Analyzing and visualizing
spreadsheets

Analyzing and visualizing
spreadsheets

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op woensdag 23 januari om 10:00 uur
door

Félienne Frederieke Johanna HERMANS

ingenieur in de informatica
geboren te Oudenbosch.

Dit proefschrift is goedgekeurd door de promotoren:

Prof. dr. A. van Deursen
Co-promotor: Dr. M. Pinzger

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. A. van Deursen	Technische Universiteit Delft, promotor
Dr. M. Pinzger	Technische Universiteit Delft, co-promotor
Prof. dr. E. Meijer	Technische Universiteit Delft & Microsoft
Prof. dr. H. de Ridder	Technische Universiteit Delft
Prof. dr. M. Burnett	Oregon State University
Prof. dr. M. Chaudron	Chalmers University of Technology and University of Gothenburg
Dr. D. Dig	University of Illinois at Urbana-Champaign

Author email: felienne@felienne.com

Acknowledgements

First of all, I would like to thank Arie. Thanks for always supporting my decisions, for being there when times were a bit tough and for loads of fun and inspiration. Martin, thanks for the gazillion spelling errors you corrected in all my papers, your patience was amazing. Stefan Jager, thanks for trusting me to experiment freely at Robeco! You allowed me to really test my ideas in practice, even when they were very preliminary.

Robert, thanks for several kilos of home made fries and for always, always laughing at my stupid jokes. Anja, I sincerely enjoyed all the time we spent 'together', I hope there is a lot more to come. The two of you are the best paranymphs a girl could wish for. And Maartje, thanks for the awesome design of my cover.

Finally, I would like to thank Jan Karel Pieterse for his extensive proof reading of a draft of this dissertation. Without his eye for detail it would have, without a doubt, contained numerous more mistakes than it does now.

Delft, *Felienne Hermans*
September 2012

Contents

Chapter 1

Introduction

1.1 Spreadsheets

Spreadsheets can be considered the most successful programming paradigm in the history of computers. End-user programmers outnumber software programmers [Sca05] and it has been estimated that 90% of computers have Excel installed [Bra09].

Their use is diverse, ranging from inventory administration to educational applications and from scientific modeling to financial systems, a domain in which their use is particularly prevailing. Panko [Pan06] estimates that 95% of U.S. firms, and 80% in Europe, use spreadsheets in some form for financial reporting.

One of the success factors of spreadsheets is their easy-to-use interface and great flexibility. Users do not have to think about a design of their spreadsheet programs: they can just start entering data and formulas. However, there are some scenario's in which a spreadsheet user wants to have information about the underlying design of the spreadsheet. For instance, when debugging, when reading a spreadsheet created by someone else or when making big changes to the spreadsheets. Given the fact that the design is hidden 'behind' the spreadsheet and inaccessible for the user, difficulties can arise in those scenarios where a user does need to understand the spreadsheet's design.

This research aims at providing spreadsheet users with information about their spreadsheet's design, whenever they need and want that information.

1.2 A brief history of spreadsheets

Although the electronic spreadsheet was first conceived in the sixties, the idea of laying out numbers in a grid dates as far back as the Babylonian times. The Plimpton 322, a Babylonian tablet from 1800 BC, lists the Pythagorean triplets in

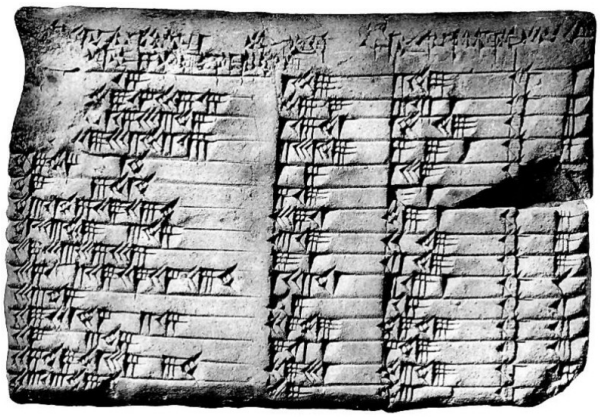


Figure 1.1: *Plimpton 322 showing the Pythagorean triplets: A spreadsheet from 1800 BC*

a very spreadsheet-like form, as shown in Figure 1.1. The first two columns can be considered input and the third column represents the results of the Pythagorean calculation (which had been known years before Pythagoras was born around about 570 BC). An interesting fact about this tablet is that there is a *copy-paste* error in it: one of the numbers in row 9 actually belongs in row 8. [1]

Mathematical tables like Plimpton 322 were used for centuries, both for mathematical purposes, such as calculation and teaching, as for administrative purposes like inventory logging. However, spreadsheet-like interfaces became more mainstream in the 15th century, when the Italian mathematician Luca Pacioli first described the double-entry bookkeeping system in his famous book 'Summa de arithmetica, gemometria, proportioni et proportionalita'. This system consists of two sides: a debit and a credit side. Each transaction has to be entered twice (hence "double-entry"), once on each side and both sides have a very spreadsheet like form. The popularity of this bookkeeping system, which is still used today in a very similar form, rendered to the grid interface a natural representation for financial records.

The story of the electronic spreadsheet begins in 1964. In his book *Simulation of the Firm through a Budget Computer Program* [Mat64] Mattessich "foreshadows the basic principles behind today's computer spreadsheets: the use of matrices, (budget) simulation, and, most important, the calculation that supports each matrix cell." [Cha96].

Mattessich also created 48 spreadsheets with FORTRAN IV and bundled the output with his book. The spreadsheets concerned many different domains, including "labor costs, material costs, purchases, sales, overhead expenses with

[1] In Chapter 6 of this dissertation, we will address copy-paste errors in modern spreadsheets

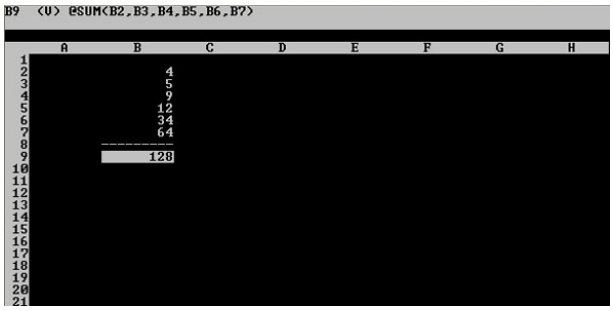


Figure 1.2: *VisiCalc (1979)*

proper allocations, as well as a projected income statement and balance sheet". These spreadsheets are generally considered the first electronic spreadsheets.

In 1979 VisiCalc 1.2 was created by Dan Bricklin and Bob Frankston. VisiCalc was the first commercially successful spreadsheet program, selling as much as 20.000 copies a month in 1983 [Sla89]. In many respects, VisiCalc is similar to modern spreadsheet systems. VisiCalc had the ability to automatically recalculate the value of a cell when the content was updated and to propagate this update to cells depending on it. "This feature of automatic updating and propagation to all spreadsheet cells was so unique for a programming system that it was discussed in many books on artificial intelligence in the 1980s." [Fre66]. In addition to the recalculation of cells, VisiCalc also supported copy-pasting cells and ranges with relative and absolute references. Also, it was able to construct formulas by selecting cells.

The first version of VisiCalc ran on an Apple II. When it launched in November 1979, at a retail price of $100, it instantly became a big hit and dealers started to bundle the software with the Apple II.[2] VisiCalc had a big impact on the success of the Apple II: many companies bought an Apple II solely to run VisiCalc on. As Steve Jobs himself stated in a 1994 interview with Rolling Stone magazine: "What drove the success of the Apple II for many years and let consumers have the benefit of that product was VisiCalc selling into corporate America. Corporate America was buying Apple IIs and running VisiCalc on them like crazy."

VisiCalc reigned the spreadsheet world until 1983, the year when Lotus 1-2-3 was released. Lotus 1-2-3 was built in highly optimized 8086 assembly-language, making it very fast. In addition to higher speed, Lotus 1-2-3 could also use more memory, which in turn allowed for larger spreadsheets. These benefits made that it quickly became the new industry spreadsheet standard.

In 1982 Microsoft released their first spreadsheet program: Multiplan. A fundamental difference between Multiplan and its competitors was Microsoft's

[2] http://history-computer.com/ModernComputer/Software/Visicalc.html.

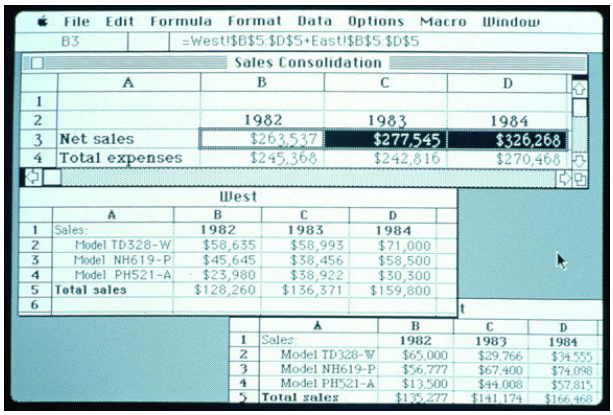


Figure 1.3: *Excel 1.0 (1985)*

decision to use R1C1 addressing instead of the A1 addressing as introduced by VisiCalc.

In the A1 addressing mode, cells are referenced by a letter indicating their column and a number denoting their row. B4, for example, means the cells in the second column and the fourth row. Both the column and the row indication can be relative—meaning that it will be changed when the formula is copied— or absolute, indicated by a $ in front of it and meaning that it will not be changed.

In the alternative R1C1 notation, cells are denoted by the letter R followed by the row number concatenated with the letter C followed by the column number. By putting the row or column numbers between square brackets, a row or column relative to the current cells in indicated. In this *relative* R1C1 notation, that is the default R1C1 style, *SUM(A2:D2)* in cell E2 is written as *SUM(RC[-4]:RC[-1])*.

Multiplan was successful on CP/M systems, but it never became as successful on MS-DOS. While Microsoft had initially planned to create a Multiplan version for Windows, this version was never released as Lotus 1-2-3 kept outselling Multiplan. Instead, Microsoft released Excel for Mac in September of 1985, just 3 years after the introduction of Multiplan. Figure 1.3 shows the interface of Excel 1.0.

Microsoft decided to keep the R1C1 notation in Excel, in addition to the A1 referencing notation and until today, both reference modes are still offered in the newest version of Excel.

One of the distinguishing features of the new Excel system was its ability to be operated with a mouse and drop down menus. This made Excel easier and faster to use than competing spreadsheet systems. A second unique feature that contributed to Excel's popularity was the fact that users could change the style of their spreadsheets with different fonts and markup options. Finally, calculations in Excel were executed more quickly, because of optimized cell recalculation, in

which only dependent cells of a modified cell are updated.

In 1987 Microsoft released Excel for Windows, and since Windows was not that popular at the time, Microsoft supplied a copy of Windows for free with each copy of Excel 2.0. Over the next few years, Excel started to challenge the position of Lotus 1-2-3. By 1988 Excel had overtaken Lotus as the dominant spreadsheet system, which it remains to date.

1.3 Motivation

Despite the huge success of spreadsheets in practice, their use is not at all without problems. The European Spreadsheet Risk Interest Group collects *horror stories*[3] about organizations losing substantial amounts of money due to errors in spreadsheets, among which stories like "$1M went missing as staff managed monstrous spreadsheets." and "£4.3M spreadsheet error leads to resignation of its chief executive". These stories, although anecdotal, underline the impact that spreadsheets can have on organizations. While other assets of companies—such as processes or software—are often validated and controlled, spreadsheets are not.

However big the damage of errors in spreadsheets is, we consider errors as merely symptoms of the real problems with spreadsheets. The underlying problem is the fact that the *design* of a spreadsheet is hidden behind the formulas and worksheets of a spreadsheet. For instance, if a worksheet dependents on calculations in another worksheet, one cannot easily see this at the worksheet level unless one looks deep down in the formulas and cells.

Some users initially have a design in mind, but as the spreadsheet is changed over time and the spreadsheet gets disconnected from its original design. Another reason the design is hidden is the fact that spreadsheets are often transferred from one employee to the other. The recipient can only view the spreadsheet itself, but its design is not transferred.

The disconnectedness between a spreadsheet and its design is the 'dark side' of the lack of design and flexibility of spreadsheets: it causes users to be afraid to modify spreadsheets after some time, since they do not know what might happen. In that case, spreadsheets are sometimes built anew—which is time-consuming—or, even worse, kept in business despite their user's lack of in-depth knowledge on them.

In this dissertation we assert that the real problem with spreadsheets is the missing connection between a spreadsheet and its underlying design. While the invisibility of the design is very useful and user-friendly in most cases, there are several scenarios in which it is of great importance to the spreadsheet user to be able to view and even modify this model.

Situations in which a spreadsheet user wants to see the design are, among others:

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

- When receiving a spreadsheet from a colleague
- When asked to judge the quality of a spreadsheet
- When making a major change to the spreadsheet

More powerful end-user programming systems, such as Microsoft Access or Filemaker, solve this problem by forcing the user to create the design upfront. However, we consider the flexibility of spreadsheets their greatest power and think the number of situations where design information is needed are rare, but important nonetheless. We assert that it is better to allow users to work as they prefer, while supporting them where needed. Hence, the central thesis of this research is *it is possible to support spreadsheet users in understanding and adapting the design of their spreadsheets, without losing the flexibility of the spreadsheet programming paradigm.*

1.4 Related work

In this section we present a high-level overview of the state of the art in spreadsheet research. While the problem of revealing the design of spreadsheets has not been solved, there is substantial amount of work already done in spreadsheet analysis. In this introduction, we mainly touch on areas of research that have been done towards our research question. The main chapters of this dissertation (Chapters 2-6) provide a more detailed background on related work.

1.4.1 Taxonomy of spreadsheet errors

Panko [Pan98] presents an overview of seven different field audits into spreadsheet errors and shows that 86% of spreadsheets contain an error.

Ronen, Palley, & Lucas [Ron89] were the first to make a classification of different types of errors, which they based on a literature review. It contains categories that relate to the model itself, such as mistakes in logic and those concerning the execution of the model, like incorrect cell references.

Panko and Halverson [Pan96] were the first to strive for a complete model of spreadsheet errors and distinguished between quantitative and qualitative errors. Quantitative errors are numerical errors and immediately produce a wrong result, whereas qualitative errors do not lead to an incorrect value, but may produce them when the spreadsheet is edited. They presented the following three main categories of errors:

Mechanical Errors result from typing errors, pointing errors, and other simple slips;

Logic Errors result from choosing the wrong algorithm or creating the wrong formula;

Omission Errors result from leaving parts out of the model and often lead to misinterpretation of the results of the model.

Most spreadsheet error papers focus on detecting mechanical errors, some focus on helping the user spot logic errors. This research focuses on extracting the design from a spreadsheet so errors can be found on the design level. This is why we expect our method to be able to help the user find logic and even omission errors.

1.4.2 Automated error detection

Erwig and Burnett have described a method for inferring types from spreadsheets [Erw02] that can be used for static checking of spreadsheets [Ahm03]. This work was implemented in the UCheck tool [Abr07a]. Unit analysis catches a specific type of errors, namely formulas that incorrectly calculate with different (inferred) types, which can be both a result of logic errors and mechanical ones. UCheck was tested on spreadsheets created by university students [Abr07a] and an evaluation with high school teachers was performed [Abr07b], which showed that UCheck supports users in correcting unit error. We are not aware of industrial evaluations of this method, hence we do not know its effectiveness when applied to spreadsheets from practice.

Besides academic papers, the presence of errors in spreadsheets has led to a series of commercial software packages, such as Spreadsheet Detective, Excel Auditor and Operis Analysis Kit. Research has shown that these tools are quite suitable for detecting mechanical errors such as values stored as text (82% success rate) or incomplete range detection (55% success) [And04]. They however do not catch errors in logic, such as the omission of a variable (18%) or an operator precedence error (9%).

In 2010 Panko [Aur10] conducted a study in which several of those commercial tools were compared, both with each other and with human auditors. “Overall, the two spreadsheet static analysis tools performed terribly, and their detection strengths paralleled the detection strengths of people instead of compensating for weaknesses in human error detection. The 33 human inspectors found 54 of the 97 errors, while only 5 errors were tagged by Error Check and Spreadsheet Professional.” [Aur10].

We conclude that—although methods for automatic detection can certainly find errors—they mainly speed up the process of error finding and do not detect many errors that would not have been found by humans. In their current form, they do not support users in assessing a spreadsheet’s design.

1.4.3 Spreadsheet testing

A third category of related work concerns the testing of spreadsheets. Centrepiece in this category is the work of Rothermel et al., who have created [Rot97] and

subsequently validated [Rot00] a method to support end-users in defining and analyzing tests for spreadsheets. Their evaluation showed that their approach had an average fault detection percentage of 81% which is "comparable to those achieved by analogous techniques for testing imperative programs." Other studies have confirmed the applicability of testing [Kru06]. The WYSIWYT methodology requires users to explicitly indicate what cells of a spreadsheet are correct and the system propagates the testedness of a cell to its dependents. Related is the elegant work of Burnett on spreadsheet assertions that uses a similar propagation system [Bur03].

Although we strongly believe in the applicability of testing to spreadsheets, we conclude that spreadsheet testing in its current form is useful to validate the results of a spreadsheet and not to validate the design of the spreadsheet itself. The information a test case gives, whether failing or succeeding, is only local and does not support the spreadsheet user in understanding and improving the design of a spreadsheet as a whole.

1.4.4 Refactoring

Refactoring source code is a"disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior"[Fow99]. Refactoring can also be applied to spreadsheet formulas and is related to the extraction of models from spreadsheets. So far two papers on this topic have been published. O'Beirne [O'B10] describes three classes of spreadsheet refactorings: for worksheets, for data and for macros (that are written in Visual Basic for Applications in Excel). Badame and Dig [Bad12] have implemented and evaluated RefBook: a tool that supports the user in refactoring of formulas. RefBook offers seven different refactorings. Their evaluation shows that their refactorings are widely applicable, increase user productivity and the increase the quality of the resulting spreadsheets.

While RefBook is very useful to improve formulas, it does not support the user in finding the complex formulas or understanding their role in the spreadsheet design. Hence, refactoring only does not solve our research question, but it is a complimentary approach.

1.4.5 Finding high-level structures

Most related is the work of Mittermeir and Clermont [Mit02; Cle03]. Their aim is similar to ours: to obtain a higher level view of spreadsheets. They find a higher level view of a spreadsheet by analyzing the several different types of equivalence:

Copy equivalence As defined in [Saj00], two cells are copy-equivalent, if they have the same format, and their formulas are equal

Logical equivalence Two formulas are logically equivalent, if they differ only in constant values and absolute cell-references

Structural equivalence Two formulas are structurally equivalent, if they differ in constant values, absolute- and relative references. Nevertheless, the same operators and functions are applied in the same order to different data [Cle03]

By abstracting over groups of cells that are equivalent, called equivalence classes, Mittermeir and Clermont obtains a higher level view of a given spreadsheet. This work has been an inspiration to us and was the first step in getting a high-level overview of large and complex spreadsheets. Our aim however is broader, as we also consider data and furthermore also take the division of the formulas in worksheets into account.

1.4.6 Modeling spreadsheets

Abraham *et al.* have developed a system called ClassSheets [Abr05b; Eng05] with which the design of a spreadsheet can be described. From this specification, spreadsheets can be generated. A related effort is the work of Cunha, who transforms a spreadsheet into a database by analyzing functional dependencies [Cun09a]. This work however does not communicate the extracted information to the user, but rather uses it to enhance the spreadsheet system with features such as auto-completion and safe deletion [Cun09c]. While evaluations showed that this systems makes users more efficient and effective, the authors also conclude that "that deeper insight on the spreadsheet models is required to maximize effectiveness" [Bec11].

1.5 Research questions

From the previous section we can conclude that the problem of design extraction from spreadsheets has not yet been addressed fully. The central problem of this dissertation, *supporting spreadsheet users in understanding and adapting the design of their spreadsheets*, has not been solved.

Spreadsheet are similar to software in many ways. Firstly, both concern data and the manipulation of that data, and both are concerned with presenting results to the user. Also, in both cases organization matters: calculations can be structured into coherent groups (worksheets and classes for instance). Finally, both spreadsheets and software are built and maintained by different people with often different styles and preferences on how to organize them.

However, in contrast to spreadsheet users, software engineers are known and trained to be very much interested in the design of software systems and the subsequent evaluation and adaption of software systems. Therefore in software engineering, much research has already been done in the field of automated analysis of software that could be potentially transferred to spreadsheets.

In conclusion, software engineering is a feasible topic to get our inspiration from. This idea leads to the central research questions of this dissertation:

RQ1 To what extent are methods and techniques from software engineering applicable to the spreadsheet domain?

RQ2 How can we visualize the underlying design of a spreadsheet in order to support spreadsheet understanding?

RQ3 To what extent does the visualization and analysis of the design of a spreadsheet support users in assessing quality of their spreadsheet?

When answering these research questions, we consider the different parts of which a spreadsheet design consist.

First, spreadsheets contain **data**. Secondly, there is the aspect of **metadata**: data entered as a description of other data. While spreadsheet systems do not distinguish this type of data from 'ordinary' data, spreadsheet users often do, by placing this data in the first row or column or giving it a different layout such as a bold or larger font. A third and central component are the **formulas**. This is the way that calculations are performed in a spreadsheet. Finally, there is the **organization** of a spreadsheet. This relates mostly to the way in which the data is divided into worksheets and into different tables within those worksheets.

For each of the four different aspects, we use different methods and techniques to analyze and visualize them.

1.6 Methodology

To answer our research questions, we have conducted several different studies. Central in all these studies has been the use of industrial spreadsheets and, where possible, the involvement of the users of the spreadsheets into our research.

In the studies, we have followed this research strategy:

1. Observe and learn form spreadsheet users and identify a real-life problem
2. Propose and implement a solution
3. Evaluate the approach with actual spreadsheets and their users

For the first step, the core method of this research is based on *grounded* theory. Grounded theory is a qualitative research method that discovers theory from data [Gla67]. When using grounded theory, observations are documented and subsequently *coded* into categories. During this coding process, which takes place after each observation—as opposed to after all observations—a theory might emerge [Ado08]. Grounded theory is a method especially suited for discovering problems that exist for the participants. Therefore, in a grounded theory study, the researcher works with a general area of interest rather than with a specific problem [McC03]. Before we started our first user study, we performed a study based on the principles of grounded theory [Her11]. The aim of this study was to

collect a set of problems that could be solved in the remainder of the research. We did not perform this study to validate an idea we already had. We wanted to learn about problems and find suitable solutions to them, possible from the software engineering domain.

Only after we had collected a set of problems, of which understandability and quality control were the most prominent ones, we started to create prototypes. While building prototypes, we took our inspiration from analyses that are well established in the domain of software engineering, and adapted them to the domain of spreadsheets.

For the third step, we have employed a *mixed method* evaluation: a combination of both qualitative and quantitative evaluations [Cre03]. More specifically, we used an *explanatory sequential design* in these mixed methods studies [Tas98]. An explanatory sequential study starts with a quantitative study to get initial answers to research questions and then searches for context in real-life case studies.

In two of the case studies, the spreadsheets under study were confidential and thus we could not share them. While we are strong believers in open data, we value research in a real industry setting even higher and this sometimes comes at the price of reduced repeatability. When possible, we have shared our data online.

We furthermore consider it important to share results from research with those who might benefit from it and have therefore, through our spinoff Infotron, made our tool Breviz available as a service.[4] While some researchers choose to make there tools open source, we consider a low threshold for trying more important and a simple upload-analysis service suits the needs of the end-users our tool aims at.

1.7 Background of this dissertation

The main chapters (Chapters 2-6) of this dissertation are slight adaptations of previously published conference papers. Since these papers were published independently, they can be read independent of each other. This is why there is some redundancy in background, motivation and examples in these chapters.

The following section gives on overview of this dissertation. In Chapter 2 we introduce a technique to automatically extract metadata information from spreadsheets and transform this information into class diagrams. The resulting class diagram can be used to understand, refine, or re-implement the spreadsheet's functionality. To enable this transformation we create a library of common spreadsheet usage patterns. These patterns are localized in the spreadsheet using a two dimensional parsing algorithm. The resulting parse tree is transformed and enriched with information from the library. We evaluate our approach on the spreadsheets from the EUSES Corpus [Fis05a] by comparing a subset of the generated class diagrams with reference class diagrams created manually. This

[4] app.infotron.nl

chapter previously appeared in the Proceedings of the 2010 European Conference on Object Oriented Programming (ECOOP) [Her10].

