter-dependence. In the understanding of a spreadsheet, this cell-to-cell inter-dependence plays a key role. Without having a clear idea of cell-to-cell inter-dependence, the modification of a fairly complex spreadsheet becomes impossible without ample risks of errors. It is considerably easier to understand for a user if the referred cell(s) in a formula are indicated in an enhanced manner with visualization techniques, instead of having to manually inspect each and every formula and trying to locate the exact cell(s) that it is referring to. Therefore a number of visualization techniques have been proposed in various research papers over the years. However there are some questions about these techniques that still need to be explored and they form the core of our investigation. They are listed in subsection 1.3.

1.2 Motivation
It is our opinion that visualization based dependence tracing techniques, as found in research literature, are not making across to the industry of spreadsheet users. In a study conducted by Hermans *et al*. [3] with spreadsheet users working in a large Dutch financial company, it was found that "*the most important information needs of professional spreadsheet users concern the structure of the formula dependencies*". This study also mentions the feeling of inadequacy felt by the users while using the only available dependence tracing tool within their reach the Excel Audit toolbar [Fig.1]; a feature of MS (Microsoft) Excel which is by far the most popular [1] spreadsheet application in the market. This feature demonstrates cell inter-dependencies with an overlaid dependency graph over a worksheet, with graph edges shown as blue arrows; the edges however are generated on a cell-by-cell basis which has to be interactively activated by the user. Findings of another informal survey conducted in October 2013 at the offices of the UK based financial modeling company F1F9[1] also point repeatedly at the direction of the sense of inadequacy the spreadsheet users are suffering from when depending heavily on this Excel Audit tracing feature. These findings lead us to the question why there are no better tools available to spreadsheet users? Nevertheless, as will be shown in this paper, there is considerable amount of research already done on this topic. This gives rise to the question why implementations of such research are not making it to the industry? Only a handful of highly

[1] F1F9: A financial modeling company http://www.f1f9.com/

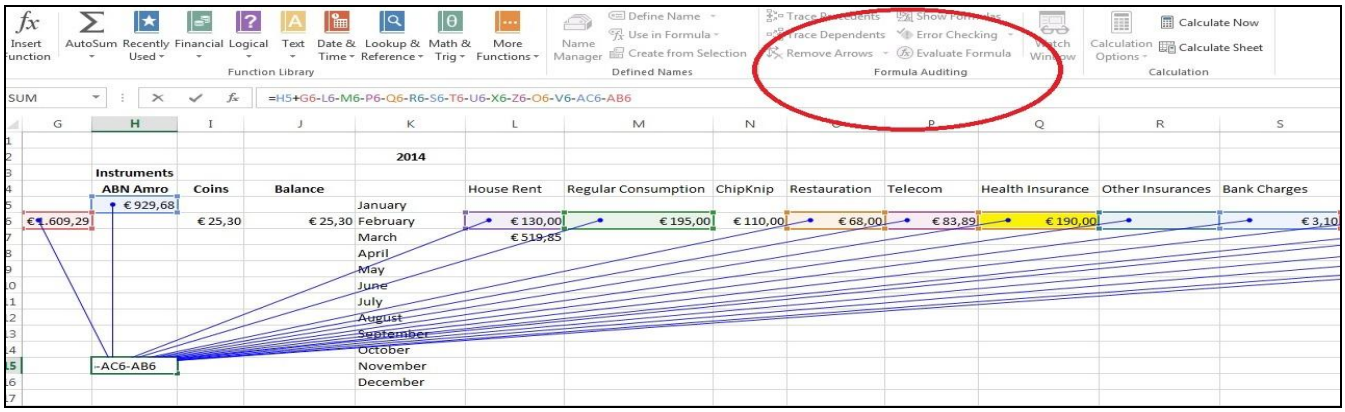


Figure 1: Tracing dependents with Excel Audit toolbar: blue connecting arrows and coloring of precedent cells

customized tools are existing today and that also are mostly used internally by organizations; they are not compared against each other based on any well accepted metrics framework. Their efficacy in actually helping in the end-user experience is not measured. Our investigation is therefore dedicated to evaluating the effectiveness of these proposed techniques in the context of real world spreadsheet users. Such an evaluation might also open up specific areas in which to improve upon or come up with newer techniques that are not only innovative but viable in terms of practically realizable implementations that can be adopted by spreadsheet users in the industry.

1.3 Hypothesis and Research Questions

Hypothesis: Proposals thus far described and demonstrated in research literature about visualization based techniques for spreadsheet dependence tracing have not adequately made it across to the industry in forms of reliable, user-friendly, wide-spread, multi-platform, and standardized software tools of both stand-alone and plug-in type.

On the basis of the premise established in Subsection 1.2 and the above mentioned hypothesis, we arrive at the following three research questions.

Research Questions:

R1. **Why the proposals thus far described and demonstrated in research literature have not reached the industry as implementations?**

An attempt to study what may be the key causes of the perceived bottleneck between research and industrial implementations.

R2. **Is there any well-accepted metrics framework with which such implementations as above (R1) can be compared to each other?**

If and when implementations are made available to the industry, it is necessary to measure their usefulness in actually helping the end-user computing experience. If such a framework is not there, then it can be devised and made into an industrial standard.

R3. **Is there any well-defined opportunity for improvement in the dependence tracing context**?

Improvement not just from the aspect of innovativeness of idea but also from the angle of how well the idea can be translated into a user-friendly and reliable implementation; the efficacy being measured against metrics as mentioned in R2.

1.4 Approach

To ascertain answers to the research questions, as a first step, we did a critical review of the existing research literature on this specific topic of visualization based dependence tracing techniques for spreadsheets. This paper summarizes in brief the findings of the review and the conclusions drawn from it. It essentially presents preliminary results and indicators related to the research questions. In order to illustrate our findings for this paper, we chose a number of research papers relevant on this topic and revisited their contents from the following aspects:

I. The basic technique/principle/strategy

II. Characteristic features related to dependents tracing

III. Tools or prototypes developed if any

IV. Comments or details available on testing, performance, and limitations

V. Current status of the research and its implementation, and its perceived relevance or influence in the industrial scene

2. THE SELECTED RESEARCH PAPERS

2.1 Fluid Visualization of Spreadsheet Structures [4]

In this paper Igarashi *et al*. provide the description of a spreadsheet visualization technique mainly based on superimposition of visual enhancement and animations on top of the regular tabular structure of spreadsheets. The strategy is primarily the use of graphical variation (color, shading, outlining, etc.), animation, and lightweight interaction that allows the user to directly perceive the spreadsheet dataflow structure, keeping the tabular spreadsheet view unchanged. The *transient local view* feature is a visual enhancement based on outlining and shading that allows a user to view the dataflow associated with a particular cell. There is a *static global view* that visually enhances the entire spreadsheet by overlaying the complete dataflow graph of all the cells. *Animated global explanation* plays an animation to illustrate the dataflow of the entire spreadsheet. *Visual editing techniques* is a graphical manipulation technique that allows the user to directly edit the generated dataflow graph in *global static view* by dragging and its effect is then reflected in the spreadsheet structure as the textual formulas are updated automatically. A prototype for UNIX was developed using Pad++ and Python. Pad++ was a visualization platform developed and maintained by University of Maryland. A video demonstration of the tool in

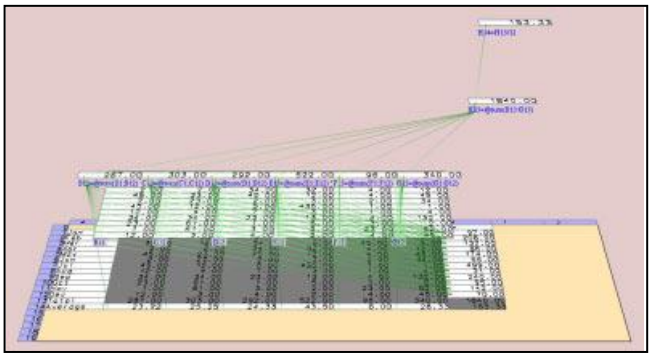


Figure 2: Recursive lifting-up operation

action is available. It is mentioned that the smoothness of animation is limited to spreadsheets of 400 cells[2] or lesser. Performance of the tool radically degrades with increase in size of the spreadsheets. There is no information if the efficacy of the prototype was tested with real spreadsheet users. No future plan is provided on how this tool can be implemented or scaled up for use in the industry of spreadsheet users. Pad++ and its support has been long discontinued and the project is closed by UMD. However, an extension of the idea of "transient local view" as proposed in this paper can be observed in MS Excel version 2007 onwards. In Excel 2007 the precedent cells of a cell are outlined in different colors. In Excel 2013 the precedent cells are actually shaded fully in different colors [Fig.1].

2.2 3D Interactive Visualization for Inter-cell Dependencies of Spreadsheets [5]

In this paper Shiozawa *et al*. propose a technique of cell dependence visualization in 3D based on an interactive lifting up operation. The technique utilizes the fact that spreadsheets are two dimensional tabular structures and therefore the third dimension can be used to depict complementary information like cell inter-dependencies. A spreadsheet is first graphically re-rendered in a 3D space. Next, users are allowed to select a cell and drag it upwards level-wise along the z-axis. The selected cell's dependent cells are pointed with arrows [Fig.2] and they themselves are also lifted up but kept one level below the selected cell. However in this case the advantage is in the fact that unlike in Excel, arrows connecting dependent cells lying on the same row would never overlap with each other to generate visual ambiguity. The lifting up operation is recursively repeated on the dependent cells as well to generate a leveled tree structure in 3D. This provides the user a clear idea of which cells in the sheet are more important by looking at the levels of dependents lying below them. A prototype for UNIX was developed by modifying the spreadsheet program SLSC. The 3D graphics were implemented with OpenGL APIs. No information regarding the performance of the prototype is provided. For an application such as this, making heavy use of computer graphics, it is presumable that performance and scaling could be a concern. Unfortunately the paper does not throw any light on this matter. Neither was given any detail about how beneficial or acceptable the tool proved for spreadsheet users.

2.3 Visual Checking of Spreadsheets [2]

In this paper Chen *et al*. propose a set of strategies aimed at checking and debugging of spreadsheets using visual methods to reveal the *deep structure* of spreadsheets to the users. A set of visual methods is described followed by strategies on how to best use those visual tools for different purposes of checking. The *functional identification* feature demarcates cells with different colors according to whether they behave as input, output, processing or standalone and this classification is based on whether a cell is having dependents, precedents, both or none. *Multi-precedents and dependents tool*, *block-precedents tool*, and the *in-block-precedents-dependents tool* are all tools that illustrate various types of inter-cell dependencies with pointed arrow-heads similar to the Excel feature. The difference here being that arrows not only connect individual cells but also have the capability of offering the visual perception that they are connecting a set of related cells that are visually grouped together by shading or coloring; such group of cells are termed in the paper as *cell block*. Three debugging strategies each for global and local context were described to illustrate the use of these tools. The tools were implemented using VBA (Visual Basic for Applications) and authors claimed that they can be plugged in to any Excel installation. In spite of claims that the tools increase usability of spreadsheets, no details were given about user acceptance or any measurement of by how much they increased usability.