Chapter 3 describes a study of the problems and information needs of professional spreadsheet users by means of a survey conducted at a large financial company. The chapter furthermore proposes an approach that extracts needed information from spreadsheets and presents it in a compact and easy to understand way: with leveled data flow diagrams. Our approach comes with three different views on the data flow that allow the user to analyze the data flow diagrams in a top-down fashion. To evaluate the usefulness of the proposed approach, we conducted a series of interviews as well as nine case studies in an industrial setting. The results of the evaluation clearly indicate the demand for and usefulness of our approach in easing the understanding of spreadsheets. This chapter was published in the Proceedings of the 2011 International Conference on Software Engineering (ICSE)[Her11].

Subsequently, in Chapter 4 we aim at developing an approach for detecting *smells* that indicate weak points in a spreadsheet's design. To that end we first study code smells and transform these code smells to their spreadsheet counterparts. We then present an approach to detect the smells, and to communicate located smells to spreadsheet users with data flow diagrams. To evaluate the approach, we analyzed occurrences of these smells in the EUSES corpus. Furthermore we conducted a case study with ten spreadsheets and their users in an industrial setting. The results of the evaluation indicate that smells can indeed reveal weaknesses in a spreadsheet's design, and that data flow diagrams are an appropriate way to show those weaknesses. This work appeared in the 2012 Proceedings of the International Conference on Software Engineering (ICSE) [Her12b].

In Chapter 5 we study smells in spreadsheets, however in this chapter we focus on smells at the formula level. Again we take our inspiration from known code smells and apply them to spreadsheet formulas. To that end we present a list of metrics by which we can detect smelly formulas and a visualization technique to highlight these formulas in spreadsheets. We implemented the metrics and visualization technique in a prototype tool to evaluate our approach in two ways. Firstly, we analyze the EUSES spreadsheet corpus, to study the occurrence of the formula smells. Secondly, we analyze ten real life spreadsheets, and interview the spreadsheet owners about the identified smells. The results of these evaluations indicate that formula smells are common and that they can reveal real errors and weaknesses in spreadsheet formulas. This chapter appeared in the 2012 Proceedings of International Conference on Software Maintenance (ICSM) [Her12c].

In Chapter 6 we study *cloning* in spreadsheets. Based on existing text-based clone detection algorithms, we have developed an algorithm to detect *data clones* in spreadsheets: formulas whose values are copied as plain text in a different location. To evaluate the usefulness of the proposed approach, we conducted two evaluations: A quantitative evaluation in which we analyzed the EUSES

Corpus and a qualitative evaluation consisting of two case studies. The results of the evaluation clearly indicate that 1) data clones are common, 2) data clones pose threats to spreadsheet quality and 3) our approach supports users in finding and resolving data clones. The final chapter of this dissertation will appear in the 2013 Proceedings of the International Conference on Software Engineering (ICSE) [Her13]. Finally, Chapter 7 presents the conclusions of this dissertation and discusses future work.

In addition to the above listed papers, we have written several other publication over the course of the PhD project. Firstly, there is the 2010 MODELS publication: *Domain-specific languages in practice: A user study on the success factors*, which studies the success of domain specific languages in practice. We analyzed the impact of ACA.NET, a DSL developed by Avanade, that it is used to build web services that communicate via Windows Communication Foundation. We evaluated the success of this DSL by interviewing developers. The questionnaire that we developed for this evaluation has been built upon by several other researchers [Gon10; Gab10].

The research conducted for this paper led to the insight that spreadsheets could be a viable topic for research. While interviewing business professionals, we noticed that they were not so interested in learning to program in a DSL, since they were already programming in Excel.

In 2010 we furthermore published a position paper at ICSE workshop FlexiTools on the possibility of extracting the information in documents to support the requirements analysis phase of a software project. In 2011, we wrote a short overview of our analysis tool Breviz that was published at the 2011 annual conference of the European Spreadsheet Risk Interest Group (Eusprig). This paper was awarded the David Chadwick prize for best student paper. At Eusprig 2012 we published a paper on spreadsheet formula metrics that formed the basis for our ICSM 2012 paper that is incorporated in this dissertation as Chapter 5.

Chapter 2

Extracting Class Diagrams from Spreadsheets

2.1 Introduction

To design and implement a software system a high degree of familiarity with the domain of the software is needed. We conjecture that a significant portion of this domain knowledge is already available in digital form. In particular spreadsheets, which are widely used for many professional tasks, are likely to contain a wealth of implicit knowledge of the underlying application domain. It is the purpose of this chaper to help make this knowledge explicit.

Spreadsheets were introduced in the early 1980's with the first spreadsheet tool called VisiCalc. This tool was then followed by SuperCalc and Lotus 123 and later on by Excel which currently is one of the most prominent spreadsheet tools. Since their introduction, spreadsheets are heavily used in industry. A study from the year 2005 shows about 23 million American workers use spreadsheets, which amounts to about 30% of the workforce [Sca05].

Spreadsheets can be a rich source of information concerning the structure of the underlying domain. They contain groups of data, computations over these groups, and data dependencies between them. In our approach, we will attempt to make this structure explicit, by representing it as a class diagram. Groups of data are turned into classes, formula's into methods, and data dependencies into associations. The resulting class diagram can be used by software engineers to understand, refine, or re-implement the spreadsheet's functionality.

The research community noticed the importance of spreadsheets and devoted considerable research effort to them. This research mainly aims at two directions: 1) the localizations of errors within existing spreadsheets [Abr04; Abr06; Abr09; Abr07b; Ahm03; Mit02] and 2) the development of guidelines on how to cre-

	A	B	C	D	E
1		Fruit			
2	Month	Apple	Orange	Total	
3	May	8	11	19	
4	June	11	5	16	
5	Total	19	16	35	

Figure 2.1: *Fruit example taken from [Abr04].*

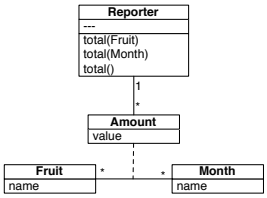


Figure 2.2: *Class diagram extracted from the fruit example.*

ate well-structured and maintainable spreadsheets [Fis02; Jan00; Kni00; Pan94; Ron89]. Both directions share the goal of improving spreadsheet quality, which is necessary because the current state of spreadsheet use leads to numerous problems as described in several papers, most notably in the work of Panko [Pan98].

While some guidelines for designing spreadsheets and algorithms for detecting errors in spreadsheets exist, the elicitation of the domain information stored in spreadsheets for developing software systems has, to the best of our knowledge, not been addressed, yet (See Section 2.10 for a discussion of the most directly related literature).

To illustrate our approach, we will use the example presented in Figure 2.1, taken from Abraham and Erwig [Abr04].

This spreadsheet is used to list the number of different fruits (i.e., apples and oranges) that have been harvested in May and June. It also provides functions to calculate the total numbers per fruit, per month, and a function for the calculation of the overall number of harvested fruits. The structure of this spreadsheet is a common pattern that occurs in many spreadsheets. Taking a closer look at this spreadsheet, the information it contains could be represented by the class diagram shown in Figure 2.2.

For the extraction of this class diagram, first the two classes *Fruit* and *Month* were identyfied, with instances *Apple* and *Orange* and *May* and *June* respectively. The two classes are linked with each other by the cells B3 to C4 that specify the amount of fruits (instances of class *Fruit*) for each instance of the class *Month*. This link is represented by the association class *Amount* with an attribute *value*. Furthermore the spreadsheet contains operations to calculate the *Total*

per fruit, per month, and the overall total number of fruits. These operations can be provided by a *Reporter* class that we added to the class diagram. The resulting class diagram contains the core design elements to represent this spreadsheet and might be used by a developer, for example, to design a simple fruit-statistic application, or to reason about (errors in) the structure of the spreadsheet.

In this chapter we focus on the automation of the extraction of such class diagrams from spreadsheets. We propose a systematic approach, called Gyro, which is supported by a tool capable of analyzing Microsoft Excel sheets. Gyro transforms spreadsheets into class diagrams automatically by exploiting commonality in spreadsheets, like the pattern in Figure 2.1. To that end we create a library containing common spreadsheet patterns, inspired by both related work in the field of spreadsheet design and analysis of a range of existing spreadsheets. These patterns are detected within the spreadsheet using a combination of parsing and pattern matching algorithms. Each pattern in the library is associated with a mapping to a class diagram.

In order to evaluate our approach we made use of the EUSES Spreadsheet Corpus [Fis05b]. This corpus contains over 4000 real world spreadsheets from domains such as finance, biology, and education. In our evaluation we demonstrate that our patterns can be found in around 40% of the spreadsheets. Furthermore we provide a systematic comparison of the generated class diagrams for a random selection of 50 spreadsheets for which we manually derived class diagrams.

The remainder of this chapter is structured as follows: Section 2.2 introduces the necessary background information on modeling spreadsheets and two-dimensional language theory. The Gyro approach is presented in Section 2.3 with details of the parsing and transformation described in the Sections 2.4, 2.5 and 2.6. Section 2.7 gives a brief description of the current implementation of the Gyro prototype. The evaluation of the approach is presented in Section 2.8. The results are discussed in Section 2.9 followed by Section 2.10 that presents an overview of the work in the field of spreadsheets. The conclusions can be found in Section 2.11.

2.2 Background

Before presenting our Gyro approach for recognizing spreadsheet patterns, we provide a brief survey of the preliminaries we build upon. These originate from the realm of spreadsheet testing and analysis [Abr04; Mit02], as well as from the domain of two-dimensional languages [Gia96].

2.2.1 Cell types

Most papers on spreadsheet analysis distinguish between different cell types [Abr04; Mit02]. Abraham and Erwig [Abr04] for instance identifies header, footer, data and filler cells. Mittermeir and Clermont [Mit02] on the other hand defines empty

cells, formulas and constant values. We mostly follow the approach of the former, but we replace filler cells by empty cells. We do so because we use patterns to identify structure in a spreadsheet. Therefore we are not interested in filler cells, which usually occur between patterns. With this, the following basic cell types are recognized by our approach:

Label A cell that only contains text, giving information about other cells (called *header* in [Abr04])

Data A cell filled with data

Formula A cell containing a calculation over other cells (called *footer* in [Abr04])

Empty An empty cell

We prefer the terms *label* and *formula* over *header* and *footer*, because the latter have some kind of intrinsic meaning concerning their position. We want to be able to freely define any pattern, including some with 'footers' on top. To determine the type of a cell, we use a simple strategy, which basically follows the approach of [Abr06]. This algorithm is described in Section 2.4.3.

2.2.2 Pattern languages for two-dimensional languages

To define patterns over spreadsheets we make use of existing notations from the theory of *two-dimensional languages* [Gia96] which is a generalization of the standard theory of regular languages and finite automata.

Let Σ be a finite alphabet. Then we define:

Definition 1. *A two-dimensional* pattern *over* Σ *is a two-dimensional array of elements of* Σ*.*

Definition 2. *The set of all two-dimensional patterns over* Σ *is denoted by* Σ^{**}*. A* two-dimensional language *over* Σ *is a subset of* Σ^{**}*.*

Given a pattern p over an alphabet Σ, let $l_1(p)$ denote the number of rows of p and $l_2(p)$ denote the number of columns of p. The pair $\langle l_1(p), l_2(p)\rangle$ is called the *size* of p. Furthermore, if $0 \leq i < l_1(p)$ and $0 \leq j < l_2(p)$ then $p(i, j)$ denotes the symbol $\in \Sigma$ on position (i, j). The pattern with size $\langle 0, 0\rangle$ is called the empty pattern and is denoted with λ. Patterns of the size $\langle 0, n\rangle$ or $\langle n, 0\rangle$ with $n > 0$ are not defined.

Next we define concatenation operations used to combine patterns. Let p be a pattern over Σ of size $\langle m, n\rangle$ and q be a pattern over Σ' of size $\langle m', n'\rangle$. We first define the rectangle we can obtain by putting q to the right of p, assuming p and q have the same number of rows, resulting in a rectangle of size $\langle m = m', n + n'\rangle$

Definition 3. *The* column concatenation *of p and q (denoted by $p \oplus q$) is a partial operation, only defined if $m = m'$, is a pattern over $\Sigma \cup \Sigma'$ given by*

$$(p \oplus q)(i,j) = \begin{cases} p(i,j) & \text{if } j \leq n \\ q(i,j-n) & \text{otherwise} \end{cases}$$

Similarly, we define how we can position q directly below p if p and q have the same number of columns, resulting in a rectangle of size $\langle m + m', n = n' \rangle$

Definition 4. *The* row concatenation *of p and q (denoted by $p \ominus q$) is a partial operation, only defined if $n = n'$, is a pattern over $\Sigma \cup \Sigma'$ given by*

$$(p \ominus q)(i,j) = \begin{cases} p(i,j) & \text{if } i \leq m \\ q(i-m,j) & \text{otherwise} \end{cases}$$

We will refer to these two operations as the *catenation operations*. Catenation operations of p and the empty picture λ are always defined and λ is the neutral element for both catenation operations. The catenation operators can be extended to define concatenations between two-dimensional languages.

Definition 5. *Let L_1, L_2 be two-dimensional languages over alphabets Σ_1 and Σ_2 respectively, the* column concatenation *of L_1 and L_2 is a language over $\Sigma_1 \cup \Sigma_2$ denoted by $L_1 \oplus L_2$ is defined by*

$$L_1 \oplus L_2 = \{p \oplus q | p \in L_1 \wedge q \in L_2\}$$

Similarly the row concatenation *of L_1 and L_2 is a language over $\Sigma_1 \cup \Sigma_2$ denoted by $L_1 \ominus L_2$ is defined by*

$$L_1 \ominus L_2 = \{p \ominus q | p \in L_1 \wedge q \in L_2\}$$

Definition 6. *Let L be a pattern language. The* column closure *of L (denoted by $L^{*\oplus}$) is defined as*

$$L^{*\oplus} = \bigcup_{i \geq 1} L^{i\oplus}$$

where $L^{1\oplus} = L$ and $L^{n\oplus} = L \oplus L^{(n-1)\oplus}$. Similarly, the row closure *of L (denoted by $L^{*\ominus}$) is defined as*

$$L^{*\ominus} = \bigcup_{i \geq 1} L^{i\ominus}$$

where $L^{1\ominus} = L$ and $L^{n\ominus} = L \ominus L^{(n-1)\ominus}$.

We will refer to these two operations as the *closure operations*. With respect to priorities we define that closure operations bind stronger than catenation operations.

2.2.3 Pattern grammars

To describe common spreadsheet patterns, we make use of *pattern grammars*. Pattern grammars are a two-dimensional generalization of ordinary grammars. This generalization is based on the observation that a production rule of the form $S \rightarrow ab$ actually means that S may be replaced by a followed by b. In regular rewriting, the 'followed by' can only occur in one direction, so this is not expressed in the grammar. To define production rules in two dimensions, we use two symbols from two-dimensional language theory that express direction, $\ominus$ and $\oplus$ and their closure operations $^{*\ominus}$ and $^{*\oplus}$

Definition 7. *The set of all two-dimensional* pattern languages *over* Σ *is a subset of* Σ^{**} *called* $\mathcal{L}(\Sigma)$ *is inductively defined by:*

$$\begin{aligned}
\lambda &\in \mathcal{P}(\Sigma) \\
a &\in \mathcal{P}(\Sigma) && \text{, if } a \in \Sigma \\
L^{*\ominus} &\in \mathcal{P}(\Sigma) && \text{, if } L \in \mathcal{L}(\Sigma) \\
L^{*\oplus} &\in \mathcal{P}(\Sigma) && \text{, if } L \in \mathcal{L}(\Sigma) \\
L_1 \ominus L_2 &\in \mathcal{P}(\Sigma) && \text{, if } L_1, L_2 \in \mathcal{L}(\Sigma) \\
L_1 \oplus L_2 &\in \mathcal{P}(\Sigma) && \text{, if } L_1, L_2 \in \mathcal{L}(\Sigma)
\end{aligned}$$

To avoid ambiguity we use the convention that closure operations bind stronger than catenation operations.

Definition 8. *Just as a normal grammar, a* pattern grammar G *is defined as a quadruple*

$$G = (V, T, S, P)$$

where V *is a finite set of non-terminals,* T *is a finite set of terminal symbols,* $S \in V$ *is a special symbol called the start symbol, and* P *is a finite set of productions.*

Productions are tuples (v, p) of a non-terminal v and a pattern p, denoted as $v \rightarrow p$. v is also indicated with *lefthand side*, whereas p is called *righthand side*. Since we only allow non-terminals on the lefthand side, this is a context free grammar. The pattern grammars in our work will always consist of the basic cell types, thus the alphabet of terminals is always equal to $\{Label, Empty, Formula, Data\}$, therefore we will omit T in definitions of grammars. Unless indicated otherwise, *Pattern* is the start symbol S of any grammar in this chapter.

2.3 The Gyro approach to spreadsheet reverse engineering

The goal of this chapter is to distill class diagrams from spreadsheets. To that end we propose the Gyro approach, in which typical spreadsheet usage patterns can be specified, automatically recognized and transformed into class diagrams.

When investigating the way people use spreadsheets, we noticed that there are some common ways in which people represent information in spreadsheets.

Typically data that concerns the same topic is found grouped together in rows or columns separated by empty cells. These *spreadsheet patterns* are found in all kinds of spreadsheets, independent of the business area the spreadsheet originates from. We exploit this commonality by introducing a library of common spreadsheet structures. The transformation into class diagrams is done in two steps, as shown in Figure 2.3.

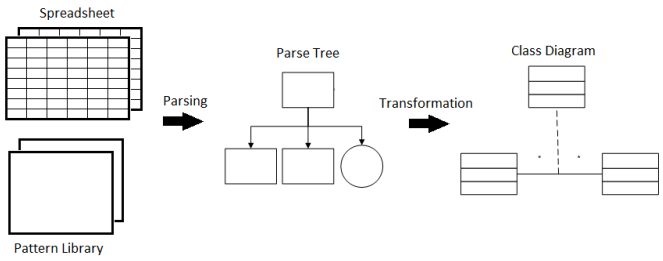


Figure 2.3: *Overview of the Gyro approach showing the two basic steps Parsing and Transformation to transform spreadsheets into class diagrams*

We start by localizing patterns from the library in the spreadsheet, by using a two-dimensional parsing algorithm. If a pattern is found the result of this algorithm is a parse tree. Each of the patterns in the library contains additional information that defines how the parse tree is transformed into a class diagram. This transformation represents the second step of our approach. The parsing is explained in more detail in Section 2.4 and Section 2.5 describes the transformation step.

The use of a library of patterns was greatly motivated by the need for flexible information extraction. We do not believe the current library contains all pattern grammars needed. So when we encounter a common spreadsheet that is not part of our library, we can add it to the library. Gyro is then able to recognize it immediately, without adaptation of the implementation. The patterns in the library are based both on existing work on spreadsheet design patterns [Jan00; Pan94; Ron89] and on the analysis of patterns encountered in the EUSES Corpus [Fis05b].

2.4 Pattern recognition

2.4.1 Overview

In order to identify regions in a spreadsheet that adhere to a given pattern, we follow the pattern recognition approach outlined in Figure 2.4. First, we identify rectangles in the spreadsheet that are filled with Data, Formula or Label cells, by using an algorithm to determine bounding boxes (Section 2.4.2). In Figure 2.1 for instance, this bounding box is given by the cells A1 $\times$ D5. Because we assume

occurrences of pattern are separated by empty cells, we evaluate these rectangles as possible occurrences of patterns. Next each cell is assigned one of the basic cell types: `Data`, `Formula`, `Label` or `Empty` (Section 2.4.3).

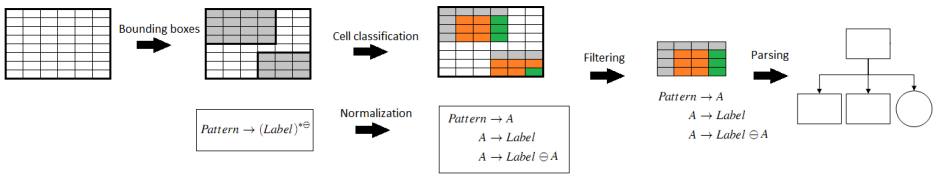


Figure 2.4: *Overview of the pattern recognition process*

In parallel, the grammar is normalized to a form that only contains catenation operations, in order to simplify the parsing step (Section 2.4.4). Then a filtering step is applied, in which we check the type of the left-most upper-most corner of the bounding box. We determine which of the patterns in the library have this symbol as a possible first terminal (Section 2.4.5) We only start the parsing algorithm if the cell in the left-most upport most corner matches a first terminal of one of the patters in the library. The parsing algorithm is an adaption of standard recursive descent parsing [Aho86] adjusted to handle two-dimensional grammars (Section 2.4.6).

It is possible several patterns are applicable for processing the cells of a bounding box. In such cases the algorithm returns the set of all matched patterns.

2.4.2 Finding bounding boxes

A bounding box is defined as the smallest rectangle containing a connected group of cells of type `Data`, `Label` or `Formula`. Two cells are connected if they touch each other horizontally, vertically or diagonally. To find such a bounding box, we apply the following strategy: Find the left-most upper-most non-empty cell that is not yet contained within a bounding box. The initial bounding is set to contain only this cell. Next this bounding box is expanded until the size remains stable. Expanding is done by inspecting all cells that connect to the bounding box. If one of these cells is non empty, the bounding box is enlarged to include this cell. In Figure 2.4 the identified bounding boxes are marked grey.

2.4.3 Cell classification

To distinguish between cell types, we use the cell classification strategy described by Abraham and Erwig [Abr04]. This algorithm starts with identifying all cells containing a formula and mark them as type `Formula` (green cells in Figure 2.4). Next we look at the content of the formula's. Cells that are referred to a formula are marked `Data`, unless they also contain a formula (orange cells in Figure 2.4).

In that case they are marked as `Formula` as well. Remaining cells that are empty are identified as `Empty` (white cells in Figure 2.4); all others are recognized as a `Label` (grey cells in Figure 2.4).

2.4.4 Normalization

To simplify the parsing algorithm, we assume that the pattern under consideration only consists of $\ominus$ and $\oplus$ symbols. Therefore, we first transform the pattern to this form, which we call *catenation normal form*. Every possible pattern has an equivalent pattern in catenation normal form. To obtain this normal form row or column concatenation closures are replaced by right recursion. For instance

$$Pattern \quad \rightarrow \quad (Label \oplus Data)^{*\ominus}$$

becomes

$$\begin{aligned} Pattern &\rightarrow A \\ A &\rightarrow Label \oplus Data \\ (A &\rightarrow Label \oplus Data) \ominus A \end{aligned}$$

This strategy is applied repeatedly until the grammar does not contain any closure symbol.

2.4.5 Filtering

In two-dimensional pattern matching filtering is a widely used approach [Bak78; Bir77; Zhu89]. Filtering reduces the problem of finding a pattern P in an array T to finding a substring p in string t, where a detected match of p in t corresponds to a possible match of P in T. We use this idea by calculating all possible first terminals of a given pattern grammar. Next we determine whether there is a detected bounding box that has this symbol in its upper-left corner. We only start parsing the given pattern for these bounding boxes.

2.4.6 Parsing

To determine whether a given bounding box complies with a given pattern Q, we apply a recursive descent parsing strategy with some small modifications to handle two-dimensional structures.

Algorithm 1 provides an outline of our approach. This procedure takes a bounding box B and a pattern grammar G as its parameters. We begin with the start symbol of the grammar and expand it until the first symbol is a terminal.

Expanding means replacing the left-hand side of a production rule by a corresponding right-hand side (Algorithm 1, lines 7-12). If a non-terminal occurs as the left-hand side of multiple production rules, all of them are evaluated. If a terminal is found, we examine whether this is the expected terminal. If it is, the parsing continues, otherwise it fails (Algorithm 1, lines 14-19).

This process requires some administration, for which we introduce a datatype Position, containing an x- and y-coordinate, representing a position in the spreadsheet, and a string T that has to be parsed at that position. The set S represents all Positions that still have to be evaluated. At the start of Algorithm 1, S is initialized to contain only one Position with coordinates (i, j) and string X. (i, j) is the upper left corner of the bounding box B and X is the start symbol of grammar G. The set T represents all successfully parsed patterns, so if parsing succeeds for a given pattern at a given position, this position is added to the set T.

The evaluation of a Position P is done in the body of the While-loop on lines 6-21 and works as follows. If P starts with a non-terminal, say Y, a new Position is added to S for every possible right-hand side r belonging to Y. That new Position has the same coordinates as P, and contains string T, where the first occurrence of Y is replaced by right-hand side r. Since S now contains all possible scenarios from the parsing of P, we can remove P from S. Evaluation continues with the remainder of set S.

If P starts with a terminal, say t, the actual parsing happens. (Lines 14-19) We determine if the symbol at position (x, y) is equal to t. If that is not the case, parsing for this particular position fails, and P is removed from S. If (x, y) is equal to t, t is removed from T. Since terminals are always followed by a catenation symbol, the first symbol of T is a catenation symbol, say c. The cursor is then moved according to c. If this is a column catenation symbol, the cursor moves to the right, if it is a row concatenation, the cursor moves downward. This moving is aware of the size of the bounding box and will fail if the cursor is about to go out of bounds. After that the evaluation continues with the modified P.

2.5 From patterns to class diagrams

2.5.1 Using annotations

Given a pattern and a spreadsheet, the two-dimensional parsing algorithm just described can identify rectangles in the spreadsheet that match the pattern. The result is a *parse tree*, representing the full content of the rectangle, as well as the hierarchy of productions applied to match the rectangle to the pattern.

In order to turn this parse tree into a class diagram, our primary interest is in the *structure* of the spreadsheet, not in the actual cell contents. Therefore, our next step consists of identifying those nodes in the parse tree that can help to reveal this structure. We do so by offering the user of our tool the possibility to add

Algorithm 1 Two-dimensional Parsing(BoundingBox B, Grammar G)

```
1: Position P ← (B.i, B.j, G.X)
2: Set T ← ∅
3: Set S ← {P}
4: while S ≠ ∅ do
5:   P ← a position from S
6:   while P.T ≠ "" do
7:     if FirstSymbol(P.T) is non-terminal Y then
8:       for all Productions Y → r do
9:         Create new Position P_l,
10:            with x = P.x, y = P.y and T = r · Rest(P.T)
11:        Add P_l to S
12:      end for
13:    else
14:      if FirstSymbol(P.T) == B(P.x,P.y) then
15:        Remove First Symbol of P.T
16:        Move Cursor according to FirstSymbol(P.T)
17:      else
18:        Remove P from S {Parsing fails for this position}
19:      end if
20:    end if
21:  end while
22:  Add P to T
23:  Remove P from S
24: end while
```

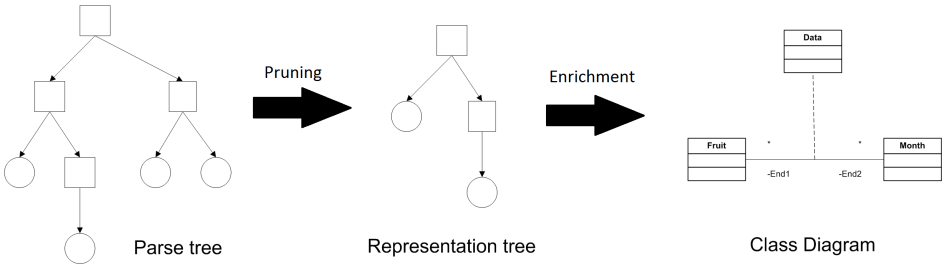


Figure 2.5: *Overview of the Transformation step to transform a parse tree into a class diagram*

	A	B
1	**Customer**	
2	Surname	McLean
3	Adress	Hillstraat 7
4		

Figure 2.6: *Simple Customer spreadsheet*

annotations to pattern definitions, which will subsequently guide a transformation of the parse tree into class diagram.

Using annotations to turn a parse tree into a class diagram is done in two steps, as depicted in Figure 2.5. First, the annotations are used to *prune* the parse tree into a *representation tree* only containing relevant nodes. Next, this representation tree is *enriched* so that a meaningful class diagram can emerge.

To see annotations in action, consider the simple spreadsheet shown in Figure 2.6. In this spreadsheet, one can recognize a class "Customer", with fields "Surname" and "Address". Thus, we define a pattern that can recognize spreadsheets like this one:

$$
\begin{aligned}
G: & \\
\mathit{Pattern} &\rightarrow \mathit{Headerrow} \ominus (\mathit{Datarow}^{*\ominus}) \\
\mathit{Headerrow} &\rightarrow \texttt{Label} \oplus \texttt{Empty} \\
\mathit{Datarow} &\rightarrow \texttt{Label} \oplus \texttt{Data}
\end{aligned}
$$

In this pattern definition, the classname can be obtained from the headerrow, and the field names from the data rows. Thus, we add *annotations* to capture

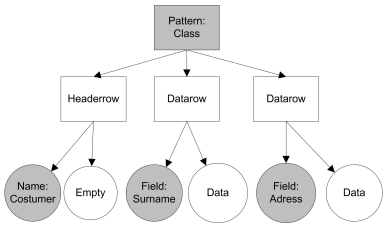


Figure 2.7: *Parse tree generated for the Customer spreadsheet*

exactly this:

$$
\begin{aligned}
G: & \\
Pattern : class &\rightarrow Headerrow \ominus (Datarow^{*\ominus}) \\
Headerrow &\rightarrow \texttt{Label} : name \oplus \texttt{Empty} \\
Datarow &\rightarrow \texttt{Label} : field \oplus \texttt{Data}
\end{aligned}
$$

Here we see that the `Label` of a header row represents the name of the *class*, and that the `Label` of a data row represents the *field* name.

Annotations are permitted on both terminals and non-terminals. For terminals they are an indication that the content of the cell contains relevant information (such as the name of a field). For non-terminals they are an indication that the non-terminal in question should be kept in the representation tree. Note that an annotation for the root is not required: Hence the result of the transformation can be either a tree or a forest.

Algorithm 2 Tree transformation

1: Remove all non-annotated leaves
2: Remove all non-annotated nodes without annotated descendants
3: Remove all non-annotated nodes, their (annotated) children become children of their lowest annotated ancestor

The annotations can be used to prune the parse tree as described in Algorithm 2. The original parse tree, with annotations marked in grey, is depicted in Figure 2.7; the corresponding pruned representation tree is shown in Figure 2.8.

2.5.2 Class diagrams

In this chapter, the output of the transformation step are class diagrams. Therefore, the annotations that can be used to define the transformation represent the basic building blocks of class diagrams: *class*, *name*, *field*, and *method*. Since the

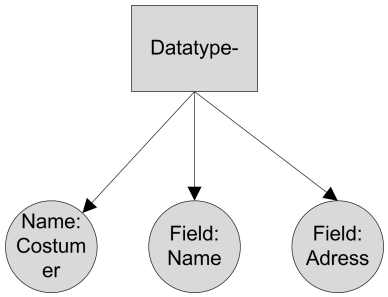


Figure 2.8: *Representation tree after the transformation of the Customer parse tree*

latter three are properties of a Class, they can be used as annotations of terminals, and Class itself occurs as annotation of non-terminals. Referring to the Customer example the class diagram contains a class `Customer` with two fields `Surname` and `Adress`.

2.5.3 Enrichment

The transformation step results in one or more classes. In most cases there is a relation between the different classes. In the second part of the annotation the relation between the resulting trees can be expressed. The following relations can be defined in the pattern:

Association(C_1, C_2, m_1, m_2, C_3) This defines an association between two classes, C_1 and C_2. Optionally, we can define multiplicities m_1 and m_2 for this association. The last argument is again a class and represents an association class of this association.

Merge(C_1,C_2) The operation merges two classes into one class containing all fields and methods of both classes. If fields or methods with equal names are encountered both will appear in the new class. To distinguish between them their original class name will be appended to the method of field name. The Merge-operation is useful if class information is scattered around the spreadsheet and can not be easily captured within one production rule.

Reference(C_1,C_2) A reference is used in the creation of the class diagram, but will not appear in the class diagram itself. A reference between two classes is added when information about the names of fields and methods of C_1 can be found in C_2. This is used when information from the spreadsheet has to be used in multiple classes.

We will see examples of the use of these relation declarations in the next section.

	A	B	C	D
1	**Number**	**Name**	**Position**	**University**
2	1	Roy D. Yates	Director	Rutgers, The State University of New Jersey
3	2	Andy Ogielski	Professor	Rutgers, The State University of New Jersey
4	3	Christopher Rose	Professor	Rutgers, The State University of New Jersey
5	4	Peter Voltz	Professor	Polytechnic University of New York
6	5	Frank Cassara	Professor	Polytechnic University of New York
7	6	V. John Mathews	Professor	University of Utah
8	7	Kamilo Feher	Professor	University of California, Davis
9	8	G. Rick Branner	Professor	University of California, Davis
10	9	Neville C. Luhmann, Jr.	Professor	University of California, Davis (Livermore)
11	10	Michael B. Pursley	Professor	Clemson University
12	11	Chalmers M. Butler	Professor	Clemson University
13	12	Anthony Q. Martin	Professor	Clemson University
14	13	Donald C. Cox	Professor	Stanford University
15	14	Theodore S. Rappaport	Professor	Virgina Polytechnic Institute and State University
16	15	Nikil S. Jayant	Professor	Georgia Institute of Technology
17	16	Hiroyuki Morikawa	Professor	University of Tokyo

Figure 2.9: *Simple class spreadsheet pattern*

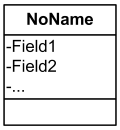


Figure 2.10: *Class diagram extracted from a Simple class spreadsheet pattern*

2.6 Spreadsheet patterns

By inspecting the EUSES spreadsheet corpus [Fis05b], and looking at related work in spreadsheet design [Jan00; Kni00; Pan94; Ron89] we have identified a number of reoccurring patterns in spreadsheets into a *pattern library*. In this section, we describe the syntax of a selection of the most interesting spreadsheet design patterns in this library.

2.6.1 Simple class

The simplest pattern in the library is a list of instances as shown in Figure 2.9. The column headers provide the names of the fields, as captured in the following pattern.

$$\begin{aligned} \mathit{Pattern} : \mathit{class} &\rightarrow X^{*\circledcirc} \\ X &\rightarrow \texttt{Label} : \mathit{Field} \ominus \texttt{Data}^{*\ominus} \end{aligned}$$

Note that the spreadsheet content provides no clue about the name of the class. The class diagram that corresponds to this pattern is shown in Figure 2.10.

2.6.2 Simple class including name

If there is a row above a simple class pattern with only one label, we assume this is the name of the resulting classs, as described by the second pattern in the library.

$$\begin{aligned} Pattern : class &\rightarrow \texttt{Label} : name \oplus \texttt{Empty}^{*\oplus} \ominus X^{*\oplus} \\ X &\rightarrow \texttt{Label} : Field \ominus Data^{*\ominus} \end{aligned}$$

2.6.3 Simple class including methods

If one of the columns contains formula's, this column is likely to represent a method of the resulting class. To include this, we add the following production rule to the simple class pattern (with or without class name).

$$X \rightarrow \texttt{Label} : method \ominus \texttt{Formula}^{*\ominus}$$

2.6.4 Aggregation

If there is a formula below a simple class, this represents a calculation over all instances, which we catch in a Reporter Class that references all the instances. For each Formula we encounter, we introduce a nameless method in the reporter class.

$$\begin{aligned} Pattern &\rightarrow Simple \ominus Reporter \\ Simple : class &\rightarrow \texttt{Label} \oplus \texttt{Empty}^{*\oplus} \ominus X^{*\oplus} \\ X &\rightarrow \texttt{Label} : field \ominus \texttt{Data}^{*\ominus} \\ X &\rightarrow \texttt{Label} : method \ominus \texttt{Formula}^{*\ominus} \\ Reporter : class &\rightarrow \texttt{Label} \oplus \texttt{Formula}^{*\ominus} : method \end{aligned}$$

The methods of the Reporter class are empty in this case, but they correspond one-to-one with the names of the fields of the simple class. Therefore a Reference clause is added, in the enrichment step the names will be copied to the Reporter class. The relation between the simple class and the reporter class is defined by the Association clause.

$$\begin{aligned} &Reference(Reporter, Simple) \\ &Association(Simple, Reporter, *, 1) \end{aligned}$$

All of the above patterns can also occur vertically, rotated 90 degrees. Figure 2.6 shows an example of a spreadsheet in which the rotated version of the "Simple

class" pattern is applied. Our library also contains the vertical variants of all above patterns.

2.6.5 Associated data

The final pattern in our library concerns two different classes, with data associated to both of them. This pattern represents two classes with an association between the two classes that contains the data.

$$\begin{aligned} \mathit{Pattern} &\rightarrow C_1 \oplus (C_2 \ominus C_3) \\ C_1 : \mathit{class} &\rightarrow \texttt{Empty} \ominus \texttt{Label}^{*\ominus} \\ C_2 : \mathit{class} &\rightarrow \texttt{Label} \oplus \texttt{Empty}^{*\oplus} \\ C_3 : \mathit{class} &\rightarrow (\texttt{Label} \ominus \texttt{Data}^{*\ominus})^{*\oplus} \end{aligned}$$

The relation between the classes is defined by the following clause.

$$\mathit{Association}(C_1, C_2, *, *, C_3)$$

Furthermore, there could also exist aggregations over this data, like in the fruit example in Figure 2.1. In that case we add an association between a Reporter class containing the aggregation methods and the association class, as shown in Figure 2.2. We model this in the representation tree as an Association with methods. To recognize this the following pattern is included in the library. This pattern also occurs in one direction only, in that case either D_1 or D_2 is omitted.

$$\begin{aligned} \mathit{Pattern} &\rightarrow (C_1 \oplus (C_2 \ominus C_3) \oplus D_1) \ominus D_2 \\ C_1 : \mathit{class} &\rightarrow \texttt{Empty} \ominus \texttt{Label}^{*\ominus} \\ C_2 : \mathit{class} &\rightarrow \texttt{Label} \oplus \texttt{Empty}^{*\oplus} \\ C_3 : \mathit{class} &\rightarrow (\texttt{Label} \ominus \texttt{Data}^{*\ominus})^{*\oplus} \\ D_1 : \mathit{class} &\rightarrow \texttt{Label} : \mathit{method} \ominus \texttt{Formula}^{*\ominus} \\ D_2 : \mathit{class} &\rightarrow \texttt{Label} : \mathit{method} \oplus \texttt{Formula}^{*\oplus} \end{aligned}$$

In this case, the classes D_1 and D_2 both represent the Reporter Class, but it is not possible to catch them within one production rule because of the structure of the spreadsheet. Therefore, we need a Merge-clause in this case. Furthermore one more association needs to be defined, between the reporter class and the association class.

$$\begin{aligned} \mathit{Merge}(D_1, D_2) \\ \mathit{Association}(C_1, C_2, *, *, C_3) \\ \mathit{Association}(C_3, D_1, *, 1) \end{aligned}$$

2.7 Implementation

The approach for extracting class diagrams from spreadsheets as described in the previous sections has been implemented in the Gyro tool suite[1], targeting Microsoft Excel spreadsheets. Gyro users can specify the directory they want to analyze. Gyro loads all .xls and .xlsx files from that directory and ignores other files. Furthermore the user can specify in which directory the patterns can be found. Patterns are just plain text files containing a pattern grammar. Options of the current implementation include the coloring of a spreadsheet representing the basic cell classification, the search for patterns within spreadsheets, the visualization of the parse tree and the pruned parse tree, and the full transformation into class diagrams.

Gyro is subject to continuous development; at the moment we are in the process of enriching the Gyro user interface, and providing a web interface in which a spreadsheet can be simply uploaded, relevant spreadsheet patterns can be selected, and spreadsheets can be analyzed "as a service."

Gyro is implemented in C#.net using Visual Studio 2010, beta 1. We make use of the Microsoft SQL Server Modeling platform (formerly "Oslo") and its MGrammar language to specify the grammar for our pattern definitions.

2.8 Evaluation

We evaluated the strength of our approach by testing the Gyro approach on the EUSES Spreadsheet Corpus [Fis05b]. The evaluation was twofold, first we tested the quality of the patterns, by determining how often the chosen patterns occur in the Spreadsheet Corpus. This way we can check whether the patterns we chose are really frequently used. Secondly, for the patterns that were found, we checked whether the generated class-diagram is a faithful representation of the underlying domain. This second evaluation was done by comparing generated class diagrams to class diagrams that were manually created.

2.8.1 The data set

The EUSES Spreadsheet Corpus is a set of spreadsheets created to help researchers to evaluate methodologies and tools for creating and maintaining spreadsheets. It consists of 4498 unique spreadsheet files, divided into 11 categories varying from financial spreadsheets to educational ones. Many papers on spreadsheet analysis use the Corpus to test their methodologies and algorithms, among which are [Abr06] and [Cun09b].

[1] Gyro is currently part of the Breviz app, see Appendix A

Table 2.1: *Number and percentage of spreadsheets of the EUSES Spreadsheet Corpus that can be processed with the current Gyro pattern library*

Type	Number of sheets	Patterns found	Success percentage
Cs101	10	4	40.0%
Database	726	334	46.0%
Filby	65	31	47.7%
Financial	904	334	36.9%
Forms3	34	14	41.2%
Grades	764	307	40.2%
Homework	804	375	46.7%
Inventory	895	125	14.0%
Jackson	21	7	33.3%
Modeling	692	334	48.3%
Personal	7	6	85.7%

2.8.2 Quality of chosen patterns

The first part of the evaluation focusses on measuring the number of spreadsheets that can be processed with our pattern library. For this we applied Gyro to the Corpus and counted the number of worksheets which in which at least one pattern could be applied. A worksheet is an individual sheet within a spreadsheet. We ignored protected, empty and invisible worksheets, as well as worksheets containing a VBA-macro. The results of this experiment can be found in Table 2.1. The results indicate that the patterns we chose are indeed quite common. There are 4 categories with a score in the 45-50% range, 3 in the 40-45% range, 3 in the <40% range and there is one (small) category scoring 85%. We noticed that spreadsheets originating from the inventory and financial categories score lower than spreadsheets from the other categories. The inventory category mainly consists of *empty* forms, spreadsheets in which the data still has to be filled in. Since our approach focusses on spreadsheets filled with data, the algorithm logically performs less on these inventory-sheets. The lower score on financial spreadsheets is probably caused by the complexity of financial spreadsheets in general. The lack of financial knowledge of the author of this dissertation could also play a role. Since we are no financial experts, we do not have knowledge of common financial structures that could occur within spreadsheets, making it more difficult to design suitable patterns for this category.

2.8.3 Quality of mapping

The second evaluation measures the quality of extracted class diagrams. For this we randomly selected 50 spreadsheets from the EUSES Spreadsheet Corpus in which one of the Gyro patterns occurs. We divided these 50 sheets among three researchers of the Software Engineering Research Group of Delft University of

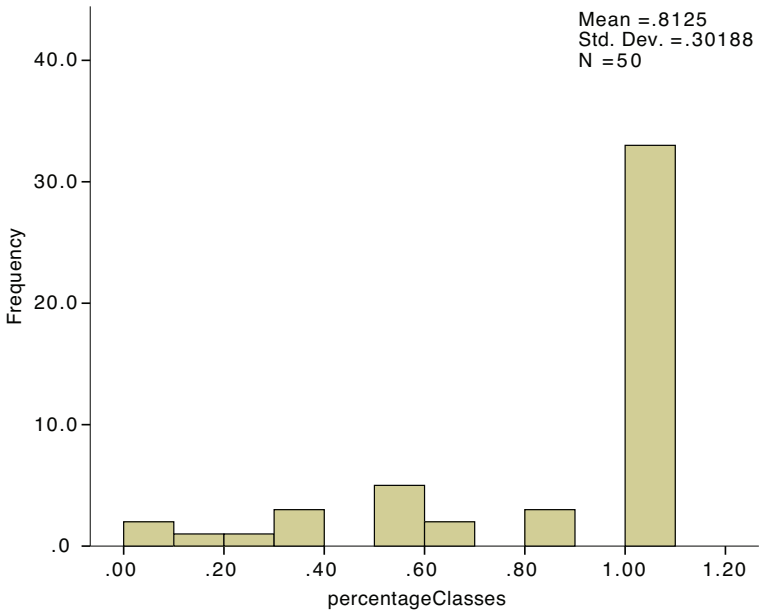


Figure 2.11: *Histogram of correctly matched classes*

Technology, one of which was the author of this dissertation, the other two are the co-authors of a previously published version of this chapter [Her10]. For these spreadsheets, each researcher created a class diagram by hand, by looking at the structure and content of the spreadsheet. We refer to these class diagrams as *reference class diagrams*. They were created without looking at the generated class diagrams. In this evaluation we compared the generated class diagrams to the reference class diagrams and counted the number of matched classes, fields and methods.

Figures 2.11, 2.12 and 2.13 depict the results of this experiment. In the majority of the spreadsheets (32), Gyro found all classes and in about half of the cases all fields and methods were also extracted correctly. In the most cases in which the right pattern was selected, all methods and fields were correct. In Figure 2.13,we see that there is a significant number of spreadsheets in which no methods were found. This is mainly due to the fact that values in the spreadsheet can have a numerical relation that did not result from a formula. This typically occurs when data is imported into a spreadsheet from another software system where the value was calculated. A human quickly notices this relationship and decides that a column represents a method. Since Gyro only takes into account the cell types, it does not recognize this implicit relationship. We keep this as a point for future work.

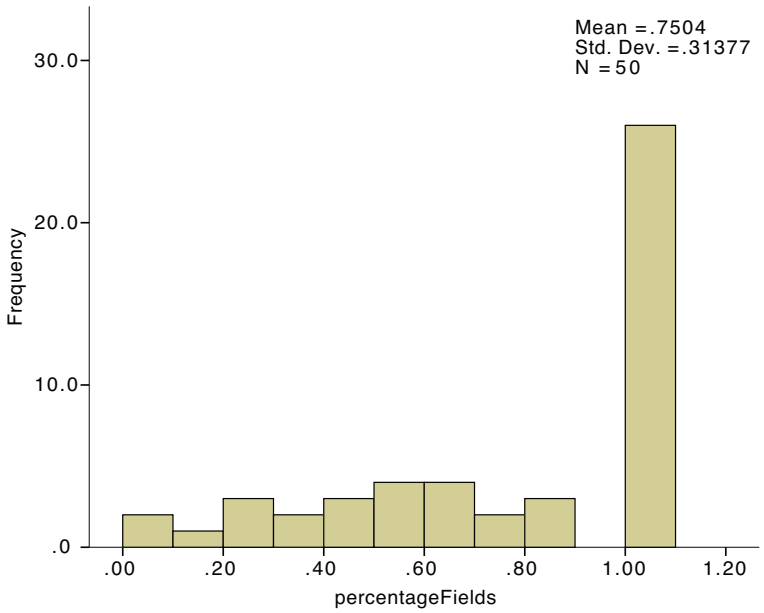


Figure 2.12: *Histogram of correctly matched fields*

Furthermore, we divided the results into four categories to get an overall view on the quality of extracted class diagrams. These categories are:

Perfect All classes, fields and methods correspond

Structure OK All classes are present, but their names, fields or methods are missing or incorrect

Classes missing Some classes are present, but others are missing

Useless The generated diagram does not correspond to the the reference diagram at all

This results of this evaluation are listed in Table 2.2. In 20 cases Gyro produced exactly the same class diagram as the one created by the authors. In 13 cases the structure was correct. There were 6 cases in which the result generated by Gyro was classified useless. We believe these results are promising since Gyro performs just as well as a human on 40% of the spreadsheets. However there are also some generated class diagrams that do not represent the underlying domain. These spreadsheets contained difficult calculation structure the Gyro toolkit is not able to process yet, like financial calculations or use a very specific layout that does not occur in the pattern library.