2.4 Spreadsheet Visualisation[3] to Improve End User Understanding [1]

In this paper Ballinger *et al*. provide description of a visualization toolkit that could ease understanding of spreadsheets by introducing visual abstraction with types of images that emphasize on layout and dependency rather than values of cells. In order to achieve this, their idea was to extract all the information contained in a spreadsheet and utilize that in a more versatile programming environment to quickly generate visualizations. They chose Java for this purpose and since Excel is the most popular spreadsheet application, their toolkit was designed to operate on

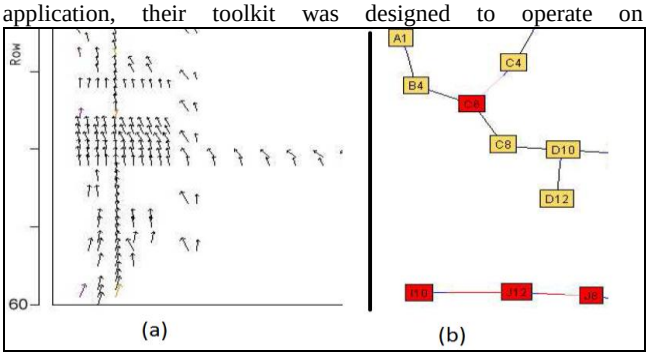


Figure 3: (a) Data dependency unit vector map (b) Spring view graph structure

Excel spreadsheets. The toolkit is capable of extracting low level structural information and data from spreadsheet files, analyze that information, and produce visualization. The *data dependency flow* feature is capable of generating 2D and 3D maps that illustrate the general drift of dataflow in a spreadsheet with arrows of unit magnitude [Fig.3 (a)]. This helps reduce the visual clutter which normally occurs with arrows of different lengths due to different distances between cells. The *graph structure* feature provides the *spring view* [Fig.3 (b)] which is a generated graph of cells stripped of their values. The *detailed inspection of formula* feature provides visualizations that are similar to Excel Audit and

[2] This is a much smaller number of cells than what is observed in typical real life spreadsheets

[3] Paper is in New Zealand English

block precedents tool (subsection 2.3) but they are not overlaid on spreadsheets; the images are generated on spreadsheet-like matrix structures and the cells are reduced to row-column intersection points, their values wiped out to reduce visual overhead on the user's understanding. The toolkit was run successfully on a corpus of 259 workbooks. User-studies were not conducted and no details were given on whether real users found it convenient enough to understand the various types of images.

2.5 Supporting Professional Spreadsheet Users by Generating Leveled Dataflow Diagrams [3]

In this paper Hermans *et al.* propose a spreadsheet visualization technique and the description of an implementation along with the findings of a user study. The work in this paper extends that of previous work by the authors about extraction of class diagrams from spreadsheets. The basic principle depends upon classifying all cells in a spreadsheet as either of type *data, formula, label,* or *empty.* Diagrams similar to ER (Entity-Relationship) diagrams are next created by representing data cells as entities and formula cells as method (operation) + entity (result). The interconnections are illustrated as relationships. Next these elements are grouped together based on the presence of *label* type cells to form larger entities that represent cell blocks. These are then assembled

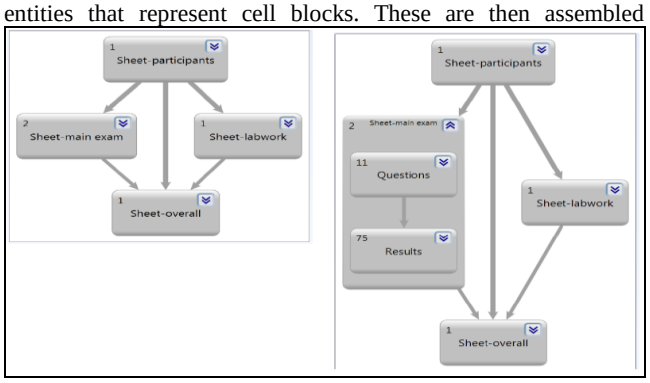


Figure 4: Global view (L) and Worksheet view (R)

inside entities that represent their respective worksheets. In this manner the hierarchical leveled dataflow diagrams are generated. The *global view* [Fig.4] feature offers the users a high level interactive visualization of the whole workbook showing the dependencies between worksheets. The *worksheet view* shows the dependencies between blocks in the same sheet and the low level *formula view* shows in details how individual cells are inter-connected via formulas. A tool was developed called GyroSAT (Gyro Spreadsheet Analysis Toolkit) in C# 4.0. The output dataflow diagram is produced in DGML (Directed Graph Markup Language) which can be viewed and navigated in Microsoft Visual Studio 2010 Ultimate's built-in DGML browser. This tool was extensively evaluated with a user group consisting of 27 professional spreadsheet users working in a large Dutch financial management company. A set of 9 spreadsheets that were used for testing in 9 case studies had number of worksheets ranging from 4 to 42, and number of cells ranging from 1048 to 503050. Subsequently this tool and its features have been integrated into the set of services offered by the spreadsheet solutions company Infotron.[4]

[4] Infotron is a spreadsheet solution company offering web based spreadsheet analysis services http://www.infotron.nl/

3. CONCLUSIONS

Our study indicates that each of the five research papers proposes unique and innovative visualization techniques based on different strategies. All of them offer rich set of features intended to help spreadsheet users from different angles. Only two of them have prototypes running on UNIX, both of which, to the best of our beliefs will prove incompatible for current use on any popular platform. One has Excel based VBA implementation which supposedly should work as plug-in to any Excel version but is subject to be tested against version incompatibility. Two of them have full-fledged standalone implementations based on Java and C#, both accepting Excel spreadsheets as inputs, but only one of them has found practical exposure in the industry. This reinforces the need to explore our research question "**R1**. **Why the proposals thus far described and demonstrated in research literature have not reached the industry as implementations?"**

Only one of the research ideas has been properly validated against a set of real world professional spreadsheet users. The efficacies of the rest of the research ideas have only been claimed in writing but not demonstrated by user studies. This further reinforces the need to explore our second research question "**R2**. **Is there any well-accepted metrics framework with which such implementations as above (R1) can be compared to each other?"**

The above findings also lead us towards the general conclusion that our third research question "**R3. Is there any well-defined opportunity for improvement in the dependence tracing context**?" is an open question indeed. In that light we therefore judge that a suitable next step would be to do a more exhaustive search of available spreadsheet visualization tools and 1) actually test them on industrially used spreadsheets such as those available in the EUSES corpus and if the tools are found to be performing in a reliable manner then 2) test them on an adequately large and well represented spreadsheet users group to measure usability.

4. REFERENCES

[1] Ballinger, D., Biddle, R., Noble, J. 2003. Spreadsheet Visualisation to Improve End-user Understanding. In proceedings of the Asia-Pacific Symposium on Information Visualisation - Volume 24 (APVIS 2003), Adelaide, Australia, pp. 99–109.

[2] Chen, Y., Chan, H. C. 2000. Visual Checking of Spreadsheets. In proceedings of the European Spreadsheet Risks Interest Group 1st Annual Conference (EuSpRIG 2000), London, United Kingdom.

[3] Hermans, F., Pinzger, M., Deursen, A. van. 2011. Supporting Professional Spreadsheet Users by Generating Leveled Dataflow Diagrams. In proceedings of the 33rd International Conference on Software Engineering (ICSE 2011), Waikiki, Honolulu, HI, USA, pp. 451–460.

[4] Igarashi, T., Mackinlay, J., Chang, B.-W., Zellweger, P. 1998. Fluid Visualization of Spreadsheet Structures. In proceedings of the IEEE Symposium on Visual Languages (VL 1998), Halifax, NS, Canada, pp. 118–125.

[5] Shiozawa, H., Okada, K., Matsushita, Y. 1999. 3D Interactive Visualization for Inter-Cell Dependencies of Spreadsheets. In proceedings of the IEEE Symposium on Information Visualization (Info Vis 1999), San Francisco, CA, USA, pp. 79–82, 148.

MDSheet – Model-Driven Spreadsheets

Jácome Cunha João Paulo Fernandes Jorge Mendes
Rui Pereira João Saraiva
{jacome,jpaulo,jorgemendes,ruipereira,jas}@di.uminho.pt
HASLab/INESC TEC & Universidade do Minho, Portugal
CIICESI, ESTGF, Instituto Politécnico do Porto, Portugal
RELEASE, Universidade da Beira Interior, Portugal

ABSTRACT

This paper showcases *MDSheet*, a framework aimed at improving the engineering of spreadsheets. This framework is model-driven, and has been fully integrated under a spreadsheet system. Also, its practical interest has been demonstrated by several empirical studies.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Spreadsheets; D.2.0 [**Software Engineering**]: General; D.2.6 [**Software Engineering**]: Programming Environments—*Graphical environments, Integrated environments, Interactive environments*

General Terms

Languages, Design, Human Factors

Keywords

Model-Driven Spreadsheets, MDSheet, Model Inference, Embedding, Bidirectional Synchronization, Querying

1. INTRODUCTION

We can not run the modern world without spreadsheets. Spreadsheets are omnipresent, from individuals needing to cope with simple needs to large companies needing to implement complex forecasts or to produce advanced reports.

The realization of such importance has made concrete impact in the scientific community as well. This is due to more research teams devoting their efforts to improving spreadsheets, and a growing number of scientific events dedicated to them.

A successful approach to address spreadsheets under a scientific perspective consists of incorporating well-established software engineering techniques in the spreadsheet development process.

Our approach is essentially based on precisely one such technique: we adopt model-driven spreadsheet engineering. In the setting we propose a spreadsheet is abstracted through a concise model, which is then used to improve effectiveness and efficiency of spreadsheet users. The framework we describe in this paper has been realized in a traditional spreadsheet development system, thus not forcing spreadsheet users to move to a different paradigm.

The spreadsheet development framework that we envision has been fully incorporated in a tool, *MDSheet*[1], whose features include:[2]

1) Model inference: we extract the abstract representation from legacy spreadsheets;
2) Embedded models: this abstract representation is manipulated and evolved in spreadsheets themselves;
3) User guidance: relying on this business model, we are able of guiding users in avoiding traditional spreadsheet mistakes;
4) Model/instance synchronization: we support the evolution of model and instances, ensuring an automatic synchronization of the unevolved artifact;
5) Model quality assessment: a set of metrics on the complexity of a spreadsheet model can be computed;
6) Querying: spreadsheet data can be queried.

2. SPREADSHEET ENGINEERING

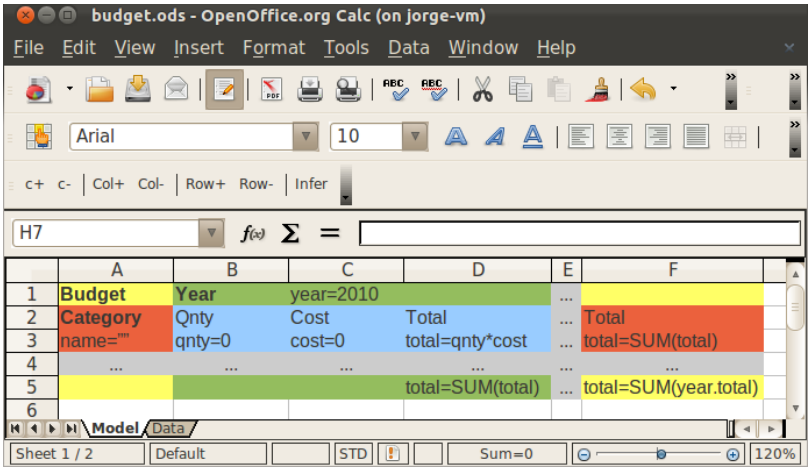
MDSheet is a framework for the engineering of spreadsheets in a model-driven fashion. This framework is highly extensible: we have actually extended it with several new functionalities that we have developed in the last few years.

2.1 Motivational Example

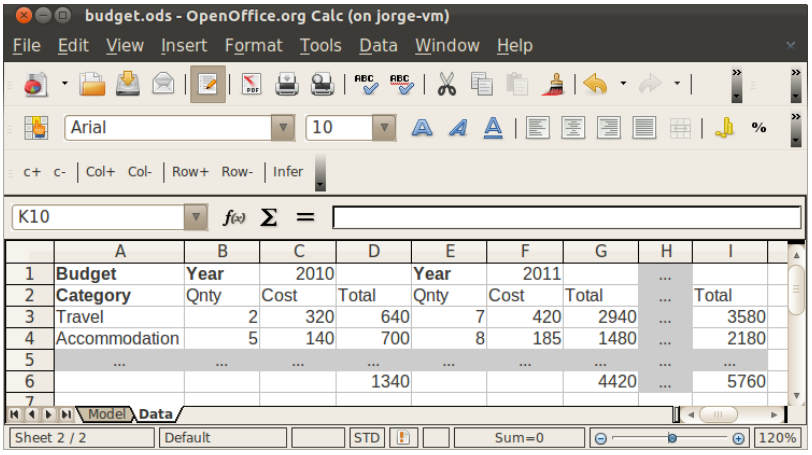
The realization of our approach to spreadsheet engineering builds upon the embedding of ClassSheets in a spreadsheet system. So, we start by introducing ClassSheets within *MDSheet* with the example given in Figure 1: we present a model for a **Budget** spreadsheet (Figure 1a), which we adapted from [13][3], and an instance of such model (Figure 1b).