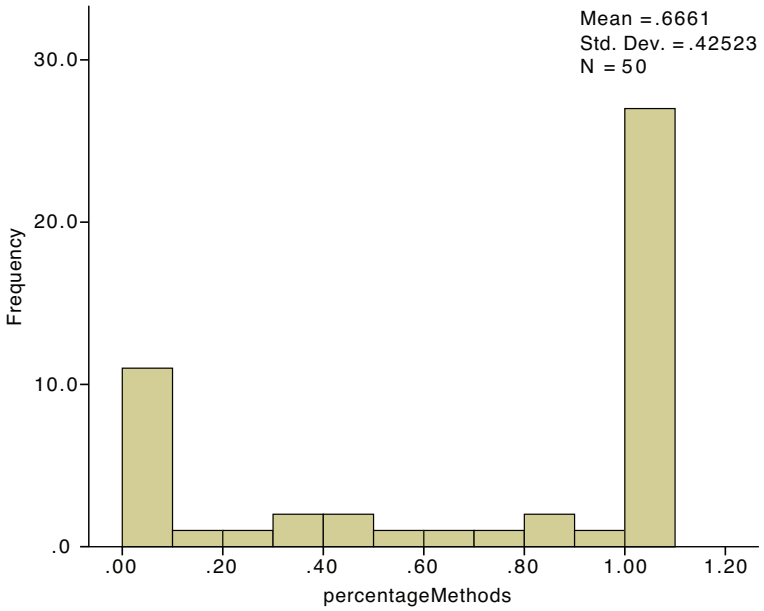


Figure 2.13: *Histogram of correctly matched methods*

Table 2.2: *Overall view on the quality of extracted class diagrams*

Number of sheets	Perfect	Structure OK	Classes missing	Useless
50	20	13	11	6

2.9 Discussion

The current implementation of Gyro, while still a prototype, enables software engineers to derive domain knowledge, in the form of a class diagram, from a collection of spreadsheets. In this section, we discuss a variety of issues that affect the applicability and suitability of the proposed approach.

2.9.1 Spreadsheet limitations

There are some spreadsheet concepts the current implementation can not handle properly. Most importantly Gyro uses only one spreadsheet as input for the detection algorithms. There often exist multiple spreadsheet files with a similar structure within a company. Imagine a spreadsheet form to record the data of an order. There will probably be one file or worksheet for every order, all having the same structure. The performance of our methods could be improved by applying the pattern matching techniques described in this chapter to multiple instances. This approach has several benefits. A higher degree of certainty could be achieved about the type of a cell. If a cell contains the same value in multiple spreadsheet files, it is very likely to be a label. There is also the benefit of gathering more information making it possible to do statistical analysis on the results. Furthermore, the current cell classification of only four cell types is quite coarse-grained. Improvements could for instance be made by refining the `Data` type into `Text` data and `Number` data.

2.9.2 Class diagram limitations

The most important class diagram feature that is currently not supported by Gyro is the inheritance relationship between classes. Inheritance is a difficult design concept per se and is typically not modeled in spreadsheets explicitly. This is also the main reason why Gyro can not extract it directly from spreadsheets using its pattern library. However, there exist approaches that can derive inheritance relationships by normalizing class diagrams. Such an approach is, for example, the FUN-algorithm [Nov01], which we plan to integrate into Gyro.

2.9.3 Beyond class diagrams

In the current research we only focussed on creating class diagrams, but the information contained in spreadsheets could also be represented in a different format. Possibilities include generating a database scheme, generating General Purpose Language code or generating more specific code, like WebDSL [Gro08] code. The extracted domain information could also be used to create a domain-specific language tailored towards a particular business domain, since important domain concepts are present in the output of our algorithm.

We also envision the use of our algorithms to support the migration of a spreadsheet-based style of working to one where a commercial ERP (Enterprise Resource Planning) or CRM (Customer Relationship Management) solution is used. This requires understanding to what extent the data and business processes reflected in the spreadsheets can be mapped onto the models imposed by the ERP or CRM tools, for which we expect that our approach can deliver very useful information.

2.9.4 Dealing with spreadsheets errors

As we mentioned in the introduction, spreadsheets are hardly ever free of errors. This could raise the question how our approach deals with errors in spreadsheets. We believe errors mostly occur within formula's; the wrong cells might be referenced or a copy-paste-error is made. Since the cells still are Formula typed in this case, this does not concern the patterns and our methods will still be able to find the right pattern. It would be interesting to compare the results of error-finding algorithms on the EUSES spreadsheet with our results to gain more confidence in this assumption. Furthermore our approach allows the option to introduce a tolerance for minor errors within the pattern, so if some cells do not fit within the pattern, it can still be recognized. More research is needed however to determine a tolerance level that is neither to strict, nor too tolerant.

2.9.5 Meaningful identifiers

Spreadsheet do not always contain all information needed to create a perfect class diagram. Consider the spreadsheet from the introduction again, in Figure 2.1. In the spreadsheet the value of the numbers is not expressed. We might assume it is the amount of fruit sold in a particular month, but it could also be the price of the fruit or the number of customers that the farmer sold to.

Because not all information in the spreadsheet is named, the naming of methods and fields is not always perfect. We do not believe this is a big problem, because the class diagram still reveals the structure of the data if some names are missing or incorrect. in this case Gyro sketches the basic diagram and users can fill in the blanks.

2.9.6 Threats to validity

A threat to the external validity of our evaluation concerns the representativeness of the EUSES Corpus spreadsheet set. This set, however, is large (over 4000 spreadsheets), and is collected from practice. Furthermore, for the manual comparison of the class diagrams we randomly picked 50 spreadsheets from this set.

With respect to internal validity, one of the threats is the fact that the reference class diagrams were only created by three people, who were also involved in the

research. The outcome might have been different if other people had created the reference diagrams, because experience, education and personal taste influence how a person creates class diagrams. This effect can be decreased by using a larger test group in future experiments. We however believe the current test group serves as a good reference group, as the persons involved all have years of experience in software engineering and modeling. Furthermore, the collection of 50 spreadsheets and accompanying class diagrams is available from our web site, allowing other researchers to challenge our reference set.

With respect to threats to reliability (repeatability), the Gyro tool, the pattern library used, the selected spreadsheets from the spreadsheet corpus and the reference set of class diagrams are available from our web site, enabling other researchers to redo our experiments.

2.10 Related work

Spreadsheets analysis is a subject of ongoing research. Most papers in this field focus on testing spreadsheets and certifying their correctness. Abraham and Erwig have written a series of articles on *unit* inference [Abr04; Abr06; Abr09; Abr07b]. Their units form a type system based on values in the spreadsheet that is used to determine whether all cells in a column or row have the same type. Their work was of inspiration to us, but their objectives differ from ours. They focus on error finding, where we aim at extracting information. However, their *hex* and *vex* groups - similarly shaped rows and columns - inspired us in defining patterns in the library. Ahmad *et al.* [Ahm03] also created a system to annotate spreadsheets, however their approach requires users to indicate the types of fields themselves. Mittermeir and Clermont [Mit02] investigate the possibility of structure finding, but their aim again was localizing errors by finding discrepancies within the structures found.

Another group of papers presents best practices in spreadsheet design [Jan00; Kni00; Pan94; Ron89], which gave us more insight into the patterns that had to be included in our library. Fisher *et al.* [Fis02] suggest a systematic approach to building a spreadsheet, but their methods are used to create spreadsheet from scratch, and not to analyze existing ones.

Besides the goal also the approach of existing work differs from ours. Where existing papers recognize structure bottom up, by building up their knowledge of the spreadsheet cell by cell, we apply a more top-down approach, by checking whether a spreadsheet complies with a given structure.

For the recognition of patterns our initial approach was to use techniques from two-dimensional pattern matching. However they match patterns of fixed size within an array. Although this does not suit our purpose, these algorithms did provide us with some valuable ideas. For instance, there is a class of *filter-based algorithms*, like Baker [Bak78], Bird [Bir77] and Takaoka-Zhu [Zhu89]. This class of algorithms is based on the reduction of two-dimensional matching to one

dimension. Another useful principle we took from existing algorithms is the notion of *approximate matching*, where small differences between patterns and arrays are allowed. In this variant of matching an integer k is introduced that indicates how many mismatches are allowed with respect to a given distance, like for instance the Levenshtein distance [Lev75].

Because the two-dimensional pattern matching approach was not applicable in our case, we started to investigate two-dimensional parsing. There have been some attempts to generalize parsing to two dimensions. First there is *array parsing* [Ros87]. In this paradigm, rewriting may be applied to subarrays of the equal size. Because of that property it is never possible to generate an array from a single start symbol. Hence the use of these grammars does not solve the problem of variable size matching. Secondly, there are *matrix grammars* [Str72], which generate arrays in two phases, a horizontal and a vertical phase. Although they are better applicable than array grammars, matrix grammars are not able to recognize patterns that are a combination of rows and columns.

The idea to transform spreadsheets into databases has been described in [Cun09b]. Their goal is to transform spreadsheets into relational databases by using the FUN algorithm [Nov01] to find functional dependencies within rows. Therefore their approach is limited to spreadsheet resembling non normalized databases.

Due to the use of spreadsheets as simple databases, our work also connects to the problem of Object-Relational Mapping. In particular, we attempt to map from two-dimensional relations back to object structures.

Last but not least, reverse engineering class diagrams from regular programs written in, e.g., Java has been studied extensively. An overview is provided by [Kol02], who also include a comparison with existing roundtrip engineering tools. The key problem in these approaches is to reverse engineer class associations from class implementations, which differs from our purpose of extracting class diagrams from spreadsheet logic.

2.11 Concluding remarks

The goal of this chapter is to underline the importance of spreadsheet analysis as a means to better understand the business domain of companies and users. To that end we have designed an approach to describe common spreadsheet design patterns, and we implemented the Gyro tool to extract the domain information automatically. The key contributions of this work are as follows:

- A notation for expressing spreadsheet patterns, as well as two-dimensional parsing algorithm capable of recognizing these patterns (Section 2.4);
- A systematic approach to transform recognized patterns into class diagrams (Section 2.5);

- A library of frequently occurring patterns (Section 2.5);
- An implementation of the proposed methods and library in the Gyro system (Section 2.7);
- An evaluation of the proposed approach on a corpus of over 4000 spreadsheets (Section 2.8).

The results of our evaluation with the EUSES Spreadsheet Corpus showed that Gyro can extract valuable domain information from 40% of the given spreadsheets. The evaluation further showed that extracted class diagrams are of reasonable quality—out of 50 spreadsheets 20 class diagrams were extracted perfectly, 13 contained minor flaws, in 11 cases classes were missing, and in only 6 cases the extracted class diagrams were rated as useless. This clearly underlines the potential of the Gyro approach.

We see several avenues for future research. First the description of patterns could be improved. Pattern grammars might be a convenient way of describing spreadsheet patterns for users with experience in programming and formal languages, but it is probably not that easy for users from the business domain. To make Gyro easier for this kind of users, we intend to create a visual editor for patterns. Furthermore spreadsheets do not have to be replaced by software in all cases. A possible other use for Gyro could be to aid users in creating structured spreadsheets, by offering pattern-based edit assistance, comparable to the discovery-based assistance in [Cun09b]. Finally we have several ideas to speed up the implementation of the algorithm. For instance, the filter-based part of the algorithm now only checks the left-most upper-most cell of the pattern. It might be better to look at the first row or the first column or a combination of both, to determine earlier that there is no match. The current parsing approach is recursive descent, which is known to be very inefficient in some cases. We would like to explore the possibilities of using an LR-like parsing strategy on the recognition of pattern grammars.

Chapter 3

Supporting Professional Spreadsheet Users by Generating Leveled Dataflow Diagrams

3.1 Introduction

Spreadsheets are widely used in industry: Winston [Win01] estimates that 90% of all analysts in industry perform calculations in spreadsheets. Their use is diverse, ranging from inventory administration to educational applications and from scientific modeling to financial systems. The financial business is a domain where spreadsheets are especially prevailing. Panko [Pan06] estimates that 95% of U.S. firms, and 80% in Europe, use spreadsheets in some form for financial reporting.

Business analysts using spreadsheets usually have very limited (if any) training as a programmer. In spite of that, they effectively are *end-user programmers*, and as such face many of the challenges of professional developers, such as identifying faults, debugging, or understanding someone else's code [Ko10].

This chapter aims at providing support for spreadsheet users to take on these end-user programming challenges. To that end, we first study the problems and information needs of professional spreadsheet users and then present an approach that presents this information in a compact and easy to understand way, with leveled dataflow diagrams.

The context, in which we conduct our research, is formed by the analysts of Robeco, a Dutch asset management company with approximately 1600 employees worldwide, and over 130 billion Euro worth of assets under management. In a survey we conducted among 27 of their analysts, we found that they use Excel for

an average of 3 hours a day, underlining the large role spreadsheets play in their daily work. Furthermore, spreadsheets have an average lifetime of more than five years, and individual spreadsheets are used by 13 different analysts on average.

In order to support analysts in their work with spreadsheets, we start by identifying *information needs*. In particular, we describe how we conducted interviews with the same group of 27 analysts, using a *grounded theory* [Gla67] approach, to obtain understanding of their use of spreadsheets. In the interviews, *spreadsheet transfer scenarios* (to a new user, to an auditor, and to a professional developer creating custom software) were identified as problematic. In those scenarios, end-users search for a better insight into the dependencies between cells, formulas, groups of cells and worksheets.

To meet these demands, we propose an approach for the automated extraction of *dataflow diagrams* from spreadsheets. Such diagrams can be used to visualize data, processes manipulating data, and dependencies between them. Furthermore, *leveled* diagrams can be used to accommodate hierarchies, for example for blocks of cells or worksheets.

We implemented our approach in a tool called *GyroSAT*, which can generate dataflow graphs from Excel spreadsheets. To evaluate our approach we conducted a series of evaluations in the concrete setting of Robeco. First, we interviewed the same group of 27 spreadsheet users, analyzing how the tool could help in transferring their own spreadsheets to colleagues. Furthermore, we conducted nine case studies, three for each type of transfer task identified. The findings of these evaluations indicate that (1) spreadsheet professionals from Robeco consider the tool as helpful; (2) the visualizations derived help to create a *story line* to explain spreadsheets to colleagues in transfer tasks; (3) the visualizations scale well to large and complex spreadsheets in use at Robeco.

This chapter is organized as follows. In Section 3.2, we describe the interviews leading to the information needs of spreadsheet users. Then in Section 3.3, we provide background information on dataflow diagrams, after which we describe our algorithm to derive dataflow diagrams in Sections 3.4 and 3.5. In Section 3.6, we cover our implementation, which we use in the evaluation described in Section 3.7. We conclude with a discussion of our results, an overview of related work, and a summary of our contributions and directions for future research.

3.2 Spreadsheet information needs

To gain understanding of the problems around spreadsheet usage, we conducted a survey at Robeco. Robeco is a Dutch asset management company with approximately 1600 employees worldwide, of which 1000 work in their headquarters in Rotterdam, where we performed our survey.

The survey was conducted following the *grounded theory* approach. Grounded theory is a qualitative research method that discovers theory from data [Gla67]. When using grounded theory, observations are documented and subsequently

coded into categories. During this coding process, which takes place after each observation—as opposed to after all observations—a theory might emerge [Ado08].

Grounded theory is a method especially suited for discovering problems that exist for the participants. Therefore, in a grounded theory study, the researcher works with a general area of interest rather than with a specific problem [McC03].

The area of interest in our study is the use of spreadsheets in industry. To investigate this area we conducted a survey amongst employees of Robeco using semi-structured interviews. For each interview, we kept a memo of statements, questions asked and answers provided by the employee. Following the grounded theory approach we continuously analyzed the memos of each day and used the results to direct the following interviews.

A total of 47 employees was invited to participate in the survey, with varying spreadsheet knowledge and expertise, working in different departments, making it a maximum variation sample [Mar96]. Of the 47 invitees, 27 participated in the interviews. The interviews were performed over a time period of 4 weeks, with 1 or 2 interviews per day. We started the interviews by asking subjects about the role of spreadsheets in their daily work and let subjects tell their own story.

During the course of the first interviews, many subjects stated that problems with spreadsheets occur in *spreadsheet transfer scenarios*. In such a scenario, a spreadsheet is transferred from one employee to another, for instance when an employee leaves the company and a colleague has to start working with his spreadsheet.

From that point on, we started asking subjects in interviews whether they experienced such scenarios. We found that the vast majority (85%) agreed that they often transferred spreadsheet to a colleague. We coded all described scenarios into the following three categories:

S1 A spreadsheet has to be transferred to a colleague. This scenario typically occurs when, for example, a new employee starts working for the company, an employee leaves the company or when a new spreadsheet is created that has to be used by an other employee.

S2 A spreadsheet has to be checked by an auditor. Auditing periodically happens at large companies, to determine whether spreadsheets in key processes are well designed, error free and well documented.

S3 A spreadsheet is being replaced by custom software. When a spreadsheet created by an end-user, becomes so complex that it no longer complies to standards in the company such as safety, access control or readability, it is transformed into a custom software system.

To determine ways to support users in these scenarios, we dived deeper into the problems occurring during a transfer scenario. As it turns out, problems typically occur when the receiving employee has too little understanding of the spreadsheet. The most of the subjects (70%) expressed that they had difficulties

with understanding a spreadsheet they received from a colleague. Zooming in even further, we identified *information needs* spreadsheet receivers have, from the stories we heard about the problems during a transfer. We then analyzed the memos and coded them into the following four categories (with the percentage of memos mentioning one of the information needs in parentheses).

I1 How are the different worksheets—spreadsheet tabs—related? (44%)

I2 Where do formulas refer to? (38%)

I3 What cells are meant for input? (22%)

I4 What cells contain output? (22%)

These results show that the most important information needs of professional spreadsheet users concern the structure of the formula dependencies. Users indicated that the only way they can find these dependencies currently is by using the Excel Audit Toolbar. This Excel feature that shows cell dependencies within one worksheet by overlaying the worksheet with a dependency graph, insufficiently meets these needs. Firstly, this graph becomes incomprehensible quickly when arrows cross each other, as shown in Figure 3.1. Furthermore, it has to be enabled cell by cell, making it impractical for large spreadsheets. Finally, it is not able to show dependencies between worksheets.

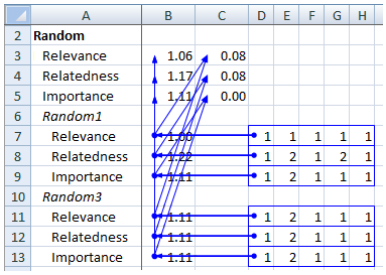


Figure 3.1: *A spreadsheet in Excel, with the Audit Toolbar enabled.*

Underlining the importance of spreadsheets at Robeco were the answers to the final question about the nature of the tasks performed with spreadsheets. We classified these answers into three categories:

Presentation of data (30%) For instance, to show data to colleagues or customers.

Calculations without a direct decision (18%) For instance, calculating the performance numbers of a portfolio to put in a brochure.

Calculations to base decisions on (52%) For instance, calculating the stocks that are generating the lowest profit, to remove them from a portfolio.

These numbers show many spreadsheets within Robeco play a key role in decision making within the company.

3.3 Background

Before presenting how we support professional users' information needs, we provide a brief overview of the preliminaries this chapter builds upon.

3.3.1 Dataflow diagrams

Dataflow diagrams—or similar techniques for representing the flow within systems, such as flowcharts—have been present in literature since the seventies [Gan77]. Dataflow diagrams show how data moves from one process to another and illustrate the relationship of processes and data in an information system. We follow the definition of De Marco [Mar79] who recognizes the following four types of objects:

- *Rectangles* representing entities, which are sources or sinks of data
- *Rounded rectangles* representing processes, which take data as input, perform an operation on the data and yield an output value
- *Open-ended rectangles* representing data stores
- *Arrows* representing the dataflow between objects

Figure 3.2 shows an example of a data flow diagram for a simple order system in which we can recognize a process 'process order' that takes data from entities 'client data' and 'books DB' and outputs its data to entity 'delivery system' and to data store 'order history'.

3.3.2 Leveled dataflow diagrams

A key element in data flow diagrams is the ability to represent a hierarchical decomposition of a process or function, using what De Marco calls a *leveled data flow diagram*. Higher levels are used to abstract the details from lower levels. The lower levels containing the details are revealed by *expanding* an entity at a higher level. By *collapsing* the entity, the details are hidden again.

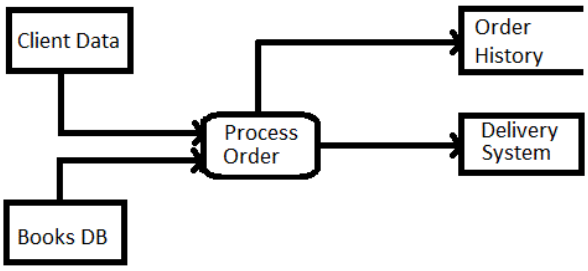


Figure 3.2: *A data flow diagram for a simple order system.*

3.4 Dataflow diagram extraction algorithm

From our analysis of the interviews we concluded that professional spreadsheet users often have difficulties with understanding spreadsheets received in transfer scenarios. We propose to support users in those scenarios by visualizing the structure and contents of a spreadsheet as a leveled dataflow diagram. Dataflow diagrams are commonly used and well understood by spreadsheet end-users. They show the dependencies between cells in the spreadsheet, grouped in levels for worksheets and data blocks.

In the following, we present our approach to derive leveled dataflow diagrams from spreadsheets. The approach consists of six steps which are outlined in Figure 3.3. The first two steps originate from our earlier work extracting class diagrams from spreadsheets [Her10]. The first step determines the cell type of all cells in the spreadsheet, which can be: `Data`, `Formula`, `Label` or `Empty` (Section 3.4.1). The second step identifies data blocks within a spreadsheet as described in Section 3.4.2. In the third step, labels describing `Data` and `Formula` cells are computed (Section 3.4.3). The fourth step generates an initial dataflow diagram by creating entities for cells of type `Data` and `Formula` and creating arrows corresponding to formula dependencies. Details on the dataflow creation can be found in Section 3.4.4. In the fifth step, the labels of cells that were computed in step 3 are attached to the corresponding entities in the diagram (Section 3.4.5). The final step adds the levels to the dataflow diagram. A level is introduced for each worksheet within the spreadsheet and for each data block within every worksheet (Section 3.4.6).

3.4.1 Cell classification

To distinguish between different cell types, we use a cell classification strategy based on the one described by Abraham and Erwig [Erw02]. This algorithm starts with identifying all cells containing a formula and marking them as type `Formula`. Next, the content of the formulas is inspected. Cells that are referred

to in a formula are marked `Data`, unless they were already typed as `Formula` in the first step.

All cells that did not get a type in these two steps are recognized as `Label` when not empty and `Empty` otherwise. Applying this algorithm to the example in Figure 3.3 results in the colored spreadsheet. The orange cells of the columns 'exam' and 'lab' are marked `Data` cells. The green cells of the column 'overall' are marked `Formula` cells. The remaining gray cells are marked `Label` cells.

3.4.2 Identifying data blocks

A data block is defined as a rectangle containing a connected group of cells of type `Data`, `Label`, or `Formula`. Two cells are connected if they touch each other horizontally, vertically or diagonally. To find such a data block, the following strategy is applied: Find the left-most upper-most non-empty cell that is not yet contained within a data block. The initial data block is set to contain only this cell. Next, this data block is expanded by inspecting all cells that are connected to the data block in all directions. If one of these cells is non empty, the data block is enlarged to include this cell. Expanding is done, until all cells next to the data block are empty. In the example in Figure 3.3 the cells are grouped into one data block at $A1 \times E7$.

3.4.3 Name resolution

In this step, names are resolved for cells of type `Data` and `Formula`. To determine the name of such a cell, we consider the data block in which it lies. We assume the name of the cell can be found on the borders of this data block as illustrated by the rectangle in the third spreadsheet of Figure 3.3.

The name of a cell is constructed of two parts: a horizontal part, the first `Label` in the row in which the cell lies, and a vertical part, the first `Label` in the corresponding column.

The algorithm for localizing the vertical part for a cell C is as follows: it starts with inspecting the first cell of the column in which C lies. If this cell is of type `Label`, the value of the cell is set as the vertical part of the name. If the cell is of type `Formula` or `Data`, the procedure is ended without having a vertical part for C. If the cell is of type `Empty`, the algorithm continues with inspecting the next cell in the column. The algorithm is repeated until either a `Formula` or `Data` cell is encountered or the position of the cell, for which the name is computed, is reached. The procedure for computing the horizontal part of the name is similar. Once the two parts are found, the name of a cell is constructed by concatenating the names of the two parts. If the name is empty, the algorithm uses the location of the cell (such as $B5$) as name.

Referring to our example in Figure 3.3, the calculation of the vertical part of the name for cell $C4$ starts at $C1$, which is empty. Next, $C2$ is inspected where the name part 'exam' is found. For the horizontal part the algorithm obtains the

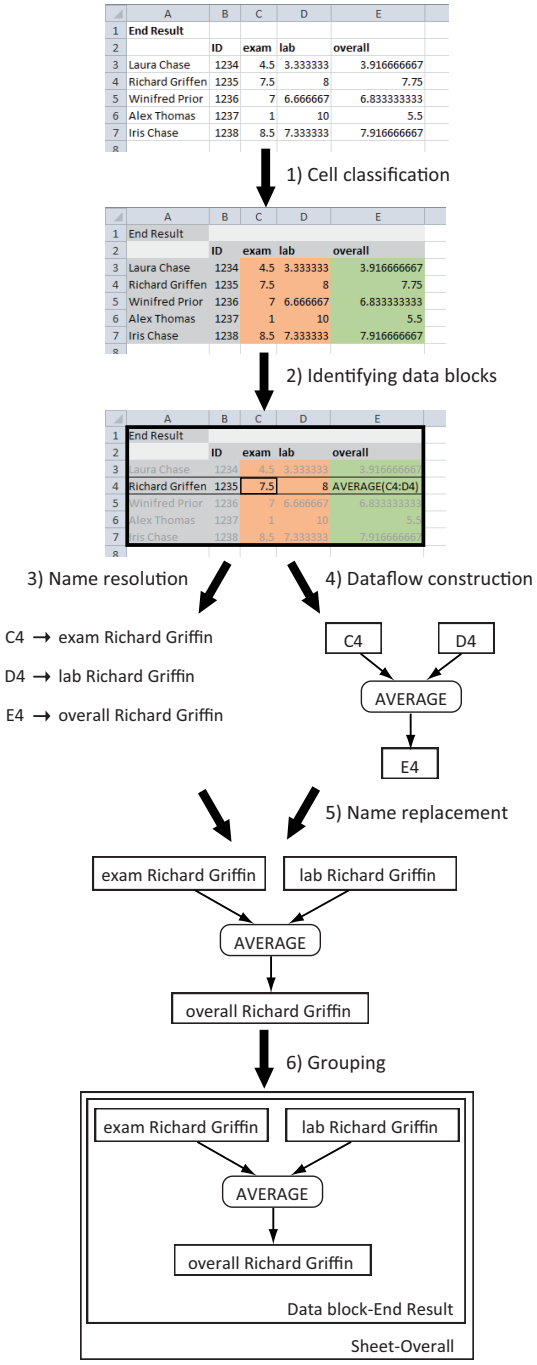


Figure 3.3: *Overview of the approach for extracting leveled dataflow diagrams from spreadsheets.*

name 'Richard Griffin' resulting in the final name 'exam Richard Griffin' for cell $C4$.

The name resolution is heuristic in nature, so it is not always able to find a suitable name. However, as we will see in Section 7, users participating in our case studies found the names useful.

3.4.4 Initial dataflow diagram construction

To create the initial dataflow diagram, we only consider cells of type `Formula` or `Data`. For each cell of type `Formula` two objects are created: an entity denoting the result of a formula and a process representing the formula's calculation. The name for this process node consists of all functions in the formula, like 'SUM' or '*'. One arrow is created from the process to the entity. As can be seen in Figure 3.3 the cell on $E4$ results in an entity 'E4' and a process named 'AVERAGE', showing what is calculated in $E4$. For a cell of type `Data`, for example $C4$, only one entity is created.

Next the arrows are created: If a formula A refers to a cell B, an arrow is introduced from the entity of B to the process of A. As shown in Figure 3.3 the entities for `Data` cells $C4$ and $D4$ are connected to the process of $E4$.

3.4.5 Name replacement

In the fifth step of the transformation, the names for cells found in the name resolution step are attached to the corresponding entities of the initial diagram. This is done so dataflow diagrams show the calculations in terms of labels occurring in the spreadsheet, in order to make the diagrams easier to understand for end-users.

3.4.6 Grouping

The final step groups the entities and processes, according to the levels identified in a spreadsheet, for data blocks and worksheets. First, a level for each data block is created, and entities and processes of the cells in that data block are included in it. We try to find a name for the data block by inspecting its left-most upper-most cell: if this cell is of type `Label` this label is used as the name of this data block. If no label can be found, the location of the left-most upper-most cell is used. Subsequently a level for each worksheet is created that contains the levels of its data blocks. For worksheets the level gets the name the user gave to the worksheet. Referring to our example in Figure 3.3, there is one worksheet 'Sheet-Overall' containing one data block named 'Data block-End Result'.

3.5 Dataflow views

Having described the generation of dataflow diagrams from spreadsheets in the previous section, we turn our attention to supporting users in navigating the diagrams obtained. We decided to express our diagrams in an existing graph format, since several sophisticated graph browsing tools already exist, like Dot, GraphML and DGML. After investigating the available options, we decided to use the DGML (Directed Graph Markup Language) an XML schema for hierarchical directed graphs that can be viewed with the graph browser that is part of Microsoft Visual Studio 2010 Ultimate. It is intended to visualize software architectures and dependency graphs for systems of around 50,000 lines of code.[1]

The DGML graph browser offers a number of features that are directly relevant to the dataflow diagrams we derive from spreadsheets. In particular, zooming, collapsing and expanding levels, the automatic grouping of multiple arrows between entities and processes in one thick arrow and the 'butterfly' mode, a feature that slices a graph showing only the nodes that depend on the selected node.

Users can also add and remove nodes, color nodes and add additional levels with the DGML browser: options that proved to be very useful in the experiments. In the next subsections, we will see how we use these features in the three views we support.

As a running example for all these views we use the spreadsheet to calculate exam scores from Figure 3.3. This spreadsheet consist of four worksheets, and contains 161 cells of which 62 are formulas. It can be obtained online from our website[2] together with the generated dataflow diagrams and a screencast showing our approach in more detail.

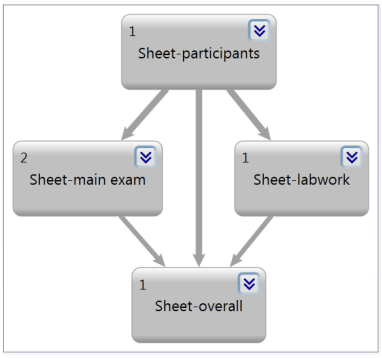


Figure 3.4: *The global view for the exam spreadsheet. Numbers in the upper left corner indicate the number of data blocks within the worksheet.*

[1] http://www.lovettsoftware.com/blogengine.net/post/2010/05/27/Managing-Large-Graphs.aspx

[2] http://swerl.tudelft.nl/bin/view/FelienneHermans/WebHome

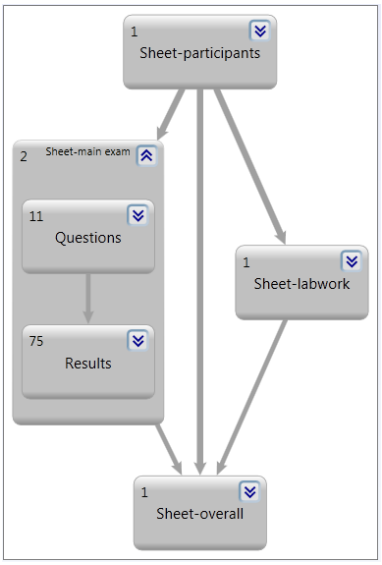


Figure 3.5: *The worksheet 'main exam' expanded to worksheet view. Numbers in the upper left corner indicate the number of cells within the data block.*

3.5.1 Global view

The most top-level view of the dataflow diagram is the *global view*. This view shows all worksheets within the spreadsheet and the relations between them. An arrow from worksheet A to worksheet B indicates a formula in worksheet B refers to a cell in worksheet A. The DGML-browser groups multiple arrows into one, so the thicker the arrow is, the more formulas reference cells of an other worksheet. Figure 3.4 shows the worksheet view of the grades example spreadsheet, where formulas in worksheet 'main exam' refer to cells in worksheet 'participants'.

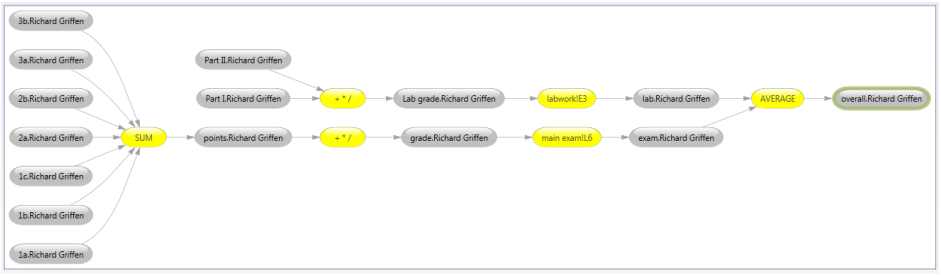


Figure 3.6: *A detailed calculation of an exam grade in the formula view.*

3.5.2 Worksheet view

While examining the spreadsheets we gathered during the initial interviews we noticed that spreadsheet users often structure the contents of a worksheet into data blocks. These data blocks as well as the dependencies between them are visualized by the worksheet view. The view is derived from the global view by expanding a level representing a worksheet. In this way, details of a worksheet are revealed while keeping the overall picture of the spreadsheet. Figure 3.5 shows the worksheet view for worksheet 'main exam' containing the two data blocks: 'Questions' and 'Results'. The data block 'Results' contains formulas that refer to cells in the data block 'Questions'.

3.5.3 Formula view

More details about the calculations within a worksheet can be observed in the formula view. At this level the relation between one specific formula and the cells it depends on is shown. To obtain this view the user opens a data block-node in the worksheet view, showing all calculations in the data block. The user can select the formula for which he wants to enable the formula view, by using DGML's butterfly mode. The original data flow diagram is then sliced to show only the entities and processes connected to the selected node. Figure 3.6 shows the formula view for the calculation of an overall exam score.

3.6 Implementation

The visualization of dataflow diagrams is developed as an extension to the existing Gyro Spreadsheet Analysis Toolkit (GyroSat [Her10]) and is implemented in C# 4.0 using Visual Studio 2010. GyroSAT, which is currently implemented as stand-alone program, parses an Excel file, identifies formulas, blocks, and cell names, and generates dataflow diagrams according to the algorithm described in Section 3.4. The output of GyroSAT is an DGML file which can be navigated by means of Microsoft Visual Studio 2010 Ultimate's built-in DGML browser.

3.7 Evaluation

To evaluate the use of dataflow visualization for spreadsheet comprehension, we performed two studies. First, we asked all respondents to the initial interviews their opinion on the usefulness of dataflow visualization. Secondly, we performed a multiple-case study [Yin02] of nine individual case studies in which we observed users working with GyroSAT. The following two subsections describe both studies in more detail. With these interviews and observations we seek to answer the following research questions.

R1 How does dataflow visualization help users to understand spreadsheets better?

R2 How is our approach able to visualize large and complex spreadsheets?

R3 What are the main limitations of the current approach?

With R1 we seek to understand if and why the approach works well; via R2 how the approach works in extreme cases; and with R3 in what situations the approach does not work well. Both our interviews and the cases aim at addressing all three questions: the interviews aim at getting generalizable answers, whereas the cases seek to increase *realism*, focusing on actual tasks the analysts had to work on in real-life cases.

3.7.1 Interviews

Study design

To test the idea of dataflow visualization we asked all 27 respondents of the initial interviews to express their opinion about dataflow diagrams in general. We interviewed them about this in individual sessions. During the interviews we first showed them an example of a simple dataflow diagram with only a few entities and processes in one level. Next, we showed them a visualization of a more complex spreadsheet, which we obtained during the information needs interviews. We asked them to what extent they were able to understand the given visualization and how they thought these kinds of visualizations could help their work.

Findings

During these interviews we found that almost all interviewees understand dataflow diagrams well. Only one of the respondents indicated that he was not familiar with the dataflow notation and that he had trouble understanding it. All other respondents expressed they feel comfortable using dataflow diagrams. As one of the respondents stated "as analysts we are used to thinking in processes and therefore this kind of diagrams is very natural to us".

With respect to the usefulness of the transformation of spreadsheet visualization to their daily work, around 80% (21 out of 27 respondents) indicated their work could benefit from these diagrams. "They really outperform the Excel Audit Toolbar, because they show the entire view of the sheet", one of the respondents answered, clearly indicating the benefits of dataflow visualization. The most positive respondent stated that this tool "would make my work 10 times easier".

Six respondents indicated problems with the visualization, for different reasons. Two subjects found the generated dataflow diagrams complex and said that they needed more time working with the diagrams to determine whether it could help their work. Two other subjects missed information about *filtering* in the

diagrams. In some cases spreadsheets at Robeco are filled with large amounts of data and Excel's filter function is used to obtain the needed rows. This information is not conveyed with the dataflow diagram, so for these two participants important information was missing.

The fifth subject liked the idea but preferred a visualization within Excel, so it would be easier to see the connection between the two. The final negative respondent indicated his spreadsheets—used to keep track of laws and regulations—were not critical to the company so it would not be worth the trouble of analyzing them.

3.7.2 Case Studies

Study Design

To evaluate the use of dataflow visualization in more depth we performed nine case studies; three for each transfer scenario. In each case study two subjects participated: an *expert*, the user who has the most knowledge of the spreadsheet and a *receiver*, the user who needs to learn about the spreadsheet. During the case study they performed a real-life transfer scenario with their own spreadsheet. There was no time limit to the observation, i.e., we let experts finish their entire explanation. Typically, one study took about one hour. Table 3.1 shows characteristics of the spreadsheets used in the studies; the number of worksheets; the average, minimum and maximum number or rows and columns over all these worksheets; the number of non-empty cells and the number of formulas.

All experts were somewhat familiar with the generated dataflow diagrams, since all of them also participated in the first evaluation study. Therefore, we did not provide them with instructions on how to use GyroSAT and the DGML-browser. We only intervened in the observation if participants got stuck. Participants were allowed to have a second machine with the spreadsheet on it, to point out information about the specifics of the spreadsheet, such as layout or diagrams. They could choose to use both the dataflow diagram and the spreadsheet for supporting their explanation.

Before the observation started we asked experts to briefly explain the contents of the spreadsheet and assess how complicated the spreadsheet was according to them. During the case study we gathered information both by observing subjects and asking them to reflect about the actions they were performing during the course of the study. We gathered all this information in hand-written notes which we used to describe the observations and reactions of the subjects.

With respect to repeatability, we unfortunately cannot provide the spreadsheets used in this experiment since they are confidential. We do give as much information about all other factors of the study to make it as easy as possible for other researchers to perform a similar study.

Scenario	Spreadsheet Description	Worksh.	# Rows			# Col.			# Cells	# Form.
			Avg	Min	Max	Avg	Min	Max		
S1a	Shares risk management	9	393	3	1989	11	1	24	29671	221
S1b	Top and bottom 5 stock performance	5	55	31	74	14	14	16	2781	1601
S1c	Combining data from different sources for weekly reports	16	88	19	294	29	6	73	9555	7215
S2a	Overview of portfolio data	42	272	15	611	12	4	27	28222	13096
S2b	Overview of gain and loss of all trades for one week	10	3269	9	32442	17	7	28	503050	38188
S2c	Constructing a stock portfolio	6	376	38	1000	9	2	28	16054	16659
S3a	Comparison of stock data from two sources	4	48	3	26	14	3	26	2345	336
S3b	Data on loaned stocks and their dividend date	5	100	5	374	26	14	32	1048	469
S3c	Calculating which trades to perform in the future	18	62	1	158	16	5	40	7386	5680

Table 3.1: *Summary of spreadsheets used in the nine case studies.*

Findings

In this section, we describe the most important observations we made during the nine case studies.

S1: Transferring a spreadsheet to a colleague A common observation in all three S1 observations is that experts are surprised by the complexity of their own spreadsheets. Before the actual transfer started they mention the spreadsheet is not very complicated. However when they see the visualization they realize that it is more complicated then they thought. In all cases there were some links in the visualization that were not immediately obvious to the experts. We observed all three experts asking "where does this data actually come from?" more than once.

The global view, with the graphical layout of the worksheets, helps receivers to see what sheet the expert is talking about. As the receiver in scenario S1c (S1c-R) puts it: "This really helps me to understand what [worksheet] is what." Experts use the global view to create a logical story line through the spreadsheet. By starting with, for instance, the worksheet at the end of the flow they can explain the structure of the computations. Expert S1a-E stated that "the global view reminds me of the plan I had when building this spreadsheet. Normally, I just walk through the worksheets from left to right, but that is not the logical way."

There were also differences between the three cases, the most significant being that in scenario S1a the spreadsheet also contained a Visual Basic for Applications (VBA) script and the effects of this script are not considered by our approach yet. The expert in this case clearly thought this was a shortcoming stating "because the VBA part is missing, we only see half of the picture, which is a pity."

S2: Checking a spreadsheet by an auditor All three receivers, in this case auditors, appreciated the help the visualization provided them, especially with acquiring a feeling of how the spreadsheet was built. As S2a-R put it "this picture leads me straight to the difficult parts". Normally, the only way to finds these parts is to select the cell and track the dependencies with the Excel Audit Toolbar. Subject S2c-R agrees with S2a-R that the only choice he had, when performing a spreadsheet audit, was "clicking through all the cells", a time-consuming and tedious process. Subject S2b-R furthermore stated that the global view shows him the "idea behind the spreadsheet" helping him to "find errors on a whole new level." All three subjects felt this tool has the potential to make their work easier.

On the downside, S2a-R and S2b-R indicated that auditors are often interested in relations between the spreadsheet and its *environment*; i.e. external files the spreadsheet retrieves its information from or outputs it results to, such as other spreadsheets or databases. This underlines information needs I3 and I4, although in a manner we did not find during the interviews.

S3: Replacing a spreadsheet by custom software All S3 receivers stated they understood the experts much better with the use of the dataflow diagrams. The global view was appreciated, since all three expressed that in the software replacing the spreadsheet similar dependencies will exist; often what is modeled in a spreadsheet as a worksheet, is represented in software by a class. As S3b-R stated: "This diagram is a basis for the architecture of the software I will build".

In these three cases, we again found that the dataflow diagram helped experts to tell their spreadsheet's story. The top-down way of starting with the global view, advancing through the detailed worksheet view to the formula view reflects the way experts normally explain the spreadsheet, yet is made more tangible by means of the diagrams. We repeatedly saw how experts use the names of cells and regions in the spreadsheet (which are present in the dataflow diagram) to explain the sheet in their own terms.

S3a-R did point out an additional downside of our approach: In industrial spreadsheets, one formula sometimes contains multiple calculations. In our current approach, each formula calculation is represented by only one entity, making it difficult to see what is calculated in the formula. Note that this is particularly relevant to the S3 scenarios, in which the receiver is a software engineer who needs to understand exactly how calculations are performed.

3.7.3 Conclusions

With the results of the 27 interviews and the 9 case studies we are able to answer the research questions.

R1: How does dataflow visualization help users to understand spreadsheets better? One of the main reasons why dataflow visualization supports a spreadsheet transfer scenario is that the global view provides an overview of the spreadsheet helping the expert to create a story around the spreadsheet. In the evaluation, both experts and receivers stated that the global view reveals the *idea* behind the spreadsheet. The detailed worksheet view supports the top-down way in which experts normally explain their spreadsheets. Finally the formula view allows end-users to find details of calculations when necessary. The interactive visualization furthermore allows users, both experts and receivers, to edit the dataflow diagram by adding or removing entities or giving them a specific color, so they can customize the diagram to fit their story exactly.

R2: How is our approach able to visualize large and complex spreadsheets? Several of the spreadsheets under study were large (see Table 1), containing thousands of formulas and dependencies. To be able to handle industrial spreadsheets of these sizes we introduced levels in the dataflow diagrams and provided three different views on the leveled diagram. In all nine case studies the global view was found easy to understand, even for cases with more than 10 worksheets. This is probably caused by the fact that spreadsheet users divide a spreadsheet into worksheets and data blocks themselves. The names for these sheets and blocks are obtained from the spreadsheet, helping users understand

levels in terms of the underlying application domain. The formula view shows the calculation for one formula only, abstracting away from all other details of the spreadsheet, no matter how large it is. This feature was also appreciated by many receivers, especially in scenarios S2 and S3, since there the receivers want to know exactly how formulas are built up.

R3: What are the main limitations of the current approach? There are some spreadsheet concepts the current implementation cannot handle properly. For instance, Visual Basic code is not considered in generating the dataflow visualization. In the observations we noticed that users do see this as a limitation of the current approach, as these concepts are important in the spreadsheets.

Furthermore, our treatment of formulas (representing them by just one process) works well for smaller ones with few operators, but is less helpful for more complex ones. An option might be to represent a formula as multiple processes, for instance representing $(a + b)/(c * d)$ with three processes, one for $+$, $*$ and $/$. We keep this as a point for future work.

For auditors, the information needs I3 and I4 where broadened by the case studies, in which we found that there is also the need to visualize dependencies between a spreadsheet and it's *external data sources*. This can be other spreadsheets, but also databases or programs. The visualization of external spreadsheets with our approach is quite straight forward. It could be created by introducing an additional *file* level in which we group the worksheet by the file they occur in, and visualize relations between different spreadsheet files based on formula dependencies as before. The implementation of relationships between a spreadsheet and other types of data sources is a bit more involved, so we defer that to future work.

In some cases, users would like to see the connection between the dataflow diagram and the original spreadsheet, which is not possible with the current implementation. This need typically occurred when users switched from the global view to the worksheet view. Then they wanted to have a look inside the worksheet to see what region in the spreadsheet corresponded to what data block, especially when GyroSAT was not able to find a name for a data block. Once they understood the connection, they continued their explanation with the support of the dataflow diagram and left the spreadsheet.

3.8 Discussion

The current implementation of GyroSAT, while still a prototype, helps users explain their spreadsheet to colleagues in a structured way. In this section, we discuss a variety of issues that affect the applicability and suitability of the proposed approach.

3.8.1 Meaningful identifiers

It is not in all cases possible to find identifiers in the spreadsheet. In some cases they are simply not present, while in other cases the layout of the spreadsheet is too complicated for our name resolution algorithm. For example, when there are empty rows between the names of the columns and the actual data, the names will be in one data block and the values will be in another, making it impossible for our current naming algorithm to find names for these cells. A point for future work is the improvement of this algorithm. Note that when names are not present they will obviously never been found. This might be overcome by allowing users to attach missing names to cells, for instance by means of a wizard.

3.8.2 Spreadsheet comprehension

Although currently we focus on using our approach for aiding in spreadsheet transfer scenarios, we believe this approach is also suited for what we call *individual spreadsheet comprehension*. This is a scenario in which one person needs to understand a spreadsheet, without someone explaining it. In future work, we will evaluate our approach in this kind of scenarios. In this type of use of our approach the identification of meaningful identifiers is even more important since there is no expert present to add missing information.

3.8.3 Threats to validity

A threat to the external validity of our evaluation concerns the representativeness of the selected set of employees at Robeco and their spreadsheets. However other papers [Hen94; Pan06] report on industrial spreadsheet stories similar to the ones we found at Robeco, so their practice seems to be representable.

Furthermore, there is a risk of aptitude treatment interaction since all of the 27 initial participants where invited for the second evaluation, and it could be the case that only the most positive ones responded to this request. Judging by the number of points of criticism in the second evaluation this seems not to be the case here.

With respect to internal validity, one of the threats is the fact that we did not pick a random sample. This effect can be decreased by using a larger test group in future experiments. We however believe the current test group serves as a good reference group, as the persons varied in their age, function and daily tasks with spreadsheets.

In our empirical study we tried to maximize the realism of our evaluation, which unfortunately comes at the price of reduced repeatability. As an alternative, we considered using the EUSES spreadsheet corpus [Fis05b], which is often used in spreadsheet research (including ours [Her10]). In this case, however, we could not use this corpus, since we were not able to identify users and transfer tasks for the spreadsheets in this corpus.

3.9 Related work

Flowcharts—direct family members of the dataflow dia-grams—have been present in computer science since its early days [Gol47]. Later, Knuth developed a system that could draw them automatically from source code [Knu63]. Experiments done by Scanlan [Sca89] showed flowcharts can be a good asset aiding users in comprehending source code.