This model holds three classes where data is to be inserted by end users: *i)* **Year**, with a default value of 2010, for the budget to accommodate multi-year information, *ii)* **Category**, for assigning a label to each expense and *iii)*, a(n implicit) relationship class where quantity and costs are registered and totals are calculated based on them. The actual spreadsheet may hold several repetitions of any of these elements, as indicated by the ellipsis. For each expense we record its quantity and its cost (with 0 as the default value), and we calculate the total amount associated with it. Finally, (simple) summation formulas are used to calculate the global amount spent per year (cell D5), the amount spent per expense type in all years (cell F3) and the total amount spent in all years (cell F5) are also calculated.

[1] *MDSheet* is available through the SSaaPP project website: `http://ssaapp.di.uminho.pt`.
[2] In the next section, we describe each such feature in a different subsection.
[3] We assume colors are visible in the digital version of this paper.



(a) Model worksheet.



(b) Data/instance worksheet.

Figure 1: A bidirectional model-driven environment for a budget spreadsheet.

Following is the description of the full set of features offered by *MDSheet*.

2.2 Model Inference

A model-driven approach to spreadsheet engineering offers an improved development experience: an abstract representation of a spreadsheet, i.e., its model, helps us, among other things, in guiding users into preventing errors. This approach, however, requires the definition of a model in parallel with the spreadsheet it abstracts. In order to handle legacy spreadsheets, i.e., the ones that have not been developed from scratch together with their model, we have devised a model inference technique [2], that has been implemented in *MDSheet*. Concretely, we infer models in the ClassSheets language, an object-oriented high-level formalism to abstract spreadsheets [13].

2.3 Embedded Models

The worksheet structure of spreadsheets is a decisive factor in their modularity. In fact, we exploited precisely this structure to make the model of a spreadsheet available within spreadsheet systems themselves: one worksheet holds the model of a spreadsheet, while another holds its data. This embedding of spreadsheets has also been implemented under *MDSheet* [6], which was demonstrated in Section 2.1. Moreover, we extended the ClassSheets language with database constraints, such as unique columns/rows or foreign keys, which have also been incorporated in *MDSheet* [11]. In fact, we have further extended the available restrictions so that the user can specify the contents of a cell using regular expressions or intervals [8]. Finally, we extended ClassSheets with references between different models making them more flexible. Note that through this embedding we can guarantee that spreadsheet data always conforms to a model.

2.4 User Guidance

The embedding of our extended version of the ClassSheet language allows us to guide the user in inserting correct data. When a model is designed, it serves as a guider in the creation of a data worksheet, which is initially empty. Only cells containing plain data can be edited as all other are inferred from the model. This prevents, e.g., users from making mistakes when defining formulas as they are locked. Moreover, the restrictions created in the model guarantee that the data in the cells respects them. In the model it is possible to define an interval of integers for a cell, or a regu-

lar expression that the content must conform to. A column or row can be marked as having only unique values or being a foreign key to another column or row. All these restrictions are enforced by *MDSheet*. In the case of foreign keys, the user can use a combo box to select existing values from the referred column/row.

2.5 Model/Instance Synchronization

As any other software artifact, spreadsheets evolve over time. *MDSheet* accommodates changes by allowing the evolution of models and instances, while automatically coevolving the unchanged artifact. For this, we introduced a formal framework to allow evolutions of the model to be automatically spanned to the instances [6, 7, 12]. We have later proposed techniques and tools to the evolution of data by the user and corresponding automatic coevolution of the model [3]. We therefore ensure that model/instance consistency is never broken.

2.6 Model Quality Assessment

In a first attempt to measure the quality of a spreadsheet model, we introduced a set of metrics to calculate the complexity of ClassSheet models [9]. These metrics are implemented under *MDSheet* and can be calculated for any ClassSheet defined using it. They are then compared to the same metrics computed for a repository of ClassSheet models so users can have a reference point for such values. The evolution mechanisms can then be used to evolve the spreadsheet improving it according to the metrics calculated.

2.7 Querying

As many spreadsheets are used as data repositories, the need to query their data is frequent. *MDSheet* also integrates a query system, which allows the definition of model-oriented queries, in the style of traditional database queries. This allows the writing of queries without having to manually observe a possibly large number of columns and rows of concrete data. Indeed, queries are written, by analyzing models, as abstractions that are simpler to understand. Our system was initially presented as a textual language [1, 4], very similar to SQL. Even being textual it already was of great help for users [14]. Still, we have further improved it by embedding the language in a worksheet, thus creating a visual language for spreadsheet querying [5].

3. EMPIRICAL VALIDATION

One of the purposes of our tool is to help users commit less errors; if possible, it also intends to help users work faster with spreadsheets. To assess these two concerns we have run an empirical study and we have found empirical evidence that indeed our model-driven spreadsheet environment can in fact help users become more efficient and more effective [10].

4. CONCLUSION

We briefly presented *MDSheet* and all the features it offers to its users. Given the fact that it has been built as a framework, new tools, even if not proposed by us, can easily be integrated in it.

We believe this tool is in a very mature state and can be used in real case scenarios. We have thus started its integration in industry: i), to support test case evolution in an agile testing framework of a software house; ii), to adapt data produced by different database systems for a car multimedia production company; and iii), to provide spreadsheet models for a food bank.

Acknowledgments

This work is part funded by the ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project `FCOMP-01-0124-FEDER-020532`. The first author was funded by the FCT grant `SFRH/BPD/73358/2010`.

5. REFERENCES

[1] O. Belo, J. Cunha, J. P. Fernandes, J. Mendes, R. Pereira, and J. Saraiva. Querysheet: A bidirectional query environment for model-driven spreadsheets. In *VLHCC '13*, pages 199–200. IEEE CS, 2013.

[2] J. Cunha, M. Erwig, and J. Saraiva. Automatically inferring classsheet models from spreadsheets. In *VLHCC '10*, pages 93–100. IEEE CS, 2010.

[3] J. Cunha, J. P. Fernandes, J. Mendes, H. Pacheco, and J. Saraiva. Bidirectional transformation of model-driven spreadsheets. In *ICMT '12*, volume 7307 of *LNCS*, pages 105–120. Springer, 2012.

[4] J. Cunha, J. P. Fernandes, J. Mendes, R. Pereira, and J. Saraiva. Querying model-driven spreadsheets. In *VLHCC '13*, pages 83–86. IEEE CS, 2013.

[5] J. Cunha, J. P. Fernandes, J. Mendes, R. Pereira, and J. Saraiva. Embedding model-driven spreadsheet queries in spreadsheet systems. In *VLHCC '14*, 2014. to appear.

[6] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. Embedding and evolution of spreadsheet models in spreadsheet systems. In *VLHCC '11*, pages 186–201.

[7] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. MDSheet: A framework for model-driven spreadsheet engineering. In *ICSE 2012*, pages 1412–1415. ACM.

[8] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. Extension and implementation of classsheet models. In *VLHCC '12*, pages 19–22. IEEE CS, 2012.

[9] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. Complexity Metrics for ClassSheet Models. In *ICCSA '13*, volume 7972, pages 459–474. LNCS, 2013.

[10] J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. Embedding, evolution, and validation of spreadsheet models in spreadsheet systems. 2014. submitted.

[11] J. Cunha, J. P. Fernandes, and J. Saraiva. From Relational ClassSheets to UML+OCL. In *SAC '12*, pages 1151–1158. ACM, 2012.

[12] J. Cunha, J. Visser, T. Alves, and J. Saraiva. Type-safe evolution of spreadsheets. In D. Giannakopoulou and F. Orejas, editors, *FASE '11*, volume 6603 of *LNCS*, pages 186–201. Springer, 2011.

[13] G. Engels and M. Erwig. ClassSheets: automatic generation of spreadsheet applications from object-oriented specifications. In *ASE '05*, pages 124–133. ACM, 2005.

[14] R. Pereira. Querying for model-driven spreadsheets. Master's thesis, University of Minho, 2013.

How can we figure out what is inside thousands of spreadsheets?

Thomas Levine
_@thomaslevine.com

ABSTRACT

We have enough data today that we it may not be realistic to understand all of them. In hopes of vaguely understanding these data, I have been developing methods for exploring the contents of large collections of weakly structured spreadsheets. We can get some feel for the contents of these collections by assembling metadata about many spreadsheets and run otherwise typical analyses on the data-about-data; this gives us some understanding patterns in data publishing and a crude understanding of the contents. I have also developed spreadsheet-specific search tools that try to find related spreadsheets based on similarities in implicit schema. By running crude statistics across many disparate datasets, we can learn a lot about unweildy collections of poorly structured data.

Keywords

data management, spreadsheets, open data, search

1. INTRODUCTION

These days, we have more data than we know what to do with. And by "data", we often mean unclean, poorly documented spreadsheets. I started wondering what was in all of these spreadsheets. Addressing my curiosity turned out to be quite difficult, so I've found up developing various approaches to understanding the contents of large collections of weakly structured spreadsheets.

My initial curiosity stemmed from the release of thousands of spreadsheets in government open data initiatives. I wanted to know what they had released so that I may find interesting things in it.

More practically, I often am looking for data from multiple sources that I can connect in relation to a particular topic. For example, in a project I had data about cash flows through the United States treasury and wanted to join them to data about the daily interest rates for United States bonds. In situations like this, I usually need to know the name of the dataset or to ask around until I find the name. I wanted a faster and more systematic approach to this.

2. TYPICAL APPROACHES TO EXPLORING THE CONTENTS OF SPREADSHEETS

Before we discuss my spreadsheet exploration methods, let's discuss some more ordinary methods that I see in common use today.

2.1 Look at every spreadsheet

As a baseline, one approach is to look manually at every cell in many spreadsheets. This takes a long time, but it is feasible in some situations.

2.2 Use standard metaformats

Many groups develop domain-specific metaformats for expressing a very specific sort of data. For example, JSON API is a metaformat for expressing the response of a database query on the web [4], Data Packages is a metaformat for expressing metadata about a dataset [17], and KML is a metaformat for expressing annotations of geographic maps [19].

Agreement on format and metaformat makes it faster and easier to inspect individual files. On the other hand, it does not alleviate the need to acquire lots of different files and to at least glance at them. We spend less time manually inspecting each dataset, but we must still manually inspect lots of dataset.

The same sort of thing happens when data publishers provide graphs of each individual dataset. When we provide some graphs of a dataset rather than simply the standard data file, we are trying to make it easier for people to understand that particular dataset, rather than trying to focus them on a particular subset of datasets.

2.3 Provide good metadata

Data may be easier to find if we catalog our data well and adhere to certain data quality standards. With this reasoning, many "open data" guidelines provide direction as to how a person or organization with lots of datasets might allow other people to use them [16, 1, 18, 13, 15].

At a basic level, these guidelines suggest that data should be available on the internet and under a free license; at the

other end of the spectrum, guidelines suggest that data be in standard formats accompanied with particular metadata.

Datasets can be a joy to work with when these data quality guidelines are followed, but this requires much upfront work by the publishers of the data.

2.4 Asking people

In practice, I find that people learn what's in a spreadsheet through word of mouth, even if the data are already published on the internet in standard formats with good metadata.

Amanda Hickman teaches journalism and keeps a list of data sources for her students [3].

There entire conferences about the contents of newly released datasets, such as the annual meeting of the Association of Public Data Users [14].

The Open Knowledge Foundation [16] and Code for America [2] even conducted data censuses to determine which governments were releasing what data publically on the internet. In each case, volunteers searched the internet and talked to government employees in order to determine whether each dataset was available and to collect certain information about each dataset.

3. ACQUIRING LOTS OF SPREADSHEETS

In order to explore methods for examining thousands of spreadsheets, I needed to find spreadsheets that I could explore.

Many governments and other large organizations publish spreadsheets on data catalog websites. Data catalogs make it kind of easy to get a bunch of spreadsheets all together. The basic approach is this.

1. Download a list of all of the dataset identifiers that are present in the data catalog.
2. Download the metadata document about each dataset.
3. Download data files about each dataset.

I've implemented this for the following data catalog softwares.

- Socrata Open Data Portal
- Common Knowledge Archive Network (CKAN)
- OpenDataSoft

This allows me to get all of the data from most of the open data catalogs I know about.

After I've downloaded spreadsheets and their metadata, I often assemble them into a spreadsheet about spreadsheets [6]. In this super-spreadsheet, each record corresponds to a full sub-spreadsheet; you could say that I am collecting features or statistics about each spreadsheet.

4. CRUDE STATISTICS ABOUT SPREADSHEETS

My first approach was involved running rather crude analyses on this interesting dataset-about-datasets that I had assembled.

4.1 How many datasets

I started out by simply counting how many datasets each catalog website had.

The smaller sites had just a few spreadsheets, and the larger sites had thousands.

4.2 Meaninglessness of the count of datasets

Many organizations report this count of datasets that they publish, and this number turns out to be nearly useless. As illustration of this, let's consider a specific group of spreadsheets. Here are the titles of a few spreadsheets in New York City's open data catalog.

- Math Test Results 2006-2012 - Citywide - Gender
- Math Test Results 2006-2012 - Citywide - Ethnicity
- English Language Arts (ELA) Test Results 2006-2012 - Citywide - SWD
- English Language Arts (ELA) Test Results 2006-2012 - Citywide - ELL
- Math Test Results 2006-2012 - Citywide - SWD
- English Language Arts (ELA) Test Results 2006-2012 - Citywide - All Students
- Math Test Results 2006-2012 - Citywide - ELL
- English Language Arts (ELA) Test Results 2006-2012 - Citywide - Gender
- Math Test Results 2006-2012 - Citywide - All Students
- English Language Arts (ELA) Test Results 2006-2012 - Citywide - Ethnicity

These spreadsheets all had the same column names; they were "grade", "year", "demographic", "number_tested", "mean_scale_score", "num_level_1", "pct_level_1", "num_level_2", "pct_level_2", "num_level_3", "pct_level_3", "num_level_4", "pct_level_4", "num_level_3_and_4", and "pct_level_3_and_4".

These "datasets" can all be thought of as subsets of the same single dataset of test scores.

If I just take different subsets of a single spreadsheet (and optionally pivot/reshape the subsets), I can easily expand one spreadsheet into over 9000. This is why the dataset count figure is near useless.

4.3 Size of the datasets

I can also look at how big they are. It turns out that most of them are pretty small.

- Only 25% of datasets had more than 100 rows.
- Only 12% of datasets had more than 1,000 rows.

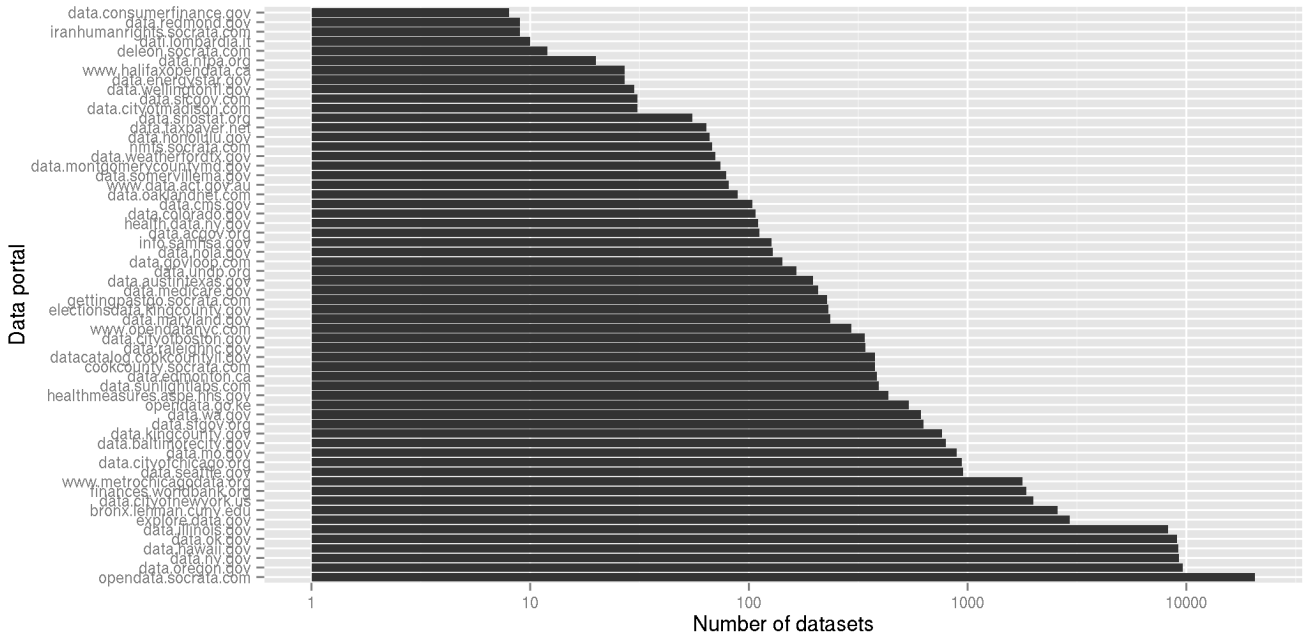


Figure 1: How many datasets (spreadsheets) each data catalog had

- Only 5% of datasets had more than 10,000 rows.

Regardless of the format of these datasets, you can think of them as spreadsheets without code, where columns are variables and rows are records.

5. MEASURING HOW WELL DIFFERENT SPREADSHEETS FOLLOW DATA PUBLISHING GUIDELINES

Having gotten some feel for the contents of these various data catalogs, I started running some less arbitrary statistics. As discussed in section 2.3, many groups have written guidelines as to how data should be published [16, 1, 18, 13, 15]. I started coming up with measures of adherence to these guidelines and running them across all of these datasets.

5.1 File format

File format of datasets can tell us quite a lot about the data. I looked at the MIME types of the full data files for each dataset on catalogs running Socrata software and compared them between data catalogs [11].

If datasets are represented as tables inside the Socrata software, they are available in many formats. If they are uploaded in formats not recognized by Socrata, they are only available in their original format.

I looked at a few data catalogs for which many datasets were presented in their original format. In some cases, the file formats can point out when a group of related files is added at once. For example, the results indicate that the city of San Francisco in 2012 added a bunch of shapefile format datasets to its open data catalog from another San Francicso government website. As another example, most of the datasets in the catalog of the state of Missouri are traffic surveys, saved as PDF files [5].

5.2 Licensing

Many of the data publishing guidelines indicate that datasets should be freely licensed. All of the data catalog websites that I looked at include a metadata field for the license of the dataset, and I looked at the contents of that field. I found that most datasets had no license [10], and this is thought to be detrimental to their ability to be shared and reused [16, 1, 18, 13, 15].

5.3 Liveliness of links

One common guideline is that data be available on the internet. If a dataset shows up in one of these catalogs, you might think that it is on the internet. It turns out that the links to these datasets often do not work.

I tried downloading the full data file for each dataset refenced in any of these catalogs and recorded any errors I received [9, 12]. I found most links to be working and noticed some common reasons why links didn't work.

- Many link URLs were in fact local file paths or links within an intranet.
- Many link "URLs" were badly formed or were not URLs at all.
- Some servers did not have SSL configured properly.
- Some servers took a very long time to respond.

I also discovered that one of the sites with very alive links,

`https://data.gov.uk`, had a "Broken links" tool for identifying these broken links.

6. SEARCHING FOR SPREADSHEETS

While assessing the adherence to various data publishing guidelines, I kept noticing that it's very hard to find spreadsheets that are relevant to a particular analysis unless you already know that the spreadsheet exists.

Major search engines focus on HTML format web pages, and spreadsheet files are often not indexed at all. The various data catalog software programs discussed in section 3 include a search feature, but this feature only works within the particular website. For example, I have to go to the Dutch government's data catalog website in order to search for Dutch data.

To summarize my thoughts about the common means of searching through spreadsheets, I see two main issues. The first issue is that the search is localized to datasets that are published or otherwise managed by a particular entity; it's hard to search for spreadsheets without first identifying a specific publisher or repository. The second issue is that the search method is quite naive; these websites are usually running crude keyword searches.

Having articulated these difficulties in searching for spreadsheets, I started trying to address them.

6.1 Searching across publishers

When I'm looking for spreadsheets, the publishing organization is unlikely to be my main concern. For example, if I'm interested in data about the composition of different pesticides, but I don't really care whether the data were collected by this city government or by that country government.

To address this issue, I made a a disgustingly simple site that forwards your search query to 100 other websites and returns the results to you in a single page [7]. Lots of people use it, and this says something about the inconvenience of having separate search bars for separate websites.

6.2 Spreadsheets-specific search algorithms

The other issue is that our search algorithms don't take advantage of all of the structure that is encoded in a spreadsheet. I started to address this issue by pulling schema-related features out of the spreadsheets (section 4.2).

6.3 Spreadsheets as input to a search

Taking this further, I've been thinking about what it would mean to have a search engine for spreadsheets.

When we search for ordinary written documents, we send words into a search engine and get pages of words back.

What if we could search for spreadsheets by sending spreadsheets into a search engine and getting spreadsheets back? The order of the results would be determined by various specialized statistics; just as we use PageRank to find relevant hypertext documents, we can develop other statistics that help us find relevant spreadsheets.

Figure 2: The search engine for words takes words as input and emits words as output

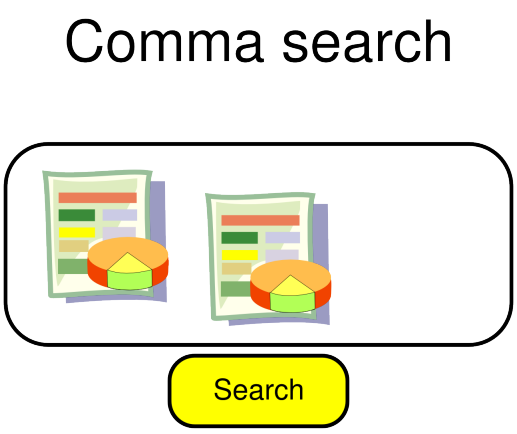


Figure 3: The search engine for spreadsheets takes spreadsheets as input and emits spreadsheets as output

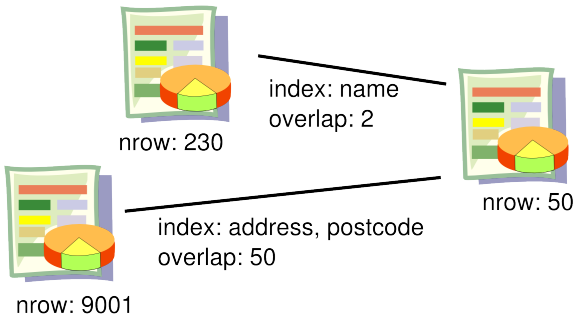


Figure 4: Commasearch infers some schema information about each spreadsheet and looks for other spreadsheets with similar schemas.

6.3.1 *Schema-based searches*

I think a lot about rows and columns. When we define tables in relational databases, we can say reasonably well what each column means, based on names and types, and what a row means, based on unique indices. In spreadsheets, we still have column names, but we don't get everything else.

The unique indices tell us quite a lot; they give us an idea about the observational unit of the table and what other tables we can nicely join or union with that table.

Commasearch [8] is the present state of my spreadsheet search tools. To use comma search, you first index a lot of spreadsheets. Once you have the index, you may search by providing a single spreadsheet as input.

In the indexing phase, spreadsheets are examined do find all combinations of columns that act as unique indices, that is, all combinations of fields whose values are not duplicated within the spreadsheet. In the search phase, comma search finds all combinations of columns in the input spreadsheet and then looks for spreadsheets that are uniquely indexed by these columns. The results are ordered by how much overlap there is between the values of the two spreadsheets.

To say this more colloquially, comma search looks for many-to-one join relationships between disparate datasets.