The problems around spreadsheet use are a topic for many papers. Nardi and Miller [Nar90] for instance found that "it is difficult to get a global sense of the structure of a spreadsheet, which requires tracing the dependencies among the cells." This statement is very similar to the ones we gathered in the interviews. Users in the survey of Nardi and Miller named this tedious process their biggest complaint about spreadsheets. Burnett and Rothermel have written a series of papers on testing spreadsheets [Bur99; Rot98]. Underlying their test methods is the Cell Relationship Graph (CRG), a graph that shows the relation between cells. They however use this graph to calculate what cells depend on what other cells and the CRGs are not shown to end-users testing the spreadsheet.

Many papers describe mechanisms to visualize spreadsheet structure to support user understanding. The first one was Davis [Dav96] suggesting the 'online data dependency tool' that extracts a *spreadsheet flow diagram*, as proposed by Ronen *et al.* [Ron89] to document spreadsheets. The tool was never implemented because at the time of writing graphical layout of these diagrams was too complicated to automate. They did perform an evaluation with a hand-generated diagram and compared its usefulness to an arrow diagram, a diagram showing dependencies directly in the spreadsheet, similar to Excel's Audit Toolbar. Results show that participants identified more cell dependencies with the diagrams than without, although the arrow diagram performed slightly better then the online diagram.

Clermont [Cle04] introduces the notion of data-dependency graph (DDG), showing end-users the relation between all cells within a spreadsheet as part of an auditing toolkit. A DDG however only contains cell names (such as $B5$) and does not show the formula calculations.

Shiozawa *et al.* [Shi99] proposes to create a dataflow diagram in 3D on top of a given spreadsheet, where more distinct connections are placed higher above the spreadsheet. This is an interactive process in which users can select cells and 'lift them up'. Although their approach seems very promising it has to date never been evaluated in practice.

The main difference between these related approaches and ours is the introduction of levels in the dataflow diagram, the automatic extraction of names for worksheets, data blocks and cells and our extensive evaluation in industry.

3.10 Concluding remarks

The goal of this research is to underline the importance of spreadsheet analysis and visualization as a means to aid users in understanding spreadsheets. To that end we have designed an approach to represent calculations within a spreadsheet as data flow diagrams and implemented this approach in the Gyro Spreadsheet Analysis Toolkit. The key contributions of this work are as follows:

- A detailed survey analyzing the information needs of spreadsheet users (Section 3.2)
- An algorithm to extract a dataflow diagram from a spreadsheet (Section 3.4)
- Different views on the dataflow diagram (Section 3.5)
- An implementation of the proposed methods in the GyroSAT toolkit (Section 3.6)
- An evaluation of the proposed approach within a large Dutch financial company (Section 3.7)

The current research gives rise to several directions for future work. As mentioned before we should concentrate on gathering more information from the spreadsheets, not only from formula dependencies, but from Visual Basic code as well. It would also be an improvement to link the dataflow diagram to the spreadsheet. Ideal would be to have a dataflow diagram and a spreadsheet on the screen simultaneously and highlight a certain part of the data flow diagram when a corresponding part of the spreadsheet was selected, and vice versa.

Chapter 4

Detecting and Visualizing Inter-worksheet Smells in Spreadsheets

4.1 Introduction

Spreadsheets are widely used in industry: Winston [Win01] estimates that 90% of all analysts in industry perform calculations in spreadsheets. Their use is diverse, ranging from inventory administration to educational applications and from scientific modeling to financial systems. Especially in the financial domain spreadsheets are prevailing. Panko [Pan06] states that 95% of U.S. firms, and 80% in Europe, use spreadsheets in some form for financial reporting.

Business analysts using spreadsheets usually have very limited training in programming or structuring data. In spite of that, they effectively are *end-user programmers*, and as such face many of the challenges of professional developers, such as identifying faults, debugging, or understanding someone else's code [Ko10].

This chapter aims at providing support for spreadsheet users to take on these end-user programming challenges, focused on support for identifying potentially risky parts in a spreadsheet's high level *design*, i.e. the way in which worksheets are organized and depend on each other.

Our starting point is the code smell metaphor introduced by Fowler [Fow99]. In particular we study the coupling and cohesion of classes (called Collaboration Disharmonies by Lanza and Marinescu [Lan05]) and transform these code smells in such a way that they apply to the coupling of worksheets rather than classes. This leads to a list of *inter-worksheet smells*.

In order to detect these smells automatically, we define metrics for each of

them. With each metric, we establish a threshold, to know at what point to identify a worksheet as smelly. We follow the approach of [Alv10] who analyzed metrics on source code and set thresholds by determining the worst 70, 80 and 90% of all methods and choosing corresponding metric values for the thresholds representing medium, high and very high risk.

We then address the issue of communicating identified smells to spreadsheet users. For this we use our original worksheet visualization, since earlier experiments have shown that these diagrams are useful for spreadsheet users to get an overview of the structure of their spreadsheet. We enrich those data flow diagrams with colors and tool tips to convey the inter-worksheet smells to spreadsheet users.

We perform a quantitative and qualitative evaluation of our approach. Firstly, we analyzed the EUSES corpus, and investigated the occurrence of inter-worksheet smells. Secondly, we conducted a series of case studies at a large Dutch financial institution, called Robeco. Here we conducted ten case studies. In each study we analyzed a real-life spreadsheet, and discussed the located smells with the spreadsheet owner, supported by the generated data flow diagram. The evaluations aim to answer the following research questions:

R_1 What inter-worksheet smells are the most common, and why?

R_2 How do inter-worksheet smells expose threats to spreadsheet quality and calculation integrity?

R_3 To what extent are enriched data flow diagrams an appropriate way of visualizing inter-worksheet smells?

The findings of these evaluations indicate that (1) inter-worksheet smells are commonly found in real-life spreadsheets, (2) inter-worksheet smells can indicate real weaknesses in a spreadsheet's design and (3) data flow diagrams seem useful to help spreadsheet users locate and understand inter-worksheet smells.

This chapter is organized as follows. Section 4.2 gives an overview of related work in the area of code smells and spreadsheet metrics. In Section 4.3 we sketch the background of our approach and illustrate it with a motivating example. Section 4.4 introduces the inter-worksheet smells, followed by Section 4.5 that explains how to automatically detect the smells. In Section 4.6 the communication of the smells to spreadsheet users by means of data flow diagrams is described. Section 4.7 describes the implementation of our prototype Breviz, while Section 4.8 explains the two evaluations we have performed in detail. Section 4.9 discusses the applicability of the proposed approach, followed by the concluding remarks in Section 4.10.

4.2 Related Work

Efforts related to our research include work on code smells, starting with the canonical work by Fowler [Fow99]. His book gives an overview of code smells and

corresponding refactorings. Recent efforts focused on the automatic identification of code smells by means of metrics. Marinescu [Mar01] for instance, uses metrics to identify *suspect* classes, those classes that might have design flaws. Lanza and Marinescu [Lan05] explain this methodology in more detail. Alves *et al.* [Alv10] focus on a strategy to obtain thresholds for metrics from a benchmark. Olbrich *et al.* furthermore investigates the changes in smells over time, and discusses their impact [Olb09].

Furthermore, there are papers that address common errors in spreadsheets. Ayalew *et al.* [Aya00] for instance names the incorrect grouping of data, and incorrect coupling between those groups as sources of errors. Panko [Pan98] names omission errors as the most dangerous of spreadsheet errors. Omission errors are those errors that arise from "misinterpretation of the situation that should be modeled" (Powell *et al.* [Pow09]). In our own previous work [Her11] we have developed an approach to visualize the flow of data within spreadsheets by means of a data flow diagram. The evaluation showed that those diagrams are useful for spreadsheet professionals that need to understand the structure of a spreadsheet. In other recent work [Her12c] we have applied code smells to individual formulas, rather than worksheets.

Finally, there are papers on spreadsheet metrics, which are also aimed at locating weak points in spreadsheets. In 2004, Bregar published a paper presenting a list of spreadsheet metrics based on software metrics [Bre04]. He however does not provide any justification of the metrics, nor did he present an evaluation. Hodnigg and Mittermeir [Hod08] propose several spreadsheet metrics of which some are similar to Bregar's. Their metrics are divided into three categories: general metrics, such as the number of formulas and the number of distinct formulas; formula complexity metrics, such as the number of references per formula, and the length of the longest calculation chain; and finally metrics, such as the presence of scripts in, e.g., Visual Basic for Applications (VBA), user defined functions and external sources. Besides the metrics, the authors also pose the interesting suggestion to use different types of visualizations for cells with different values for the metrics. Hole *et al.*[Hol09] propose an interesting approach to analyze spreadsheets in terms of basic spreadsheet metrics, such as the number of functions used, the presence of charts and the complexity of program code constructs to predict the level of the spreadsheet creator.

We have combined the work on the definition and detection of code smells with existing spreadsheet work, to obtain a list of inter-worksheet smells that can be automatically detected.

4.3 Background & motivating example

In previous work [Her11] we created an approach for generating leveled data flow diagrams from spreadsheets, to support professionals in explaining their spreadsheets to colleagues. Figure 4.1 depicts an example of such a data flow diagram,

showing all worksheets within a spreadsheet and the formula relations between them. Rounded squares represent worksheets, and an arrow between worksheet A and worksheet B means that formulas in worksheet B refer to cells in worksheet A. The thickness of the arrow indicates how many unique formulas connect A and B. From this figure we can learn that the spreadsheet in question contains five worksheets, and that, for example formulas in 'exam' refer to cells in worksheet 'result79813'.

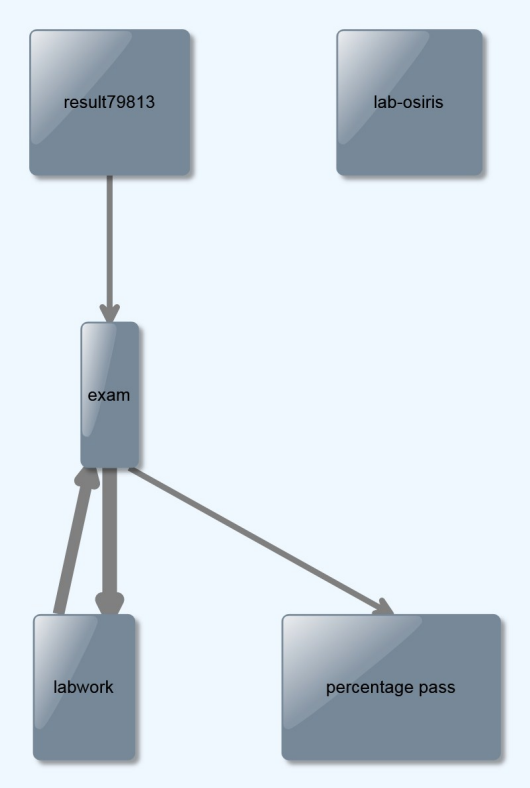


Figure 4.1: *A leveled data flow diagram, representing data flow in a spreadsheet*

We have implemented the generation of leveled data flow diagrams, and tested our approach at Robeco, a Dutch investment bank [Her11]. While we observed users working with the data flow diagrams, we noticed that they often used the diagrams as a means of assessing the quality of the spreadsheet it represented. A spreadsheet leading to a 'spaghetti' diagram was considered of lower quality than a diagram that looked very structured.

This observation led to the idea of using our diagrams to assess spreadsheet quality, which we will elaborate on in the remainder of this chapter. We will in-

vestigate the hypothesis that the global view of our leveled data flow diagrams can also serve as a means of identifying weak points, or even flaws in a spreadsheet's design.

The following example demonstrates this. Figure 4.1 shows the data flow visualization of a real-life spreadsheet that is used by a university professor to calculate the grade for a course. It consists of five worksheets. From the diagram some aspects of the spreadsheet immediately catch the eye. For instance, one of the worksheets *lab-osiris* is not connected to the other sheets. This sheet contains the data from Osiris, the university's grading system. This information can be important when working with the spreadsheet, since the spreadsheet user might mistakenly think that the exam scores are automatically updated when the information in Osiris is updated. To determine this without Breviz would require the user to select all cells for all worksheets one by one and checking their dependents. Furthermore, the loop between *exam* and *labwork* stands out. The name *exam* could suggest that the worksheet only contains data about the exam, however it apparently also contains information regarding the lab work. This example illustrates the type of information that can be gathered from a data flow diagram generated from a spreadsheet.

We found, in observing users at Robeco working with Breviz that the questions raised by looking at the diagram, help spreadsheet users assess the quality of their spreadsheet.

4.4 Inter-worksheet smells

In this section we look at Fowler's code smells [Fow99] and investigate which of the code smells can be adapted in such a way that it applies to spreadsheets. We focus on inter-class code smells, since they can be related to the inter-worksheet smells. We also leave out the code smells that involve inheritance—Parallel Inheritance Hierarchies and Refused Bequest—since that concept does not directly occur in spreadsheets.

4.4.1 Inappropriate Intimacy

This smell indicates that a class has too many dependencies on implementation details of another class. A related spreadsheet smell would be a worksheet that is overly related to a second worksheet. This is possibly unhealthy for several reasons. First, adapting one sheet likely requires inspecting the other worksheet, requiring the spreadsheet user to switch back and forth between the two worksheets, increasing the chance that errors are made [Nar90]. Secondly, since there is a strong semantic connection between the two worksheets, the fact that they are split could influence understandability.

4.4.2 Feature Envy

Feature Envy is the phenomenon where a method M seems more interested in the fields of another class than of the class that contains M. In general, Fowler suggests to put a method in the class that contains most of the data the method needs. This code smell seems very applicable to spreadsheets: if there is a formula that is more interested in cells from another worksheet, it would be better to move the formula to that worksheet. This will likely improve understandability, since the formula is then closer to the cells it is referring to. Conway and Ragsdale [Con97] stated in their spreadsheets guidelines that "things which are logically related should be arranged in close physical proximity and in the same columnar or row orientation". If the result of the formula is needed in other worksheets, it could still be linked to the necessary sheets after it is moved.

4.4.3 Middle Man

Fowler defines a middle man as a class that delegates most of its operations to other classes, and does not contain enough logic to justify being a separate class. When this occurs, it might be time to refactor out the middle man.

This smell could also occur in spreadsheets, where 'middle man' formulas occur: formulas that only contain a reference to another cell, like the formula '=Sheet1!A2.' If a worksheet contains many of those middle man formulas, it might be better to remove this worksheet, and move its functionality to other worksheets.

A worksheet suffering from the Middle Man smell could complicate the structure of a spreadsheet, and therefore reduces spreadsheet quality. Many papers on spreadsheets stress the importance of structure in a spreadsheet's design. Janvrin and Morrison [Jan00] for instance, state that using a structured design approach reduces risks in end-user spreadsheet development. Markus and Keil agree and further note that structured design methodologies increase the accuracy and reliability of information systems [Lyn94]. Cheney *et al.* [Che86] found that the more structured an end-user works, the more likely the success of the application.

4.4.4 Shotgun Surgery

Source code smells of 'shotgun surgery' when one change results in the need to make a lot of little changes in several classes. One of its common forms is a method A that is referred to by several other methods in several different classes. When A is changed, it is likely that also the callers of A will have to be changed, resulting in a lot of changes at different places.

The spreadsheet translation of this code smell is a formula F that is referred to by many different formulas in different worksheets. By the same logic, the chances are high that many of the formulas that refer to F will have to be changed if F is changed.

Shotgun Surgery could have an impact on the maintainability of a spreadsheet, since it requires the user to make a number of changes when F is changed.

4.5 Detecting inter-worksheet smells

In this section, we present our approach to automatically detect worksheet smells in spreadsheets. We base our approach on existing work done in the domain of object-oriented software engineering, such as [Alv10; Lan05; Olb09]. We mainly follow the approach by Marinescu described in [Mar01]: for each of the smells we define one or more metrics that indicate the presence of that particular smell. We subsequently analyze a large body of spreadsheets and set thresholds by determining the worst 70, 80 and 90% of all worksheets and choosing corresponding metric values for the thresholds representing medium, high and very high risk.

Definitions To be able to reason about spreadsheets and smells, we define the following types, sets and functions.
Cell The type C represents a cell in a spreadsheet.
Worksheet W represents a worksheet in a spreadsheet, and is defined as a set that contains all cells that are located in the worksheet.
Spreadsheet S represents a spreadsheet, and is defined as a set that contains all worksheets contained in the spreadsheet.
Precedents Precedents P is a function of type C $\rightarrow$ {C}
representing all precedents of a given cell. Precedents are the cells that a formula refers to.
Connection A connection K is a tuple (A,B) of two cells. Two cells are called connected if A$\in$P(B).
Connection Set The set KS is the set containing all connections of a spreadsheet.

4.5.1 Inappropriate Intimacy

To detect Inappropriate Intimacy we investigate the amount of data coupling between two different worksheets. We count the number of connections between two worksheets, which we call the *Intimacy* of these worksheets that is of type W $\times$ W $\rightarrow$ int and is defined as follows

$$\text{Intimacy}(w_0,w_1) \equiv |\{(c_0,c_1 \in KS : c_0 \in w_0 \wedge c_1 \in w_1 \wedge w_0 \neq w_1\}|$$

We count the number of pairs in KS where the cell c_0 is contained by worksheet w_0 and the cell c_1 is contained by worksheet w_1 and the two worksheets are not the same. Note that this definition implies that if there are two formulas on worksheet y referring to the same cell in x, we count this as two connections rather than one.

To get the intimacy for one worksheet, we take the maximum intimacy the worksheet has with any of its connected worksheets

$$\mathrm{II}(w_0) \equiv \max\{\mathrm{Intimacy}(w_0, w_1) : w_0,w_1 \in S\}$$

4.5.2 Feature Envy

The Feature Envy smell is actually a smell that applies to a formula rather than to a worksheet. We adhere to the approach of Olbrich *et al.* [Olb09] who state that when a method has a certain smell, by extension the class that contains it is also smelly.

To detect the Feature Envy smell on worksheets, we inspect all its formulas and check the cells that they are 'interested in'; the cells that they refer to. As a metric for enviousness, we count all references a formula has to cells contained by the other worksheets of a spreadsheet. Hence the definition of *Enviousness(FE)*, a function of type C $\rightarrow$ int is

$$\mathrm{FE}(c_0) \equiv |\{(c_0,c_1) \in KS : \exists w:\ c_0 \in w \wedge c_1 \notin w\}|$$

We count the number of pairs from the connection set where cell c_0 is contained by w but not the cell c_1.

4.5.3 Middle Man

The Middle Man smell is detected when a worksheet is mainly used to pass values to another worksheet. To detect this smell we use the definition of a special type of formulas, the *middle man* formula. A middle man formula does not contain any operations besides the = operation that gets a value from another cell. The function MMF: C $\rightarrow$ bool indicates whether a formula is a middle man formula.

When there is a calculation chain that contains two consecutive passing formulas, there is risk of the Middle Man smell. We therefore count the number of middle man formulas in a worksheet that are referred to by another middle man. This makes *Middleman (MM)* a function of type W $\rightarrow$ int, defined as

$$\mathrm{MM}(w) \equiv |\{(c_0,c_1) \in KS:\ c_1 \in w \wedge \mathrm{MMF}(c_0) \wedge \mathrm{MMF}(c_1)\}|$$

4.5.4 Shotgun Surgery

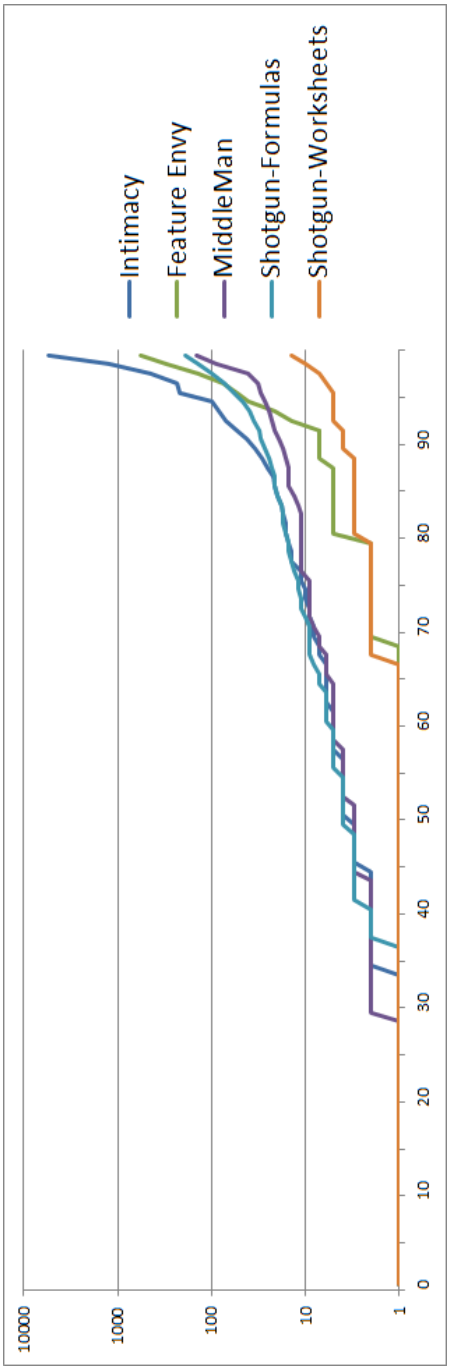


Figure 4.2: *The distribution of the various metrics on a quartile graph with logarithmic scale on the y-axis*

We count the number of connected formulas that are both middle man formulas. To detect shotgun surgery in worksheets, we take our inspiration from the method described by Lanza and Marinescu [Lan05]. This method entails counting *changing methods*; the number of distinct methods that call a method A of the class and *changing classes*; the number of classes in which the methods that call the measured method are defined. These methods and classes are called *changing*, because they are the methods and classes likely to change if method A changes. We adapt this method to make it applicable to spreadsheets, and introduce the following definitions, both of type $W \rightarrow int$

Changing Formulas The number of formulas that refer to a formula in worksheet w, defined as

$$\text{ChangingFormulas(w)} \equiv |\{(c_0,c_1)\in KS : c_0 \notin w \wedge c_1 \in w\}|$$

Changing Worksheets The number of worksheets in which the changing formulas lie, defines as

$$\text{ChangingWorksheets}(w_0) \equiv |\{w_1 \in S: (\exists (c_0,c_1)\in KS : c_0\ w_1 \wedge c_1 \in w_0)\}|$$

4.5.5 Determining the thresholds

Now we have determined the metrics for each of the smells, we establish the thresholds for each of the metrics. We establish the thresholds by analyzing the distribution of the metric values over a large, representative body of spreadsheets. We follow the approach of Alves *et al.* [Alv10]. They inspect the percentage of metric values below a certain given percentage, and set the thresholds accordingly.

The body of spreadsheets we use is the EUSES Spreadsheet Corpus [Fis05b]. This corpus contains real-life spreadsheets from all sorts of domains, ranging from educational to financial, and from inventory to biology. It was created in 2005 and has since then been used by several researchers to evaluate spreadsheet algorithms, among which [Abr06] and [Cun09b].

The corpus comprises of 4,223 spreadsheets, which together contain 15,015 worksheets. Of those spreadsheets, there are 1,711 spreadsheets that contain formulas, divided over 4,250 worksheets. Figure 4.2 shows the distribution of all metrics over the worksheets in a quartile graph.

As can been seen in this figure, the metrics all follow a power law like distribution, having most of their variability on the tail. Feature Envy and Shotgun Surgery-Worksheets only have values higher than one above 65% of the worksheets, and the other three metrics pass the value of 10 at around 70%.

Because the metrics follow a distribution similar to the metrics in [Alv10], we use the same method for setting the thresholds for our metrics. We therefore choose 70%, 80% and 90% as the thresholds for medium, high and very high

risk that a worksheet suffers from an inter-worksheet smells as introduced above. Table 4.1 shows the thresholds of the five metrics for the four smells.

Table 4.1: *Thresholds for the metrics determining the smells*

Smell	70%	80%	90%
Inappropriate Intimacy	8	16	42
Feature Envy	3	5	7
Middle Man	7	11	19
Shotgun - Changing Formulas	9	16	30
Shotgun - Changing Worksheets	2	3	4

4.6 Visualizing inter-worksheet smells

Once we have established the preliminary code smells, there is the question how to communicate the smells to spreadsheet users. Since we have found in earlier work that data flow diagrams are very suitable to explain the structure of a spreadsheet, we will use them to convey the inter-worksheet smells as well. For this research we adapted the diagrams by enriching them with information about the inter-worksheet smells. When a smell is located, the worksheet box in the dataflow diagram is colored yellow, orange or red, for smells at the 70%, 80% or 90%. A tool tip explains the spreadsheet user what smell is located, and what cells contribute to this smell.