7. REVIEW

I've been downloading lots of spreadsheets and doing crude, silly things with them. I started out by looking at very simple things like how big they are. I also tried to quantify other people's ideas of how good datasets are, like whether they are freely licensed. In doing this, I have noticed that it's pretty hard to search for spreadsheets; I've been developing approaches for rough detection of implicit schemas and for relating spreadsheets based on these schemas.

8. APPLICATIONS

A couple of people can share a few spreadsheets without any special means, but it gets hard when there are more than a couple people sharing more than a few spreadsheets.

Statistics about adherence to data publishing guidelines can be helpful to those who are tasked with cataloging and maintaining a diverse array of datasets. Data quality statistics can provide a quick and timely summary of the issues with different datasets and allow for a more targeted approach in the maintenance of a data catalog.

New strategies for searching spreadsheets can help us find data that are relevant to a topic within the context of analysis.

9. REFERENCES

[1] T. Berners-Lee. Linked data. `http://www.w3.org/DesignIssues/LinkedData.html`, 2006.
[2] Code for America. *U.S. City Open Data Census*, 2014.
[3] A. Hickman. *Where to Find Data*, 2014.
[4] S. Klabnik and Y. Katz. Json api: A standard for building apis in json. `http://jsonapi.org/`.
[5] T. Levine. *License-free data in Missouri's data portal*, 2013.
[6] T. Levine. Open data had better be data-driven. `http://thomaslevine.com/!/dataset-as-datapoint`, 2013.
[7] T. Levine. *OpenPrism*, 2013.
[8] T. Levine. *commasearch*, 2014.
[9] T. Levine. Dead links on data catalogs. `http://thomaslevine.com/!/data-catalog-dead-links/`, 2014.
[10] T. Levine. Open data licensing. `http://thomaslevine.com/!/open-data-licensing/`, 2014.
[11] T. Levine. What file formats are on the data portals? `http://thomaslevine.com/!/socrata-formats/`, 2014.
[12] T. Levine. Zombie links on data catalogs. `http://thomaslevine.com/!/zombie-links/`, 2014.
[13] C. Malamud, T. O'Reilly, G. Elin, M. Sifry, A. Holovaty, D. X. O'Neil, M. Migurski, S. Allen, J. Tauberer, L. Lessig, D. Newman, J. Geraci, E. Bender, T. Steinberg, D. Moore, D. Shaw, J. Needham, J. Hardi, E. Zuckerman, G. Palmer, J. Taylor, B. Horowitz, Z. Exley, K. Fogel, M. Dale, J. L. Hall, M. Hofmann, D. Orban, W. Fitzpatrick, and A. Swartz. 8 principles of open government data. `http://www.opengovdata.org/home/8principles`, 2007. Open Government Working Group.
[14] A. of Public Data Users. *Association of Public Data Users Annual Conference*, 2013.
[15] Open Data Institute. *Certificates*, 2013.
[16] Open Knowledge Foundation. *Open Data Census*, 2013.
[17] R. Pollock, M. Brett, and M. Keegan. Data packages. `http://dataprotocols.org/data-packages/`, 2013.
[18] Sunlight Foundation. *Open Data Policy Guidelines*, 2014.
[19] T. Wilson. OgcÂő kml. Technical Report OGC 07-147r2, Open Geospatial Consortium Inc., 2008. `http://portal.opengeospatial.org/files/?artifact_id=27810`.

Sheetmusic: Making music from spreadsheets

Thomas Levine
csv soundsystem
_@thomaslevine.com

ABSTRACT

The spreadsheet provides an intuitive paradigm for the expression of musical scores. Musical scores can be expressed as data tables, with each record corresponding to a place in time and each column corresponding to a note or instrument. Sheetmusic is a plugin for Gnumeric that provides music sequencing spreadsheet functions. Tools like Sheetmusic provide intuitive music composition interfaces for people who are used to data analysis software. Moreover, they help us plot data with the sense of sound.

Keywords

music, spreadsheets, gastronomification, data analysis

1. NON-VISUAL DISPLAYS OF QUANTITATIVE INFORMATION

Data analyists often use visualization as a means for plotting data, but there are other approaches!

1.1 Data sonification

Just as data can be expressed visually, data can also be expressed in sound. As demonstration of this, Ferguson & al. [4] created auditory analogs for simple visual plots, such as the dotplot and boxplot.

Visual plots are far more common than auditory plots. Why is this? My hunch is that our technology for visual rendering is simply much further advanced; printing technology has been around for centuries, and writing has been around for millenia. With this history, we have also developed advanced theory related to the visual plotting of data. Audio recording is a comparably recent invention, and our theory around auditory plotting is accordingly less developed.

In my view, we separate data sonification from data visualization only because of technological constraints; there isn't a fundamental difference between these two processes.

1.2 Data-driven music videos

Combining the visual and auditory senses, we can plot data in the form of music videos. One example of this is the FMS Symphony (figure 1). In the FMS Symphony, each beat of music corresponds to a business day during the period between 2005 and 2013, the pitch of one instrument corresponds to the United States interest rate, the pitch of another instrument corresponds to the distance to the United States debt ceiling, and the activation of certain flourishes corresponds to changes in the balance of the United States treasury. These data are also represented visually, through the combination of an animated line plot and a Chernoff face [1].

1.3 Food

Why stop at just vision and hearing? We can plot data as food and use all five senses. One example of this is Census Spices, a set of spices that represent different neighborhoods based on demographics collected by the United States Census [5].

2. OUR TOOLS FOR MUSICAL PLOTTING

We at csv soundsystem have been exploring multisensory data plotting methods, including music videos and food. In our production of data-driven music videos, we have recognized a need for data analysis software and music software to be more strongly integrated. We wanted a more seamless transition between modeling and music, and we wanted it to be easier for data analysts to work with music. We have developed tools like Sheetmusic to bridge this gap.

To use the language of the Grammar of Graphics, [8] we have abstract data and concrete plot elements, and we define aesthetics that provide mappings between the abstract data and the concrete elements. The primitive plot elements that we use for music are things like key, rhythm, pitch, and interval.

2.1 Data tables

We've found that the tabular representation of data aligns very well with typical representations of music. Our data music tools work by mapping these two concepts to each other.

We can think of data tables as collections of similar things, with the same sorts of information being collected about each thing. In tidy data tables [7] each row corresponds to an observation (a thing), and each column corresponds to a

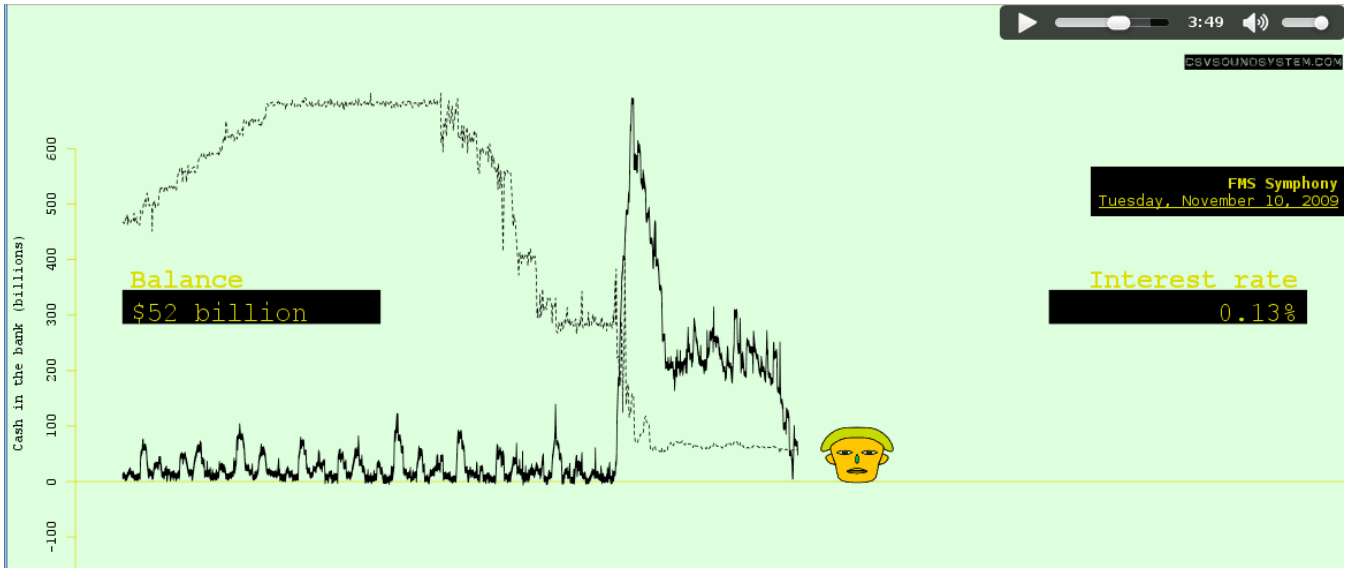


Figure 1: Here is a frame from the FMS Symphony video. I unfortunately can't play the accompanying song in this paper.

variable. We add more rows to the table as we observe more things, and we add more columns to the table as we collect more information about each thing.

We can think of music as a composition of many different sounds over time, with sounds coming from many different instruments. In musical scores we represent time as movement from left to right, and we represent different notes played at the same time by different dots on a staff, The staff becomes wider as the song gets longer, (They are often spread across multiple pages.) and we add more dots as we add more notes (figure 2).

Rather than composing music as traditional sheet music, we can use a table-editing program of our choice to compose this sort of table. Our data music software simply adds musical functions to table containers in various data analysis tools.

Sheetmusic is our offering for spreadsheets, but we also have libraries for R data frames [3] and Pandas data frames [2].

3. HOW TO USE SHEETMUSIC

Let's divide Sheetmusic's features into two groups. The first group is spreadsheet functions for music synthesis—these are functions like `CHORD_PROGRESSION` that take spreadsheet cells or values as input and return values to other spreadsheet cells. The second group is functions for rendering the music to external devices, including MIDI and sheetmusic.

3.1 Organization of the spreadsheet

Sheetmusic expects that the spreadsheet be organized as follows. Each column corresponds to a musical track, and different tracks can have different music instruments. Row corresponds to a beat (of time). Each cell contains the frequency of sound to be played, represented in scientific notation (C4, D4, &c.).

3.2 Composing music

The data analyst can use conventional spreadsheet approaches for composing music. For example, the following function can be used to produce a major scale in a spreadsheet column.

```
=IONIAN_SCALE("A4")
```

Once you have a major scale in one column, you can easily make chords with a spreadsheet functions like this.

```
=MAJOR_THIRD_INTERVAL(B1)
```

If you put this in cell B1, A1 and B1 will form a major third interval.

3.3 Rendering music

Once we have composed our piece, we can select the appropriate cells, specify the key and time signatures of the piece, and export it as MIDI or sheetmusic.

It is possible, of course, to convert to any number of music formats, just as we can convert spreadsheets to any number of data table formats. Only MIDI and sheetmusic are implemented at present, but you can indirectly convert to many formats by first saving as MIDI and then converting from MIDI to your output format of choice.

3.4 Musical plots

I've discussed how we can use Sheetmusic for conventional music composition. To use it as a plotting tool, we simply have to map our abstract data to musical notes. Sheetmusic provides the `FROMWHOLENUMBER` function to enable this. If we imagine an infinitely wide piano with the C0 as the left-most note, `FROMWHOLENUMBER` starts at C0 and walks i keys to the right, where i is the argument passed to `FROMWHOLENUMBER`.

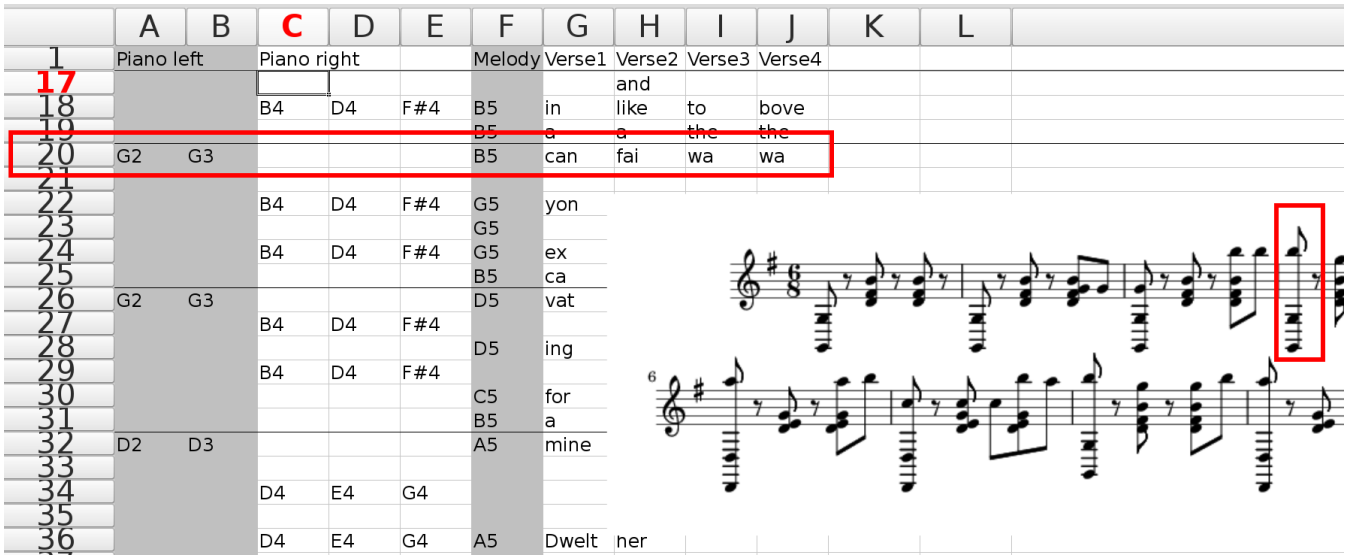


Figure 2: A spreadsheet is displayed alongside some corresponding ordinary sheet music, with a corresponding row/beat highlighted.

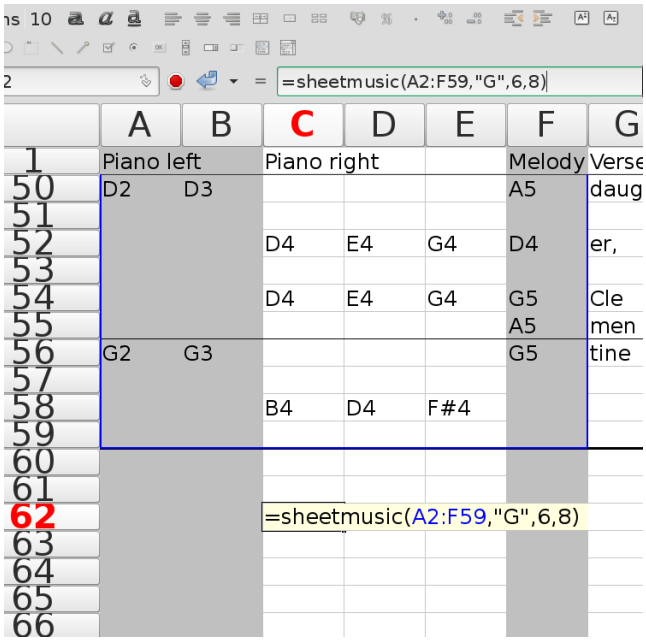


Figure 3: Using Sheetmusic to render a spreadsheet as sheetmusic

After using ordinary spreadsheet modeling functions to manipulate data, a user may scale and round the data appropriately and then run `FROMWHOLENUMBER` to convert them into notes.

4. RELEVANCE

I hope that I've shown how data can be plotted in the form of music. I would be remiss not to discuss the merits of this plotting method.

4.1 Easier composition of music

When we plot data as music, we effectively let data compose music for us. We still have to choose datasets that will produce interesting music and map the data to the music appropriately, but the randomness of the data can provide the various subtleties of music that we would otherwise have to design ourselves.

4.2 Data literacy

When we start using data analysis software for other things, we blur the line between data analysis and other things. Data analysis seems very magical to many people. When we represent data as familiar things like music, people seem to be a bit less scared of data analysis.

4.3 Expressing high-dimensional datasets

The use of multiple senses may also allow for the expression of high-dimensional datasets. Tufte advocates for the production of visuals that express the multivariate nature of the world.[6] I think that the use of multiple senses has the potential to facilitate the expression of more easily express many variables at once, and this may aid in the identification of high-dimensional relationships.

5. REFERENCES

[1] B. Abelson, J. Bialar, B. DeWilde, M. Keller, T. Levine, and C. Podkul. *FMS Symphony*, 2013.

[2] csv soundsystem. *Data music for big data analysis*, 2013.

[3] csv soundsystem. *ddr: Data-driven Rhythms in R*, 2013.

[4] S. Ferguson, W. Martens, and D. Cabrera. *Statistical Sonification for Exploratory Data Analysis*.

[5] H. Kang-Brown. *Making Census Data Taste Like New York City*, 2013.

[6] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, USA, 1986.

[7] H. Wickham. Tidy data. `http://vita.had.co.nz/papers/tidy-data.pdf`.

[8] L. Wilkinson. *The Grammar of Graphics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Are We Overconfident in Our Understanding of Overconfidence?

Raymond R. Panko
Shidler College of Business
University of Hawai`i
2404 Maile Way
Honolulu, HI 96821
001.808.377.1149
Ray@Panko.com

ABSTRACT

In spreadsheet error research, there is a Grand Paradox. Although many studies have looked at spreadsheet errors, and have found, without exception, has error rates that are unacceptable in organizations, organizations continue to ignore spreadsheet risks. They do not see the need to apply software engineering disciplines long seen to be necessary in software development, in which error types and rates are similar to those in spreadsheet development..[1] Traditionally, this Great Paradox had been attributed to overconfidence. This paper introduces other possible approaches for understanding the Grand Paradox. It focuses on risk blindness, which is our unawareness of errors when they occur.

Categories and Subject Descriptors

K.8.1: Spreadsheets. D.2.5 Testing and Debugging.

General Terms

Experimentation, Verification.

Keywords

Methodology. Spreadsheet Experiments, Experiments, Inspection. Sampling, Statistics

1. INTRODUCTION

Despite overwhelming and unanimous evidence that spreadsheet errors are widespread and material, companies have continued to ignore spreadsheet error risks. In the past, this Great Paradox had been attributed to overconfidence. Human beings are overconfident in most things, from driving skills to their ability to create large error-free spreadsheets. In one of the earliest spreadsheet experiments, Brown and Gould [1] noted that developers were extremely confident in their spreadsheets' accuracy, although every participant made at least one undetected error during the development process. Later experimenters also remarked on overconfidence. Panko conducted an experiment to see if feedback would reduce overconfidence, as has been the case in some general overconfidence studies. The study found a statistically significant reduce in confidence and error rates, but the error rate reduction was minimal. Goo performed another experiment to see if feedback could reduce overconfidence and errors. There was some reduction in overconfidence but no statistical reduction in errors.

2. RISK BLINDNESS IN BEHAVIORAL STUDIES

This paper introduces other possible approaches for understanding the Grand Paradox. It focuses on risk blindness, which is our unawareness of errors when they occur.

Naatanen and Summala [9] first articulated the idea that humans are largely blind to risks. Expanding on this idea, Howarth [5] studied drivers who approached children wanting to cross at an intersection. Fewer than 10% of drivers took action, and those actions would have come too late if the children had started crossing the street. Svenson [14] studied drivers approaching blind bends in a road. Unfamiliar drivers slowed down. Familiar drivers did not, approaching at speeds that would have made accident avoidance impossible.

Fuller [2] suggested that risk blindness in experienced people stems from something like operant conditioning. If we speed in a dangerous area, we get to our destination faster. This positive feedback reinforces risky speeding behavior. In spreadsheet development, developers who do not do comprehensive error checking finish faster and avoid onerous testing work. In contrast, negative reinforcement in the form of accidents is uncertain and rare.

Even near misses may reinforce risky behavior rather than to reduce it. In a simulation study of ship handling, Habberley, Shaddick, and Taylor [4] observed that skilled watch officers consistently came hazardously close to other vessels. In addition, when risky behavior required error-avoiding actions, watch officers experienced a gain in confidence in their "skills" because they had successfully avoided accidents. Similarly, in spreadsheet development, if we catch some errors as we work, we may believe that we are skilled in catching errors and so have no need for formal post-development testing.

Another possible explanation comes from modern cognitive/ neuroscience. Although we see comparatively little of what is in front of us well and pay attention to much less, our brain's constructed reality gives us the illusion what we see what is in front of us clearly [11]. To cope with limited cognitive processing power, the CR construction process includes the editing of anything irrelevant to the constructed vision. Part of this is not making us aware of the many errors we make [11]. Error editing makes sense

for optimal performance, but it means that humans have very poor intuition about the error rates and ability to avoid errors [11]. For the CR process this is an acceptable tradeoff, but it makes us confident that what we are doing works well.

Another explanation from cognitive/neuroscience is System 1 thinking, which has been discussed in depth by Kahneman [7]. System 1 thinking uses parallel processing to generate conclusions it is fast and easy, but its working are opaque. If we are walking down a street and a dog on a leash snaps at us, we jump. This is fast or System 1 thinking. It is very effective and dominates nearly all of our actions, but it has drawbacks. First, it gives no indication that it may be wrong. Unless we actively turn on slow System 2 thinking, which we cannot do all the time, we will accept System 1 suggestions uncritically. One problem with doing so is that System 1 thinking, when faced with an impossible or at least very difficult task, may solve a simpler task and make a decision on that basis. For instance, if you are told that a bat and ball cost a dollar and ten cents and that the bat costs a dollar more than the ball, a typical System 1 thought response is that the ball costs ten cents. This is wrong, of course, but System 1 thinking tends to solve the simpler problem, $1.10 - $1.00. If we do not force ourselves to engage in slow and odious System 2 thinking, we are likely to accept the System 1 alternative problem solution.

This may be why, when developers are asked whether a spreadsheet they have just completed has errors, they quickly say no, on the basis of something other than reasoned risk. Reithel, Nichols, and Robinson [13] had participants look at a small poorly formatted spreadsheet, a small nicely formatted spreadsheet, a large poorly formatted spreadsheet, and a large nicely formatted spreadsheet. Participants rated their confidence in the four spreadsheets. Confidence was modest for three of the four spreadsheets. It was much higher for the large well-formatted spreadsheet. Logically, this does not make sense. Larger spreadsheets are more likely to have errors than smaller spreadsheets. This sounds like System 1 alternative problem solving.

3. CONCLUSION

If we are to address the Great Paradox successfully and convince organizations and individuals that they need to create spreadsheets more carefully, we must understand its causes so that we can be persuasive. Beyond that, we must address the Spreadsheet Software Engineering Paradox—that computer scientists and information systems researchers have focused on spreadsheet creation aspects of software engineering, largely ignoring the importance and complexity of testing after the development of modules, functional units, and complete spreadsheets. In software engineering, it accepted that reducing errors during development is good but never gets close to success. Commercial software developers spend 30% to 50% of their development resources on testing [6,8], and this does not count rework costs after errors are found. Yet spreadsheet engineering discussions typically downplay or completely ignore this five-ton elephant in the room. It may be that spreadsheets are simply newer than software development, but spreadsheets have been use for a generation, and strong evidence of error risks have been around almost that long.

We have only looked at the situation at the individual level. Testing must be accepted by groups and even corporations. Even at the group level, this paper has not explored such theories as the diffusion of innovations. If spreadsheet testing is mandated, that will reduce risks. However, user developers must have the freedom to explore their problem spaces freely by modifying their spreadsheets as their understanding grows. Testing methods must reflect the real process of software development.

4. REFERENCES

[1] Brown, P. S. and Gould, J. D. 1987. An experimental study of people creating spreadsheets. *ACM Transactions on Office Information Systems.* 5, 3 (Nov. 1987), 258-272.

[2] Fuller, R. 1990. Learning to make errors: evidence from a driving simulation. *Ergonomics*, 33, 10/11 (Oct/Nov, 1993), 1241-1250.

[3] Goo, Justin M. W. 2002. *The effect of feedback on confidence calibration in spreadsheet development.* Doctoral Dissertation, University of Hawaii.

[4] Habberley, J. S., Shaddick, C. A., and Taylor, D. H. 1986. *A behavioural study of the collision avoidance task in bridge watchkeeping.* College of Marine Studies, Southampton, England. Cited in Reason (1990).