Colors and tool tips are simple and well-known user interface aspects, hence this seems an effective way of indicating code smells. Figure 4.3 depicts the user interface of our tool Breviz, with the spreadsheet smell indicators, for a spreadsheet from the EUSES corpus. As this figure shows, smells are located in two worksheets, 'Capital Exp' and 'Cost-Quarterly'. The tool tips indicate what smells are located, Feature Envy in 'Capital Exp' and Inappropriate Intimacy in 'Cost-Quarterly'. The tool tips furthermore name the cells that contribute to these smells.

4.7 Implementation

The enriched data flow visualizations were incorporated into our existing spreadsheet analysis system Breviz [Her11]. Breviz is implemented in C# 4.0 using Visual Studio 2010. It utilizes the Gembox component to read Excel files[1] and visualizes the data flow diagrams using YFiles for .NET/WPF[2]. Breviz, which

[1] `http://www.gemboxsoftware.com/spreadsheet/overview`
[2] `http://www.yworks.com/en/products_yfilesdotnet_about.html`

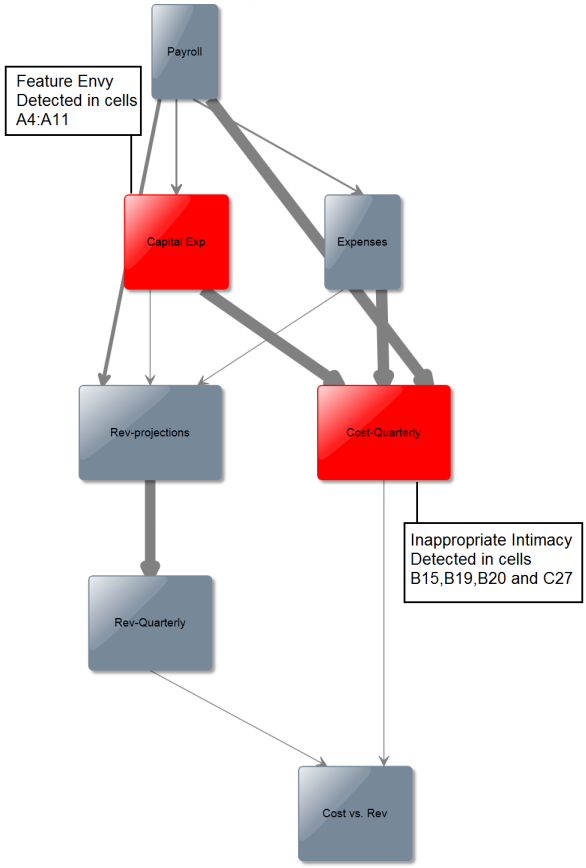


Figure 4.3: *A screenshot of a spreadsheet data flow visualization enriched with smell information*

is currently implemented as stand-alone program, reads an Excel file, writes the metrics described in Section 4.4 to a SQL Server 2008 database and subsequently generates an annotated data flow diagram.

4.8 Evaluation

In this section we will explain how our inter-worksheet smells and their visualization were evaluated. With the evaluation aim to answer the research questions. To do so, we performed two separate evaluations, both a quantitative and a qualitative evaluation. In the quantitative evaluation we analyzed the occurrence of inter-worksheet smells in the EUSES Spreadsheet corpus, given the thresholds we have selected. With this evaluation, we focused on research question R_1.

For the qualitative evaluation, aimed at more understanding of R_1, plus answers to R_2 and R_3, we asked 10 professional spreadsheet developers for access to their real-life spreadsheets. We let our tool Breviz identify possible code smells and showed the user the enriched data flow visualization. We subsequently asked the spreadsheet users to reflect on the identified inter-worksheet smells.

The following subsections describe the two evaluations in more detail.

4.8.1 Inter-worksheet smells in the EUSES Corpus

Goal

During the first evaluation we want to learn more about the occurrence of the four inter-worksheet smells, and hence focus on the question what smells are most common(R_1).

Setup

In this evaluation we used the EUSES Spreadsheet Corpus. As stated above, this corpus consists of 4,223 real-life spreadsheets, from all sorts of domains, ranging from educational to financial, and from inventory to biology.

For each of the four inter-worksheet smells, we checked how many spreadsheets contain worksheets with metric values above the 70%, 80% and 90% thresholds. This gives an overview of the distribution of the smells over the spreadsheets.

Results

Table 4.2 shows the results of the first evaluation. As can be seen in this table, Feature Envy is the most common smell, with 12.4% of the spreadsheets containing worksheets with metric values above the 70% threshold.

Feature Envy is followed by Inappropriate Intimacy, with 9.6% above the 70% threshold. Feature Envy and Inappropriate Intimacy are related, since Feature Envy can be the cause of Inappropriate Intimacy.

The observation that those two smells are common is consistent with previous work, in which we have seen that it is difficult for spreadsheet creators—that are usually not trained as programmers—to structure their spreadsheets in a logical way [Her11].

We believe these smells pose a real threat to spreadsheet understandability, since in previous user studies it has been shown that it is difficult for spreadsheet users to work with the connection between multiple worksheets. Nardi and Miller [Nar90] for instance found that "it is difficult to get a global sense of the structure of a spreadsheet. That requires tracing the dependencies among the cells." We will investigate these smells in more detail in the second evaluation.

Third most common is the Middle Man smell, with which 5.9% of the spreadsheets are diagnosed. Finally, with 4.1% of the spreadsheets suffering from Shotgun Surgery at the 70% level, this is the least common smell in the EUSES corpus.

Table 4.2: *Percentage of spreadsheets in the EUSES corpus that contains at least one worksheet that suffer from at least one smell above the 70, 80 and 90% thresholds.*

Smell	**>70%**	**>80%**	**>90%**
Feature Envy	12.4 %	8.7%	5.8%
Inappropriate Intimacy	9.6%	6.8%	4.2%
Middle Man	5.9 %	5.0%	4.0%
Shotgun Surgery	4.1 %	3.3%	1.5%
Any of the above smells	23.3 %	17.6%	12.8%

Table 4.3: *Characteristics and number of smells above the 70% thresholds of spreadsheets used in the ten case studies.*

ID	Spreadsheet Description	#Wrks.	#Cells	#Form.	#Uniq.	Size(Kb)	II	FE	MM	SS
1	Calculate dividend	5	13,697	6,012	53	183	2	-	-	-
2	Overview of investment strategies	5	21,600	3,031	98	605	3	2	1	3
3	Model companies in the energy sector	14	82,000	14,742	531	826	2	7	2	6
4	Valuation of swaps	8	31,485	5,034	67	1,690	1	3	-	-
5	Overview profit and loss of all traders	10	17,263	9,152	142	4,261	-	-	-	-
6	Overview of risk for different sectors	9	9,190	148	12	332	-	-	-	-
7	Comparing different calculation models	14	24,580	3,388	39	348	5	3	5	-
8	Planning of trades	1	2,329	1,683	64	76	-	-	-	-
9	Report interest rate and liquidity risk data	25	59,116	17,930	117	1,693	8	6	-	4
10	General ledger data for analysis	11	11,906	3,049	56	1,732	3	3	-	-

4.8.2 Inter-worksheet smells in ten real-life case studies

Goal

The objective of the second evaluation is to answer the "why" of research question R_1, as well as to provide the answers to R2 and R3.

Setup

For the second evaluation we gathered 10 professional spreadsheets from the financial domain. We performed our evaluation research at Robeco, a Dutch asset management company with approximately 1600 employees worldwide, and over 130 billion Euro worth of assets under management. In a survey we conducted among 27 of their analysts, we found that they use Excel for an average of 3 hours a day, underlining the importance of spreadsheets in their daily work. Furthermore, we found that spreadsheets have an average lifetime of more than five years, and individual spreadsheets are used by 13 different analysts on average [Her11].

We invited participants of this previous survey to participate in this evaluation. We asked them to select one of their large and complex spreadsheet, which they worked with often. Ten subjects responded positively and participated in this evaluation. The ten subjects were familiar with our data flow diagram visualization, since all of them participated in our previous study [Her11].

Before the case studies started we provided subjects with information about the setup. We explained participants that we were going to identify inter-worksheet smells that could possibly indicate weak points in their spreadsheets. We furthermore told them we wanted to discuss the smells with them in a short interview. Finally, we provided subjects with a list of the four inter-worksheet smells and a short explanation of the smells, so they could study this before the experiment.

For each of the ten subjects and their spreadsheet, the procedure was as follows:

First, we asked the subjects to explain the purpose and context of the spreadsheet. We then generated the annotated data flow diagram. Subjects studied the data flow diagram and the corresponding spreadsheet, for a maximum of 10 minutes, after which the interview part of the study started. In this part of the study, we asked the subject for each of the located inter-worksheet smells:

- Do you understand why this part of your spreadsheet is marked as potentially risky?
- Do you agree that there is indeed a problem or an error here?
- Can you explain why you structured the spreadsheet like this?
- Does the data flow visualization help you find the smells?

With these questions we analyzed whether the smells and their impact are recognized by spreadsheet users, and investigate their causes. We furthermore learned about the reception of the data flow diagrams.

Table 4.3 shows an overview of the characteristics of the spreadsheets used in the case study. As can be seen in this table, the spreadsheets are of considerable size, with as many as 25 worksheets. Seven of the ten spreadsheets suffered from at least one of the inter-worksheet smells, and Inappropriate Intimacy is the most common smell among the spreadsheets.

Results

General observations In all the ten case studies, we noticed that the subjects were surprised when confronted with the data flow diagrams. They often expressed statements such as "are those worksheets really that connected?" or even "are you sure that arrow is correct?". These experiences corroborate what we found in a previous study [Her11]. Since the arrows in the data flow diagram are thicker when more formulas are connecting two worksheets, subjects could easily see that two worksheets were strongly connected. We noticed that subjects found the data flow visualization to be helpful in understanding the smells. One of the subjects stated "that arrow is so fat. That can't be good".

The popups then helped to explain what smell exactly was found. We found the addition of the smelly cells in the popup to be of great value as well. This way users could locate the smells within the worksheet, and determine how the spreadsheet could be improved.

Inappropriate Intimacy When interviewing the subjects about the Inappropriate Intimacy smell, we were again struck by how difficult it is for spreadsheet users to understand dependencies between the worksheets. In all seven cases where Inappropriate Intimacy was found, it took users time to understand what cells were connecting the worksheets and why, stating e.g. "what did I do here again" and "I don't remember why I needed to connect those sheets". Even when supported by the data flow diagrams, they were searching for the reason that a strong dependency was detected, and why they constructed the spreadsheet in that way. This was caused mainly because the spreadsheets did not have the right documentation to support users in these type of questions. While some spreadsheet contained documentation that explained how to work with the spreadsheet, none of the spreadsheets contained information on the *design decisions* of the spreadsheet.

Asking the subjects whether they understood why the Inappropriate Intimacy smell was found, all responded positively. However not all of them agreed that the smell was indeed harmful. We recognized two patterns causing for Inappropriate Intimacy. One is the use of an 'auxiliary' worksheet, in which data is stored, in combination with another worksheet in which this data is referred to. This construction is similar to a join on two tables. This is often implemented using a VLOOKUP, MATCH or IF function.

A second case, deemed more dangerous by subjects, involves two worksheets referring to each other without a clear distinction between the two worksheets. In this case, we witnessed subjects making statements along the lines of "it would in fact be better to merge these two worksheets". In future work, we plan to investigate these two types of intimacy in more detail.

The data flow diagrams were helpful when identifying the smells. They helped the subjects to see what worksheets where connected. One of the subjects stated: "a red block draws my attention, and the thick arrow indicates where there is too much connection".

However, especially in the second case, where two worksheets were overly intertwined, users looked in the spreadsheet, to investigate the formulas in more detail.

Feature Envy A formula suffers from Feature Envy at the 70% level when at least 3 of its references are located in another worksheet than the formula itself is in.

When a formula refers to a number of cells that lie in a different worksheets, understanding the formula becomes harder. This is due to the fact that modern spreadsheet programs such as Excel highlight the cells that a formula refers to when a formula is selected, but only those cells that are located on the same worksheet as the selected formula. The highlighting feature is used extensively by spreadsheet users, but it does not provide any help when analyzing references across different worksheets. The above contributes to the fact that in all cases in which we discovered Feature Envy, subjects agreed that a refactoring would help, since they recognized the difficulty of analyzing a formula with many references lying in a different worksheet. One subject stated about a formula in his spreadsheet, which referred to no less than 8 cells in a different worksheet: "this formula annoys me, I have to go back to the other sheet so many times to look up the references, it makes me dizzy".

Performing the calculation on the worksheet where the referents are, and than referring to the result from the worksheet where the result is needed, seemed more healthy to all subjects.

In the case of Feature Envy, the data flow diagrams also supported the identification of the smell. However, in this case subjects needed to dig deeper into the spreadsheet formulas to understand why they were marked as envious. While the data flow diagrams list the cells that are smelly, subjects often felt the need to inspect the worksheet, and select the listed cells, to inspect and judge their smelliness.

Middle Man While performing the interviews with spreadsheet owners whose spreadsheets suffered from the Middle Man smell, we were surprised to see that there was one case in which middle man formulas were passing on information *within* the worksheet. We had not anticipated this use of middle man formulas, and we also did not find this kind of Middle Man smell in the spreadsheets of the EUSES

corpus.

When we asked the subject why these values were passed within the worksheet, he answered that "it is easy to always have the value in sight when working on the spreadsheet, because then I can see the values I am calculating with". While explaining this he realized that maybe the worksheet had grown too big to still be easy to work with. The subject then came up with a 'refactoring', such as splitting the functionality of the worksheet in two, or 'locking' the values needed in a row or column. As such, the Middle Man smell within a worksheet can be a smell indicating the worksheet has grown too big.

A second, related category of Middle Man occurrences happens when a value is passed along worksheets to be in sight everywhere. This value passing can be implemented in two different ways. Either it is passed from the source sheet to each worksheet individually, for instance from worksheet A to worksheet B, and from worksheet A to worksheet C. This does not cause the Middle Man smell.

However, the value passing can also be done in a chain. This happens when values are passed from worksheet A to worksheet B, and the same values are then passed from worksheet B to worksheet C. This second situation does cause the Middle Man smell, and is risky for several reasons. Firstly, it is not imminent what the source of the data is, since from worksheet C, it looks like the data is coming from worksheet B, and inspection of worksheet B is needed, to see that the data stems from worksheet A. Secondly, it is possible that the 'chain' gets broken somewhere. In this case, Middle Man smell can be a reason to change the structure of the spreadsheet such that each worksheet gets its needed input directly from the source worksheet.

In most cases where we saw a formula value being passed, it was to have the value in sight. It is interesting that the Middle Man smell actually reveals a weakness in the implementation of current spreadsheet systems, since apparently there is a large need among spreadsheet users to have some of their values from other parts of the spreadsheet in sight when working on it. Since there is no feature to do this, the problem is solved with formulas, which might make the spreadsheet bigger and more difficult to understand.

In the case of Middle Man smells, the three subjects found the data flow diagrams a good way to indicate inter-worksheet smells. All three quickly understood what data was being passed, and why a worksheet was marked as Middle Man. As one of them stated "The arrows in the diagram exactly show where data is being passed, so I can judge the structure of the spreadsheet".

Shotgun Surgery In the cases where Shotgun Surgery was observed, all three subjects agreed with the fact that there was a shortcoming in the design of their spreadsheet.

In the most extreme case (spreadsheet 9) there was a worksheet on which 220 formulas depended, spread over 10 worksheets. When faced with this, the subject answered that he understood it would be extremely difficult for someone to modify

this worksheet, since it involved checking those ten worksheets, and possibly also modifying them. In other words, he foresaw the Shotgun Surgery that would be necessary in order to adapt this worksheet. Another reason that Shotgun Surgery exposes a threat to the spreadsheet quality occurred in spreadsheet 3, where 7 worksheets depended on one single worksheet with 97 formulas. It turned out that the spreadsheet had been revised recently, and that some of the references were no longer actually in use. In this case too the subjects felt the need to change the spreadsheet when seeing the annotated data flow diagram, to make it up-to-date and more easy to understand for others. One of the subjects stated "I should really take some time to improve these formulas, since this is already confusing for me, so it must be terrible for others". In the third case there were 3 worksheets depending on the smelly worksheet, far less than in the other two cases. However even in this case the spreadsheet owner agreed that the current structure could be improved to increase understandability.

In the cases with Shotgun Surgery we observed the same reception to the data flow diagrams as with the Feature Envy smell: initially it is useful to identify the smells. However in order to know exactly what causes the smell, spreadsheet owners wanted to inspect the worksheet themselves.

4.8.3 Conclusions

With the results of the EUSES analysis and the case studies, we revisit the research questions.

R_1 **What inter-worksheet smells are the most common, and why?** In the first evaluation, Feature Envy is the most frequent smell, and Inappropriate Intimacy comes second. In the second evaluation with 10 industrial spreadsheets, this was almost as high – second from the top.

As we have stated before, we believe these two smells are both due to the fact that it is difficult for end-user programmers to create the right abstraction for their worksheets. This calls for an explicit visualization of the design, as provided by our dataflow diagrams.

R_2 **How do inter-worksheet smells expose threats to spreadsheet quality and calculation integrity?**

In the second evaluation, we have learned that the inter-worksheet smells can indeed reveal weaknesses in spreadsheets. Most notable was the fact that upon occurrence of the Shotgun Surgery smell, subjects came up with adaptations of the spreadsheet structure themselves, because they recognized the dangers that the smell was posing.

R_3 **To what extent are enriched data flow diagrams an appropriate way of visualizing inter-worksheet smells?** In general, the ten participants found the dataflow diagrams a good way to indicate inter-worksheet smells. Since the data flow diagram shows the relation of all worksheets, it was easy for the subjects to understand what worksheet was marked as smelly and why. However, in some cases, especially when investigating Feature Envy and Shotgun Surgery,

users wanted to see details, and felt the need to open the spreadsheet and inspect the marked formulas.

In future work we plan to address these limitations, such as to link the data flow diagram visualization with the spreadsheet. This allows the user to navigate from the diagram to the interesting cells in the worksheets and vice versa.

4.9 Discussion

Our current approach to finding inter-worksheet smells helps spreadsheet users to understand the weaknesses in their spreadsheets design. In this section, we discuss a variety of issues that affect the applicability and suitability of the proposed approach.

4.9.1 VBA code, pivot tables and charts

In this stage of the research, we have only taken formulas into account when calculating the spreadsheet metrics to detect the smells. Spreadsheets can contain other constructions as well, such as pivot tables, charts and VBA code. Those too might be *smelly*, lowering the quality of the spreadsheet as a whole. In future work we will turn our attention to detection of smells in other constructs of spreadsheets.

4.9.2 Mental model of a spreadsheet

Both in this chapter and in previous work we have found that spreadsheet users often do not have the correct mental model of the structure of a spreadsheet. When they were shown the visualization of the data flow diagram, they were often wondering about certain arrows. We plan to deepen this research in a future study, for instance, by having users draw the mental model of a spreadsheet, and comparing that to the as-implemented model. In that way we can learn why their is often a difference in a user's perception of the worksheet relationships.

4.9.3 Understanding spreadsheet design decisions

While some spreadsheets in practice contain a manual or documentation on how to use the spreadsheet, we so far have not found spreadsheets containing the design decisions that were taken while building the spreadsheet. In the qualitative evaluation we have seen that it is hard for spreadsheet users to recover this design information from memory. While our data flow diagrams support in learning the current state of the spreadsheet, it does not help in understanding why the spreadsheet was constructed the way it is. In future work we will investigate this fact, by storing the history of a spreadsheet, and supporting spreadsheet users in documenting changes they make.

4.9.4 Threats to validity

A threat to the external validity of our evaluation concerns the representativeness of the EUSES Corpus spreadsheet set. This set, however, is large (containing over 4000 spreadsheets), and is collected from practice. A second threat to the external validity of our evaluation concerns the representativeness of the selected set of the spreadsheets in the employees at Robeco and their spreadsheets. However other papers [Hen94; Pan06] report on industrial spreadsheet stories similar to the ones we found at Robeco, so their practice seems to be representable.

Furthermore, there is a risk of aptitude treatment interaction since all of the 10 participants where involved in a previous evaluation of data flow diagrams, and it could be the case that only the most positive ones responded to our request to participate in this study.

With respect to internal validity, one of the threats is the fact that we did not pick a random sample. This effect can be decreased by using a larger test group in future experiments. We however believe the current test group serves as a good reference group, as the persons varied in their age, function and daily tasks with spreadsheets.

4.10 Concluding remarks

The goal of this chapter is to underline the importance of inter-worksheet smells as a means to assess and improve spreadsheet quality. To that end we have revisited literature on code smells and spreadsheet design. This has resulted in a list of inter-worksheet smells, which we have subsequently evaluated in both a quantitative study on the EUSES corpus and a qualitative evaluation with ten professional spreadsheet users and real-life spreadsheets.

The key contributions of this chapter are as follows:

- The definition of four inter-worksheet smells, based on known code smells (Section 4.4).
- An approach for the automatic detection (Section 4.5) and visualization (Section 4.6) of the inter-worksheet smells.
- An implementation of that approach into our spreadsheet analysis toolkit Breviz (Section 4.7).
- A twofold evaluation of the proposed inter-worksheet smells, first on the EUSES corpus, and secondly with 10 professional spreadsheet users in an industrial context (Section 4.8).

The current research gives rise to several directions for future work. Firstly there is the expansion of the smells analysis to other spreadsheet concepts, such as pivot tables, graphs and VBA code. Secondly it would be interesting to examine

the relation between actual errors in spreadsheets and inter-worksheet smells. Could smell detection have prevented those errors?

Finally, code smells are inseparably connected to refactoring. Hence it would be exiting to try to create refactoring strategies and refactoring tools for spreadsheet systems, and test their applicability with professional spreadsheet users.

Chapter 5

Detecting Code Smells in Spreadsheet Formulas

5.1 Introduction

The use of spreadsheets is very common in industry, Winston [Win01] estimates that 90% of all analysts in industry perform calculations in spreadsheets. Spreadsheet developers are in fact end-user programmers that are usually not formally trained software engineers. There are many of those end-user programmers, more than there are traditional programmers, and the artifacts they create can be just as important to an organization as regular software. Technically, spreadsheets also have similarities to software. One could view spreadsheet formulas as little pieces of source code, since both consist of constants, variables, conditional statements and references to other parts of the software. It therefore seems logical to research what principles from software engineering are also applicable to spreadsheets.

In previous work [Her12b] we have defined code smells between worksheets, such as high coupling between worksheets and *middle men* worksheets. The evaluation of those smells showed that they can indeed reveal weak spots in a spreadsheet's design. In this chapter we follow that line of thought, but focus our attention on smells within spreadsheet formulas. To that end we present an set of *formula smells*, based on Fowler's code smells. We subsequently define metrics for each of the formula smells, to enable the automatic detection of the smells. We then describe a method to detect these formula smells. Our detection approach uses thresholds to divide the severeness of each formula smell into low, moderate, and high. The thresholds are based on the analysis of 4,233 spreadsheets from the EUSES corpus [Fis05b]. Thereon we address the issue of communicating identified smells to spreadsheet users. We choose to do this within the spreadsheet itself, with a spreadsheet *risk map*, a colored overlay on the spreadsheet, indicating risk

in the spreadsheet formulas. Finally we evaluate the catalog of smells in two ways, with a quantitative and qualitative evaluation. We perform a quantitative evaluation on the EUSES spreadsheet corpus. The qualitative analysis was performed with ten real life spreadsheets and their developers from industry. With both studies we aim to answer the three research questions: R_1 What formula smells are most common, and why? R_2 To what extent do formula smells expose threats to spreadsheet quality? R_3 To what extent are risk maps an appropriate way to visualize formula smells?

The results of these evaluations show that formula smells can indeed reveal weaknesses, and even find real mistakes in a spreadsheet. The risk maps, although not yet perfect, are a good aid in helping to locate and understand formula smells.

5.2 Formula smells

We define a number of *formula smells*, based on the existing work in the field of source code smells, initiated by Fowler [Fow99]. Smells in source code indicate suspicious parts that the developer might want to refactor to improve readability and minimize the chance of future errors. Formula smells are inspired by source code smells: they indicate formulas that are suspicious; not easy to read or error-prone. In the following we present our set of formula smells plus ways to refactor them.