[5] Howarth, C. I. 1990. The relationship between objective risk, subjective risk, and behavior. *Ergonomics, 31,* 527-535. Cited in Wagenaar & Reason, 1990.

[6] Jones, T. C. 1998. *Estimating software costs*. McGraw-Hill, New York, NY.

[7] Kahneman, D. 2011. *Thinking, fast or slow*. Farrar, Strauss and Giroux, New York, NY.

[8] Kimberland, K. 2004. *Microsoft's pilot of TSP yields dramatic results*, news@sei, No. 2. http://www.sei.cmu.edu/news-at-sei/.

[9] Naatanen, R. and Summala, H. 1976. *Road user behavior and traffic accidents*. North-Holland, Amsterdam. Cited in Wagenaar & Reason, 1990.

[10] Panko, R. R. 2007. Two experiments in reducing overconfidence in spreadsheet development. *Journal of Organizational and End User Computing*, 19, 1 (January–March 2007), 1-23.

[11] Panko, R. R. 2013. The cognitive science of spreadsheet errors: Why thinking is bad. *Proceedings of the 46th Hawaii International Conference on System Sciences (Maui, Hawaii*, January 7-10, 2013).

[12] Reason, J. 1990. *Human error*. Cambridge University Press, Cambridge, England.

[13] Reithel, B. J., Nichols, D. L., and Robinson, R. K. 1996. An experimental investigation of the effects of size, format, and errors on spreadsheet reliability perception. *Journal of Computer Information Systems*, 54-64.

[14] Svensen, O. 1977. *Risks of road transportation from a psychological perspective: A pilot study*. Report 3-77, Project Risk Generation and Risk Assessment in a Social Perspective, Committee for Future-Oriented Research, Stockholm, Sweden, 1977. Cited in Fuller, 1990.

[15] Wagenaar, W. A. and Reason, J. T. 1990. Types and tokens in road accident causation. *Ergonomics, 33,* 10/11 (Nov. 1993), 1365-1375.

Anonymizing Spreadsheet Data and Metadata with AnonymousXL

Joeri van Veen
Infotron
Delft, the Netherlands
joeri@infotron.nl

Felienne Hermans
Delft University of Technology
Delft, the Netherlands
f.f.j.hermans@tudelft.nl

ABSTRACT

In spreadsheet risk analysis, we often encounter spreadsheets that are confidential. This might hinder adoption of spreadsheet analysis tools, especially web-based ones, as users do not want to have their confidential spreadsheets analyzed. To address this problem, we have developed AnonymousXL, an Excel plugin that makes spreadsheets anonymous with two actions: 1) remove all sensitive metadata and 2) obfuscate all spreadsheet data within the Excel worksheets such that it resembles, untraceably, the original values.

1. INTRODUCTION

When commercializing our Breviz analysis toolkit [2, 3, 4] as an online tool called PerfectXL, we ran into the problem that customers often do not want to upload, share or even show us confidential spreadsheets. Therefore, we have developed a tool that obfuscates [1] both the data and the metadata in a spreadsheet, while the values still resemble the original ones. By construction, we guarantee that our anonymization does not create or resolve Excel errors. This enables us to run our smell detection tool on the anonymized spreadsheets as if we were analyzing the original. This paper describes the capabilities, limitations and applications of AnonymousXL.

2. METADATA REMOVAL

AnonymousXL removes spreadsheet metadata: the author, the date the file was last opened and the total edit time, in order to remove any ties with the company that the spreadsheet originally came from. In addition, worksheet names within the spreadsheet are replaced with anonymous names.

2.1 Numerical and Date Related Metadata

All numerical metadata information is converted to 0. At this time, only the numerical metadata "revision number" and "total editing time" are converted. All metadata that is of a date type is set to the day of anonymization: "last print date", "creation date" and "last save time".

2.2 Textual Metadata

The following textual metadata are set to the text string "anonymous": title, subject, author, keywords, comments, template, last author, application name, security, category, format, manager, company.

3. DATA OBFUSCATION

Data obfuscation is the alteration of data to make it anonymous. This happens linearly, from the first sheet to the last sheet, from the first to the last cell of the used range of cells in each worksheet. We use different techniques for different types of data in the spreadsheet: numeric data, dates, textual data, formulas and other types of data.

3.1 Numeric Data

The basic step for anonymizing a number is to randomly add or subtract up to 60% of its original value. Or, mathematically, for any number N in a cell, N is replaced by $N \pm N \times 0.6 \times r$ where r is a random value in the range [0, 1]. We treat integers and real numbers differently: Integer values remain integer, real numbers keep their decimals.

There is one exception in the anonymization: In PerfectXL, one of the analyses that is performed is the occurrence of so-called 'magic numbers', numbers of which the meaning might be unclear to the user. There are some numbers, however, that are not considered to be magical, because of their frequent occurrence: 0, 1, 2, 12, 100, 365, 1000. Therefore, these numbers remain as is in our anonymization process. Since all text fields (including column names) get changed, we believe that leaving the non-magic numbers intact does not pose a threat to the anonymity of the spreadsheet, since labels give numbers semantics.

3.2 Dates

Dates are converted into random dates in the range of representable dates in VBA, in contrast with metadata, in which all date values are set to the day of anonymization. This randomness is introduced as to maintain data variation.

3.3 Textual Data

For textual data, it does not suffice to simply change all textual values to "text", since in many situations, it matters to keep equal strings equal. An example of such a situation is a pivot table, as shown in Figure 1. Should we change all categories to "text", the spreadsheet would not work any-

more, as pivot tables cannot contain two fields of the same name. If we would replace all textual values by unique ones, such as "text1", "text2", "text3", as shown in Figure 2, it does work, pivot tables however are often based on textual data (which denote categories, for example). This means that where there once were three categories ("a", "b" and "c", in Figure 1), now there are many (eight different ones in Figure 2). Pivot tables calculate their size based on the number of unique values they find for a category, so pivot tables become larger than they were originally. This can lead to problems, since multiple pivot tables are often situated close to each other on the same worksheet. If the pivot tables grow because of the anonymization, they can start to overlap and unfortunately, this causes Excel to crash.

	A	B	C	D	E
1	Category	Count		Row Labels	Sum of Count
2	b	20		a	53
3	b	41		b	144
4	a	29		c	179
5	b	53		Grand Total	376
6	c	88			
7	a	24			
8	c	91			
9	b	30			

Figure 1: Original spreadsheet with three categories

	A	B	C	D	E
1	Category	Count		Row Labels	Sum of Count
2	text1	20		text1	20
3	text2	41		text2	41
4	text3	29		text3	29
5	text4	53		text4	53
6	text5	88		text5	88
7	text6	24		text6	24
8	text7	91		text7	91
9	text8	30		text8	30
10				Grand Total	376

Figure 2: Simple text replacement

	A	B	C	D	E
1	unique 1	unique 2		unique 3	Sum of unique 2
2	unique 5	28		unique 5	102
3	unique 5	48		unique 6	52
4	unique 6	18		unique 7	151
5	unique 5	1		unique 8	305
6	unique 7	143			
7	unique 6	34			
8	unique 7	8			
9	unique 5	25			

Figure 3: AnonymousXL applied to table

Therefore, we anonymize all textual values while keeping intact cell uniqueness by replacing texts with "unique1", "unique2", "unique3", etc. (for example, "unique6" represents the textual value "a" in Figure 3).

3.4 Formulas

Formulas are basially left alone. The only modification made to formulas are sheet references, since sheet names are made anonymous as well.

3.5 Other Types

Other data types usually fall under either categories mentioned (for instance, a currency type is simply considered a number). A special note on booleans TRUE and FALSE: as booleans are interpreted by Excel as 0 and 1, they are not changed. However, booleans are seldom present as literal values. They are often the result of formulas, in which case they only change in accordance with modifications to the data they depend on.

4. INTRODUCING EXCEL ERRORS

By changing data in Excel cells, errors might be induced that were not present in the original spreadsheet. For instance, in the formula =A1/(3-A2), division by zero might occur (and thus be reported after analysis) if A2 becomes 3, which could happen because of the anonymization step in which data in cells is decreased or increased by 60% of their original value.

To resolve this, we save the list of all formulas that result in an error before the anonymization. Then, after we anonymize each data cell, we verify that we have not changed this list. For this, we do not have to analyze all formulas in the spreadsheet, we only analyze the recursive precedents of the cell, plus all formulas that contain the INDIRECT function.

5. LIMITATIONS

5.1 Confidential formulas

Every so often, spreadsheets contain confidential formulas. All formulas, including those confidential ones, are left unaltered to preserve analysis results. This might not be sufficient for some users.

5.2 Embedded constants

In the current implementation, we only change numeric values in cells and not within formulas, such as in =SUM(A1:A10)*1.2. This is a limitation because these constants too can be of importance to the spreadsheet owner and thus confidential.

5.3 Analysis Types

Different kinds of spreadsheet analyses scan for different kinds of patterns. Developed to complement PerfectXL, AnonymousXL leaves intact formulas, boolean literals and certain numbers for they are key to mimicking analysis of the original spreadsheet. Nevertheless, AnonymousXL or a slight variation of it could carry great potential for alternative analysis types.

6. REFERENCES

[1] D. E. Bakken, R. Parameswaran, D. M. Blough, A. A. Franz, and T. J. Palmer. Data obfuscation: Anonymity and desensitization of usable data sets. *IEEE Security & Privacy*, 2(6):34–41, 2004.

[2] F. Hermans, M. Pinzger, and A. van Deursen. Supporting professional spreadsheet users by generating leveled dataflow diagrams. In *Proc. of ICSE '11*, pages 451–460, 2011.

[3] F. Hermans, M. Pinzger, and A. van Deursen. Detecting and visualizing inter-worksheet smells in spreadsheets. In *Proc of ICSE '12*, pages 441–451, 2012.

[4] F. Hermans, M. Pinzger, and A. van Deursen. Detecting code smells in spreadsheet formulas. In *Proc of ICSM '12*, pages 409–418, 2012.

Using a Visual Language to Create Better Spreadsheets

Bas Jansen
Delft University of Technology
b.jansen@tudelft.nl

Felienne Hermans
Delft University of Technology
f.f.j.hermans@tudelft.nl

ABSTRACT

It is known that spreadsheets are error-prone. It is very difficult for users to get an overview of the design of the spreadsheet, and this is causing errors. Furthermore users are not always aware of the best way to structure a spreadsheet and just start modeling. To address this we will build a visual language to develop spreadsheet models. This enables users to visualize the design of their spreadsheets. A spreadsheet generator will create the spreadsheet based on the specifications made with our visual language. During this process, best practices for structuring spreadsheets are automatically incorporated. There will be a bidirectional link between the model and the associated spreadsheet.

1. INTRODUCTION

Spreadsheets are extensively used by companies. Information embedded in these spreadsheets often forms the basis for business decisions [7]. The quality of these spreadsheets impacts the quality of the decisions. It is known that spreadsheets are error-prone [9]. A poorly structured spreadsheet is often the cause of errors. Because of the nature of the spreadsheet user interface it is difficult to keep an overview of the underlying design. And without a clear design the structure of the spreadsheet gets messy. Also users do not always possess the knowledge to structure a spreadsheet properly.

In order to address this, we present a research plan to develop an alternative user interface that enables the user to design a spreadsheet using a visual language. Based on the instructions made with this language the spreadsheet is generated automatically. The transformation between the model represented in the visual language and the automatically generated spreadsheet is bidirectional. Changes in the model are propagated to the spreadsheet and vice versa. The visual language will help the user to keep an overall overview of the spreadsheet design. And even more important, because the spreadsheet is generated automatically, well-known design patterns can be incorporated to structure the spreadsheet.

2. PROBLEM DEFINITION

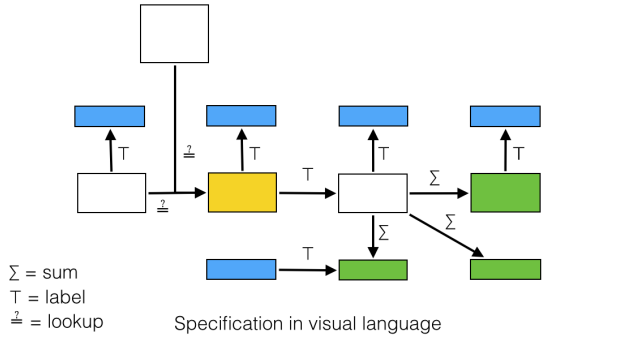
The success of spreadsheets can be partially explained by their easy-to-use interface. However, it is this same interface that is responsible for some of the problems that are associated with spreadsheets. If you write a document or a program you can scroll up and down to get an overall overview of the object you are creating. Also you can use tools (like an outline view, table of contents generator, or dependency graph) within the software to get a better overview. In Excel users can easily enter data and formulas in cells, but as soon as they hit the enter-button a spreadsheet will only show the result of the formula and not the formula itself. In a formula, references are made to other cells in the spreadsheet. It is not possible to immediately see these references. You could say the underlying design of the spreadsheet is hidden 'behind' the spreadsheet itself. The hidden design makes it difficult to understand a spreadsheet and to get an overview of the design. This is causing errors [8].

A part of the design of a spreadsheet is the way the information is structured within the sheet. There are best practices for the structure of a spreadsheet that can be found in literature. A commonly found model is to split input, model and output. This works quite well for some of the problems that people want to solve with spreadsheets. However, it is not the best model in every situation. Spreadsheets are also often used for what-if questions: "What is going to happen if I change this?" In this situation it would be better to have input and output closely together. Putting input and output closely together will lead to a completely different structure of the spreadsheet than implementation of the input, model, output principle. The optimal way to structure a spreadsheet depends on the kind of problem that you want to solve with the spreadsheet.

In practice, users are not making a conscious choice of how they structure their spreadsheets. They want to solve a problem as quickly as possible and just start entering the data and formulas without giving the structure a lot of thought. When the complexity of spreadsheets increases over time they end up with a messy model. At that point it is difficult to change the underlying structure and the risk of errors is imminent.

This brings us to the two problems we want to focus on in our research:

1. The design of a spreadsheet is hidden behind the spread-



AccountNr	Description	Q1	Q2	Q3	Q3	Total
80000	Sales Revenu	€ 9.091	€ 7.124	€ 6.873	€ 6.694	€ 29.782
30000	Cost of Sales	€ -5.091	€ -3.277	€ -3.299	€ -3.682	€ -15.349
Gross Profit		€ 4.000	€ 3.847	€ 3.574	€ 3.012	€ 14.433

Associated spreadsheet

Figure 1: Model and associated spreadsheet

sheet itself, making it very difficult for an user to get an overview of the design

2. Users are not aware of the best way to structure a spreadsheet and just start modeling. The model works, but is poorly structured and error-prone.

3. PROPOSED SOLUTION

To address the above mentioned problems, we will develop an alternative user interface for the development of a spreadsheet. The basis for this user interface is a visual language (see also Figure 1). One of the success factors of spreadsheets is their flexibility and ease of use. If users have to learn a specific programming language before they can start developing a spreadsheet, we expect that the adoption of this alternative user interface will be very low. However, if we can develop a visual language that is easy to understand, works intuitively and at the same time makes use of a drag and drop interface, we expect a higher adoption.

It will be possible for the user to develop a spreadsheet with the visual language in one screen and seeing the associated spreadsheet in another screen. The link between the model and the associated spreadsheet should be bidirectional. Changes made in the model should be propagated to the associated spreadsheet and vice versa.

To implement this solution we face many challenges. It it is not in the scope of this paper to address them all, but we would like to highlight two of them in more detail. First of all, we face the challenge of the scalability of a visual language. Real-life spreadsheets are often complex models and it is difficult to present such a model visually in a clear way. Careful attention should be paid to the level of details that are presented in the visual language. Second, if we want to make the link between the model en the spreadsheet bidirectional, we have to think about what kind of operations are allowed in the spreadsheet. How should we, for example, handle a change in the spreadsheet that violates the model?

4. HYPOTHESIS

There are several hypotheses underpinning the proposed solution that will be evaluated during the research:

1. Users are more likely to use a visual language than a written language as an alternative to develop a spreadsheet.
2. The representation of a spreadsheet in the visual language will help users to get an overall overview of the spreadsheet.
3. If the user has a better overall overview of the spreadsheet, the spreadsheet will contain fewer errors.
4. The automatic spreadsheet generator will use common design patterns to structure the data in a spreadsheet. This will improve the underlying design of the spreadsheets.
5. A better structured spreadsheet contains fewer errors.

5. APPROACH

A spreadsheet model consists of formulas and operations on data. These formulas and operations are the constructs of our visual language. To develop this visual language we have to research what constructs should be included. With our language it is possible to model the majority of questions that are solved with spreadsheets. This implies that we need to get an understanding of the different questions that users try to solve with spreadsheets. Finally we will also explore if the constructs we find are depending on the domain in which the spreadsheet is used or if they are used regardless of the domain.

To answer these questions we will use the EUSES spreadsheet corpus [6]. This corpus contains over 4000 real world spreadsheets from domains such as finance, biology, and education. We will analyze what kind of formulas are used in spreadsheets and how they are combined. This will be translated to the constructs that are needed to build the visual language. The spreadsheets from the EUSES corpus will be complemented with real-life spreadsheets collected from our industrial partners.

To automatically generate a spreadsheet and to structure it in the most optimal way, we will inventory the best practices in spreadsheet design. Besides, we will research if additional design patterns are needed to cover the majority of questions. We will carry out a literature study to get an overview of the commonly used and known design patterns. Furthermore, we search for additional design patterns by analyzing the spreadsheets in the EUSES corpus and the spreadsheets collected from our industrial partners.

Based on the knowledge gained about the required constructs for the visual language and the best practices to structure a spreadsheet, we develop a prototype of a 'graphical spreadsheet generator'. The spreadsheet generator will generate the spreadsheet based on the specifications made

with the visual language and uses a suitable design pattern to structure the data. At this point, we also investigate if it is possible to automatically generate a graphical representation (using the syntax of the visual language) from an existing spreadsheet. This is needed to synchronize the visual model with the spreadsheet and allow the users to make changes in both the model and the spreadsheet.

6. EVALUATION

To validate our hypotheses, we carry out two different kinds of evaluations. First, we will evaluate the impact of the visual language on the behavior of the users. Is it true that they have a better overall overview of the spreadsheet and do they prefer a visual language over a written language? These two questions can be answered with case studies [11]. In the case studies we will ask users to develop a real-life spreadsheet with the new interface and afterwards interview them about their experiences.

Furthermore, we want to know if the spreadsheets that are developed with our visual language contain less errors and if this is caused by a better structure or because the end-user has a better overview/understanding of the spreadsheet or both. To evaluate this, controlled experiments will be performed. Two sets of participants are asked to develop a certain spreadsheet model. One group will use the visual language, the other group the classical spreadsheet interface. The two resulting sets of spreadsheets will be compared with each other concerning the number of errors and development time. Besides the experiments, we interview the participants to get a better insight of the users experience with the visual language.

7. RELATED WORK

Already in 2001 Burnett et al [2] developed Forms/3, a general purpose visual programming language. Main rationale to develop this language was to remove spreadsheet limitations without sacrificing consistency with the spreadsheet paradigm. Two principles in particular guided the development process: directness and immediate visual feedback.

In our approach we are less concerned with the limitations of modern spreadsheet languages. We want to improve the overall quality of spreadsheets by introducing a visual language that supports users by visualizing the design of their spreadsheet and help them to better structure their data. However, directness and especially immediate visual feedback are also two valuable guiding principles in our research.

Engels and Erwig [5] have described an automatic transformation process to generate a spreadsheet from a so called ClassSheet. The development of ClassSheets is a further elaboration of the work on spreadsheet templates [1]. With ClassSheets, it is possible to model spreadsheets according to domain-related business object structures. The ClassSheet represents both the structure and relationships of the involved (business) objects and classes and the computational details of how attributes are related and derived from each other. ClassSheets help to reduce the semantic distance between a problem domain and a spreadsheet application.

Cunha et al. [4] have further improved the concept of ClassSheets. They have embedded the ClassSheets spreadsheet models in spreadsheets themselves. Because of this, users do not have to familiarize themselves with a new programming environment. Furthermore, the authors have presented a technique to perform co-evolution of the ClassSheet model and the related spreadsheet. Modifications to the model are automatically propagated to the spreadsheet.

The main difference between the ClassSheet approach and ours is the introduction of a visual language that does not use the tabular two-dimensional layout of spreadsheet design. We agree that users should not be required to familiarize themselves with a new programming environment before they can develop a spreadsheet. Therefore the visual language will be used in the same environment as the associated spreadsheet. However, if the model is represented in a spreadsheet-like layout, it is still difficult for users to get a good overview of the design of the model. That is the reason why we develop a language that specifies and visualizes the model at the same time.

Also, our visual language embraces object-oriented principles, but does not expect the users to be aware of these principles. The overall goal of our research is to apply software engineering principles to the design of spreadsheets to improve the overall quality of spreadsheets. However, the spreadsheet user - who is not a professional programmer - should be able to develop the spreadsheet without being required to have knowledge of these principles.

Furthermore, our visual language can be used to generate the associated spreadsheet. This does not imply that the user is restricted in influencing the layout of this spreadsheet (as is the case with the ClassSheet approach). It was estimated that 95% of U.S. Firms uses spreadsheets for financial reporting [10] and layout is an important factor for effective reporting.

Finally, the current ClassSheets approach enables the co-evolution of the spreadsheet model and the spreadsheet data. At the theoretical level, the evolution of the instance of the model and the co-evolution of the model itself has been realized [3]. In our research, we focus on bidirectional transformations and integrate them in the prototype of the spreadsheet generator.

8. EXPECTED CONTRIBUTION

This research will lead to the following contributions:

1. A classification of the type of questions that end-users try to solve with spreadsheets.
2. Better understanding of the kind of formulas that are used in spreadsheets and the way these formulas are combined to solve questions.
3. Best practices for the design/structure of spreadsheets.
4. A visual language to model spreadsheets.
5. Methods to automatically generate a spreadsheet from the visual language.
6. Methods to automatically generate a graphical representation (using the syntax of the visual language) of a spreadsheet.

7. A prototype of an alternative user interface for the development of spreadsheets that is based on a visual language.

9. REFERENCES

[1] R. Abraham, M. Erwig, S. Kollmansberger, and E. Seifert. Visual specifications of correct spreadsheets. In *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*, pages 189–196. IEEE, 2005.

[2] M. M. Burnett, J. W. Atwood, R. W. Djang, J. Reichwein, H. J. Gottfried, and S. Yang. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of functional programming*, 11(2):155–206, 2001.

[3] J. Cunha, J. P. Fernandes, J. Mendes, H. Pacheco, and J. Saraiva. Bidirectional transformation of model-driven spreadsheets. In *Theory and Practice of Model Transformations*, pages 105–120. Springer, 2012.

[4] J. Cunha, J. Mendes, J. Saraiva, and J. P. Fernandes. Embedding and evolution of spreadsheet models in spreadsheet systems. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 179–186. IEEE, 2011.

[5] G. Engels and M. Erwig. Classsheets: automatic generation of spreadsheet applications from object-oriented specifications. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 124–133. ACM, 2005.

[6] M. Fisher and G. Rothermel. The euses spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.

[7] F. Hermans, M. Pinzger, and A. van Deursen. Supporting professional spreadsheet users by generating leveled dataflow diagrams. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 451–460. ACM, 2011.

[8] F. Hermans, B. Sedee, M. Pinzger, and A. v. Deursen. Data clone detection and visualization in spreadsheets. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 292–301. IEEE Press, 2013.

[9] R. R. Panko. What we know about spreadsheet errors. *Journal of Organizational and End User Computing (JOEUC)*, 10(2):15–21, 1998.

[10] R. R. Panko and N. Ordway. Sarbanes-oxley: What about all the spreadsheets? *arXiv preprint arXiv:0804.0797*, 2008.

[11] R. K. Yin. *Case study research: Design and methods*, volume 5. sage, 2009.