5.2.1 Multiple Operations

One of the most well-known code smells is the Long Method. Inspired by this code smell, we define the formula smell *Multiple Operations*. Analogous to a long method, a formula with many different operations will likely be harder to understand than a shorter one. Especially since in most spreadsheet programs, there is limited space to view a formula, causing long formulas to be cut off.

A corresponding refactoring is the division of the Multiple Operations over multiple cells in a spreadsheet. For instance, instead of putting *SUM(A1:A6)*(B1+8)/100* in one cell, this could be split into two cells, one for the SUM and another for the division that are subsequently multiplied.

5.2.2 Multiple References

Another code smell we use as a basis is the Many parameters code smell. A method that uses many input values might be split into multiple methods to improve readability. The formula equivalent of this smell occurs when a formula references many different other cells, like *SUM(A1:A5; B7;C18;C19;F19)*. In this case the essence of the formula is clear; some cells are summed. However locating the different cells that are contained in the sum can be challenging.

In this case there are several options to refactor. Firstly, like in the case of a formula with many operations, we could split the formula into several smaller formulas, each performing one step of the calculation. A second option is the relocation of the cells in the spreadsheet. One solution is to place the values in B7;C18;C19;F19 in A6 until A10, and to rewrite the formula as *SUM(A1:A10)*.

5.2.3 Conditional Complexity

Fowler states that the nesting of many conditional operations should be considered a threat to code readability, since conditionals are hard to read. Since spreadsheet systems also allow for the use of conditionals, spreadsheet formulas are at risk of this treat too. We hence consider a formula with many conditional operations as smelly, like the formula *IF(A3=1,IF(A4=1,IF(A5<34700,50)),0)*, because of the many conditional branches, the formula is hard to read.

To reduce conditional complexity of a formula, again it could be split into multiple steps, by putting each branch of the if in a separate cell, turning our example formula into *IF(A3=1,B1,B2)*, where cell B1 contains *IF(A4=1, IF(A5<34700, 50))* and B2 contains 0. B1 could again be refactored in this fashion. By separating the ifs, it is easier to understand what happens in each case. A different option is to combine multiple if formulas into one *SUMIF* or *COUNTIF* formula, by putting the branches of the if in separate cells.

5.2.4 Long Calculation Chain

Spreadsheet formulas can refer to each other, hence creating chains of calculation. To understand the meaning of such a formula, a spreadsheet user has to trace along multiple steps to find the origin of the data. Nardi and Miller [Nar90] described that spreadsheet users find tracing a long calculation chain a tedious task.

To lower the number of steps of a calculation chain, steps of the chain could be merged into one cell. Note that there is a trade off between this metric and Multiple Operations and Multiple References. When they are lowered, this metric will be increased, and vice versa. Such trade offs occur in source code smells too.

5.2.5 Duplicated Formulas

Finally there is the *duplication* code smell, which indicates that similar snippets of code are used througout a class. This is a concept common in spreadsheets too, where some formulas are partly the same as others. Consider, for example, a worksheet that contains a cell with formula *SUM(A1:A6)+10%* and a second formula *SUM(A1:A6)+20%*. This formula exhibits *duplication*; the part *SUM(A1:A6)* is contained in more than one formula. Duplication is suspicious for two reasons. Firstly it poses a threat to maintainability, since when the duplicated part of the formula is adapted, this adaptation has to be performed at multiple places. This could be forgotten by the spreadsheet user, or a mistake could be made in some

of the cases. Secondly, there is an impact on readability, when long parts of the formula are duplicated, it is hard to see how they differ from each other.

Duplicated formulas can be refactored by putting the shared subtrees in a separate formula and replacing the subtree with a reference to that formula.

5.3 Formula metrics

To identify smell occurrences automatically, we make use of metrics, an approach common in the automatic detection of code smells [Moh10]. We follow our approach outlined in [Her12b] defining a metric to detect each of the formula smells in spreadsheets. This method entails the definition of a metric for each of the smells to indicate the presence of that particular smell.

Multiple Operations

We measure the length of a formula in terms of the total number of functions that the formula contains.

Multiple References

This metric is counted in terms of the number of ranges a formula is referring to.

Conditional Complexity

Conditional complexity is measured in terms of the number of conditionals contained by a formula.

Long Calculation Chain

This metric is defined as the length of the longest path of cells that need to be dereferenced when computing the value of the formula.

Duplicated Formula

For the localization of this smell more information about spreadsheet formulas is needed. Consider the spreadsheet in Figure 5.1. All the formulas in column E calculate the minimum of the four cells left to it, followed by the addition of a certain percentage. We argue that in this case, duplication should be detected. However, looking at the formulas, they do not look similar. We therefore will use the *relative R1C1 notation* when detecting duplication.

In the relative R1C1 notation, references to other cells are expressed relative to the cell containing the formula. *MIN(A2:D2)* in cell E2 is written as *MIN(RC[-4]:RC[-1])* in relative R1C1 notation. With this notation, all formulas in Figure 5.1 contain the subtree *MIN(RC[-4]:RC[-1])*, with different percentages. With this, we will measure the duplicate code smell as the number of formulas, located

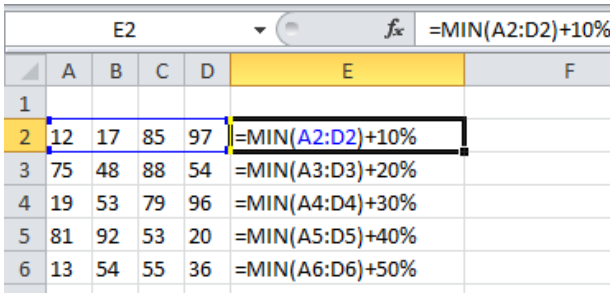


Figure 5.1: *Formulas containing similar subtrees*

in the same worksheet and expressed in relative R1C1 notation, with which a formula shares at least one proper subtree. We exclude the entire tree as subtree, since having the same R1C1 formula in an entire row or column is the usual way of defining a formula in spreadsheets.

5.4 Determining smell thresholds

In order to use the metrics as smell indicators, we determine thresholds for each of the metrics. We do this by analyzing a large body of spreadsheets and based on the values for the metrics we find in that large body of spreadsheets, we set the thresholds for the metrics that indicate the smell.

The body of spreadsheets we use is the EUSES Spreadsheet Corpus [Fis05b]. This corpus consists of more than 4,223 real life spreadsheets, from all sorts of domains, ranging from educational to financial, and from inventory to biology. It was created in 2005 and has since then been used by several researchers to evaluate spreadsheet algorithms, for instance [Abr06].

The total of 4,223 spreadsheets together contain 15,015 worksheets and 426,986 formulas. A spreadsheet however often contains many rows with formulas that are equal in the relative R1C1 notation, which we call *unique* formulas. We collect the metrics for all unique formulas, of which there are 55,736 in the EUSES corpus.

Figure 5.2 shows on overview of the metric values for all unique formulas in the EUSES corpus. As can been seen in this figure, the metrics all follow a power law like distribution, having most of their variability on the tail. We therefore propose to select the values at 70, 80 and 90% of the metric values, which will correspond to risk levels low, moderate and high. These are percentages that are also common in the analysis of source code smells [Alv10]. Table 5.1 shows the thresholds that follow from the given selection for the five formula smells.

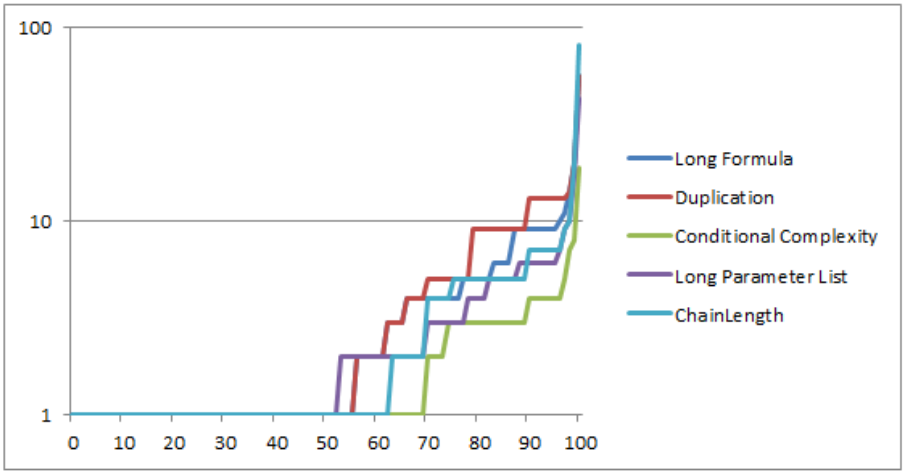


Figure 5.2: *A quantile plot (% of formulas) for the five metrics for formulas in the EUSES corpus, with a logarithmic scale for the y axis*

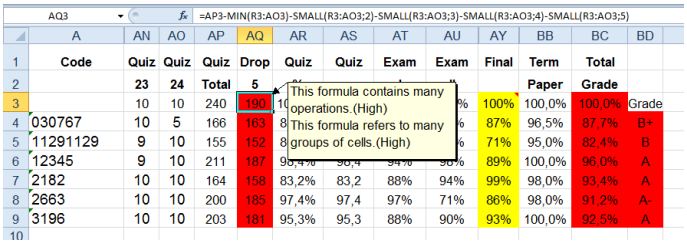
Smell	70%	80%	90%
Multiple Operations	4	5	9
Multiple References	3	4	6
Conditional Complexity	2	3	4
Message Chain	4	5	7
Duplication	6	9	13

Table 5.1: *Thresholds for the metrics that indicate the formula smells*

5.5 Risk maps

Having established a method to detect the formula smells, in this section, we investigate a way to communicate the located smells to spreadsheet users. We have chosen to show the formula smells inside the spreadsheet itself. We have seen in previous work that, when reasoning about formulas, spreadsheet users like to see the context of the formula [Her11]. We therefore use a colored overlay over the spreadsheet that indicates the formula smells, inspired by other spreadsheet tools like UCheck [Abr07a; Cha09; Erw09], combined with pop-ups showing what smell is detected, when the cell is selected by the user. These pop-ups are similar to those used in UFix [Abr05a]. We call this the spreadsheet *risk map*.

We attach a comment to each colored cell, so when a spreadsheet user clicks a cell, an explanation about the located smell will pop up. The three different risk levels are marked with different colors; yellow for low; orange for moderate and red for high.



	A	AN	AO	AP	AQ	AR	AS	AT	AU	AY	BB	BC	BD
1	Code	Quiz	Quiz	Quiz	Drop	Quiz	Quiz	Exam	Exam	Final	Term	Total	
2		23	24	Total	5						Paper	Grade	
3		10	10	240	190	10			%	100%	100,0%	100,0%	Grade
4	030767	10	5	166	163	8			%	87%	96,5%	87,7%	B+
5	11291129	9	10	155	152	8			%	71%	95,0%	82,4%	B
6	12345	9	10	211	187	95,4%	95,4	94%	96%	89%	100,0%	96,0%	A
7	2182	10	10	164	158	83,2%	83,2	88%	94%	99%	98,0%	93,4%	A
8	2663	10	10	200	185	97,4%	97,4	97%	71%	86%	98,0%	91,2%	A-
9	3196	10	10	203	181	95,3%	95,3	88%	90%	93%	100,0%	92,5%	A
10													

Figure 5.3: *A spreadsheet with its risk map*

5.6 Implementation

The generation of the risk maps is incorporated into the existing spreadsheet analysis system Breviz [Her11]. Breviz is implemented in C# 4.0 using Visual Studio 2010. It utilizes the Gembox component to read Excel files. [1] Breviz reads an Excel file and calculates the metrics described above and subsequently generates the spreadsheet risk map. Furthermore the metric values are stored in a SQL Server 2008 database to perform the threshold detection.

5.7 Evaluation

In order to evaluate the formula smells, metrics, thresholds, and risk map visualization, we perform two separate evaluations. In the first evaluation we turn our attention back to the EUSES Spreadsheet corpus. We analyze all spreadsheets in the corpus and investigate the occurrence of the five spreadsheet smells. With this analysis, we seek to find a preliminary answer to research question R_1: what formula smells are common, and why.

For the second evaluation, aimed at a deeper understanding of R_1, plus answers to R_2 and R_3, we ask ten professional spreadsheet developers for access to their real life spreadsheets. We let our tool Breviz identify possible formula smells and show the participants the generated risk map. We thereon ask the spreadsheet users to reflect on the identified spreadsheet smells, in a semi-structured interview.

The following sections describe the two evaluations in more detail.

5.8 Smell occurrences in the EUSES Corpus

5.8.1 Goal

The objective of the first study is to understand how common the five formula smells are, given the thresholds we have selected. While the thresholds were

[1] http://www.gemboxsoftware.com/spreadsheet/overview

chosen such as percentages of formulas containing a smell, here we are interested in the distribution of *smelly* formulas across spreadsheets.

5.8.2 Setup

We use the Breviz tool to analyze the spreadsheets in the EUSES corpus for the presence of formula smells and their severity. Per spreadsheet the tool outputs the metric values for each of the five formula smells. We use this data to analyze the distribution of the formula smells over the three metric categories; above the 70%, 80% and 90% thresholds.This gives a first idea of the distribution of the formula smells over the spreadsheets.

Smell	$> 70\%$	$> 80\%$	$> 90\%$
Multiple References	23.8%	18.4%	6.3%
Multiple Operations	21.6%	17.1%	6.3%
Duplication	10.8%	7.1%	3.7%
Long Calculation Chain	9.0%	7.9%	3.3%
Conditional Complexity	4.4%	3.0%	1.1%
Any of the above smells	42.7%	31.4%	19.7%

Table 5.2: *Percentage of spreadsheets in the EUSES corpus that suffer from at least one of the five spreadsheet smells in EUSES corpus, for the three thresholds*

5.8.3 Results

Table 5.2 shows the results of the first evaluation. As shown in this Table, Multiple Operations and Multiple References are most common in the EUSES Corpus. This is consistent with previous spreadsheet experiments, where it has been shown that spreadsheet are often adapted by their users [Her11]. In that process, spreadsheet formulas tend to get longer and more complex. As opposed to software development, where code is sometimes rewritten to improve readability or maintainability, the answers of the ten subjects of the second evaluation (see below) have taught us that this is not common practice among spreadsheet creators. Hence, when a formula has become long and complex, it is likely to remain that way.

Third and fourth come the Duplicated Formula and Long Calculation Chain. These two smells share the property that they are not immediately visible to the spreadsheet user. In most modern spreadsheet systems, when a cell is clicked, the formula it contains is shown. However in the case of Duplicated Formula and Long Calculation Chain, the formula does not reveal where the calculation chain of the formula ends, and with what formulas a cell shares subtrees. So it is

interesting to see that around 10 percent of spreadsheets suffer from a smell that is not visible to a spreadsheet user that is not explicitly looking for it.

Conditional Complexity is the least common smell. This is surprising, since we have seen before that conditional operations are quite common in spreadsheets. We therefore dived into this phenomenon deeper. We found that of the total of 426,986 formulas in the corpus, 5,976 contain at least one conditional operation, this is 1.4% of all formulas. These formulas are divided over 380 spreadsheets, which amounts to 22.2% of the spreadsheets with formulas. We can hence state that the use of conditional operations is relatively common.

However, only 92 of the spreadsheets (5.3%) contain formulas with more than 2 conditional operations in one formula, adding up to only 695 formulas (0.2%). Evidently, the nesting of conditional formulas is not very common. We will hence devote attention to this fact in the second evaluation.

Regarding the thresholds, given our current choices a substantial percentage of the corpus suffers from spreadsheet smells, especially in the low category. In the second case study we will continue to investigate the thresholds, by observing how spreadsheet users from industry feel about formulas that are marked as smelly by these thresholds.

5.9 Formula smells in an industrial case study

5.9.1 Goal

The aim of our second evaluation is investigating which of the formula smells actually poses a threat to spreadsheets (R_2), and whether risk maps help spreadsheet users find and understand formula smells (R_3). To determine this, we have surveyed 10 spreadsheet users and interviewed them about a spreadsheet that they often work with and that they found was large and complicated.

5.9.2 Setup

For the second evaluation we interviewed 10 professional spreadsheet users from the financial domain. We conducted our evaluation at Robeco, a Dutch asset management company with approximately 1600 employees worldwide, and over 130 billion Euro worth of assets under management.

We invited 27 participants of a previous study performed at Robeco [Her11] to participate in this evaluation, where they were asked to provide us with a spreadsheet that they worked with regularly. We explained to participants that we were going to analyze the quality of their spreadsheet, and discuss it with them. We provided subjects with a list of the five formula smells and a short explanation of each smell, so they could study this before the experiment. During each of the 10 case studies, the procedure was as follows. First we asked the subject to explain the spreadsheet to us. We then analyzed the spreadsheet and

generated the spreadsheet risk map, which we showed to users in Excel 2010. We subsequently let subjects inspect the risk map, and asked them in a semi-structured interview setup, for each of the located formula smells 1) whether they understood the smell that was identified and 2) whether they thought this smell posed a threat to spreadsheet understandability, and if yes, why, and how severe the threat was, according to them. We finally asked them how the risk map and the pop-up helped them in understanding the formula smells.

5.9.3 Results

Table 5.3 shows an overview of the spreadsheets used in the case study in terms of the numbers of worksheets, cells, formulas, unique formulas and file size in Kb. The five final columns indicate the number of unique formulas in the spreadsheets suffering from the given smell, Multiple Operations (MO), Duplicate Formula (DF), Multiple References (MR), Conditional Complexity (CoC) and Long Calculation Chain (CaC). As can be seen from this table, formula smells occur frequently in the ten spreadsheets. The following describes the result of the case studies in detail.

ID	Spreadsheet Description	#Wrks.	#Cells	#Form.	#Uniq.	Size	MO	MR	DF	CaC	CoC
1	Calculate dividend	5	13,697	6,012	53	183	14	6	3	7	-
2	Investment strategies	5	21,600	3,031	98	605	9	8	4	3	-
3	Model energy companies	14	82,000	14,742	531	826	58	31	11	-	-
4	Valuation of swaps	8	31,485	5,034	67	1,690	17	-	4	-	21
5	P&L overview of all traders	10	17,263	9,152	142	4,261	26	23	17	7	-
6	Risk overview for different sectors	9	9,190	148	12	332	4	3	2	-	-
7	Comparing calculation models	14	24,580	3,388	39	348	17	14	23	-	-
8	Planning of trades	1	2,329	1,683	64	76	15	33	-	4	-
9	Report interest and liquidity risk	25	59,116	17,930	117	1,693	32	18	-	9	-
10	General ledger data for analysis	11	11,906	3,049	56	1,732	10	23	7	-	4

Table 5.3: *Characteristics and number of smells above the 70% thresholds of spreadsheets used in the case study.*

General observations

When confronted with the most smelly formula cells, participants often needed time to explain the formula. In all cases the participants expressed statements like "what was this formula again?" or "let me just have a quick look". A commonality among the ten cases in the study is the fact that all participants immediately recognized the impact a smelly, hence complex formula could have on spreadsheet formulas understandability. When we discussed the most complex formulas in the spreadsheet with the participants, they indicated that it was going to be very hard to understand or adapt this formula for someone else than the spreadsheet's creator.

Most participants (8) had never considered the situation where another spreadsheet user had to understand their formulas that much. One of our previous studies confirms the importance of such *transfer scenarios* [Her11]. What we found in our present study, however, is that participants did not realize prior to this study that keeping formulas short and simple would help future users of the spreadsheet understand it better and faster. A side effect of our study was increased awareness with our participants that they should take maintainability into account when building and adapting spreadsheets.

We found that the three thresholds and the corresponding coloring help subjects estimate severeness of the detected smell. One of the subjects compared this to the triangle function in Excel. This function marks potential errors, like calculations over empty cells with a small green triangle at the bottom of a cell. He stated:"That function is annoying, since many cells get colored. Because you have different shades, I can start inspecting the red ones, and ignore the yellow ones for now".

Regarding the values of the thresholds, we discussed each colored cell with the spreadsheet owner, systematically going through the worksheets. In all but one case the subjects agreed with the classification of the formulas. Only spreadsheet user S_3 stated that he felt that the system was too strict. His spreadsheet contained 3 cells with five different references and four operations. These cells were hence marked as having both the Multiple Operations and the Multiple References smell, while the user still found this acceptable. In the other formulas in his spreadsheet where these smells were located, he did found they were smelly, since the metric values for those formulas where higher than respectively 5 and 4. So from the perspective of this user the thresholds should be higher, however as stated above, he was the only one; the other nine subjects stated all marked formulas were indeed smelly.

Multiple Operations

Findings Multiple Operations were found in all ten spreadsheets, making them the number one common smell. In all cases we found that the subjects said that keeping the formulas short makes them easier to read and understand. Two of

the subjects believed that formulas with many operations are often a cause of errors, saying "the chance of errors in such a long formula is so much bigger; when I find errors, it is almost always in long formulas". When asked for the reason that Multiple Operations where created, all subjects stated that this was an evolutionary process. Multiple Operations are hardly ever created at once. They are the result of the repeated adaptation of a formula, adding operations as the spreadsheet changes. As one of the subjects stated "Usually it starts with just a sum, but than you want to round it, add something and before you know it, the formula is two lines long". The two subjects above — the ones who had realized the error risk of Multiple Operations— did try to minimize formula length. However, sometimes, for instance, when a deadline was approaching, Multiple Operations were introduced anyway. There was no common practice among the spreadsheet users to restructure the spreadsheet after such a deadline. One of these two subjects mentioned "when it works, it works. No one really cares how it is done".

We found that the risk map helped in the case of Multiple Operations. In the case where this smell was located, the smelly cells were clearly marked with a color (yellow, orange or red). Hence, the reason why the smell was detected was immediately clear; many subjects stated something like "I understand why this formula is selected by your system, it is quite long."

Conclusions Spreadsheet users feel that Multiple Operations are more error prone than shorter formulas. Since Multiple Operations are harder to read, it is more difficult for users to spot an error, so formulas with multiple operations will less likely be corrected when they are wrong. Multiple Operations are often the result of changes to the spreadsheets, and the refactoring of complex formulas is not something that spreadsheet users do.

Multiple References

Findings Experiences with Multiple References were similar to those with Multiple Operations; when confronted with the smelly cells, it took the nine subjects a considerable amount of time, in the longest case even ten minutes, to explain the formula. This made them realize that it would be very hard for others to understand and adapt the formula, especially since locating the references can be a challenge. Excel supports users in locating the references by coloring the referenced cells. However, if there are many references and colors users find this feature to be more annoying than helpful as confirmed by nine of our participants. One of the subjects stated, when looking at a formula that referred to no less than 17 ranges "this formula is a real puzzle".

In this case, as opposed to the Multiple Operations smell, some participants did not immediately understand how to adapt this formula to make it less smelly. When asked, one of the participants even stated "but I need all that input to make the calculation". Splitting the formula into different steps seemed more difficult than with the Multiple Operations. In that case the formulas consist of different

operations, and the splitting would consist of separating the operations. In this case however, we encountered formulas like SUM(A1:A5;B6;D7;E12), of which it was not immediately clear to the spreadsheet users how to improve it. It can be split into multiple steps, but what steps are logical is not so easy to determine for the user. We asked the nine participants to describe how they were going to split the formula, and only one was able to formulate a strategy. The other hesitated, one of them stated "I don't know where I should start, because I don't remember how I made this formula". As an alternative, cells could be moved, such that this formula will refer to one range. This made the participants hesitate even more. They clearly felt that moving formulas around was a tricky operation, since the effect of this move on other formulas is not clear. One of the subjects that tried to lower the references said "if I move this, what will happen to the other formulas? I would like to preview that".

For this smell again, the risk maps are a good way of conveying this smell. Formulas with many references were colored red, orange or yellow; and hence attracted the attention of the spreadsheet users. Clicking the formula revealed easily that the formula had too many references.

Conclusions Subjects found that formulas with many references are not easy to understand, since finding all references can be difficult. Even though the risk was understood, subjects found it hard to come up with the right refactoring to overcome this smell. This is partly caused by the fact that a long list of references can indicate that the placement of formulas is not optimal, and hence this smell can also reveal a weakness in the organization of the spreadsheet.

Refactorings to the location of formulas were found especially hard for the subjects, and support for this, like a preview, is definitely a promising avenue for future research.

Finally we found it interesting that Excel's feature to color the cells referenced by a formula is only helpful in cases with few references (typically above 6 it got confusing for the participants). There is apparently a need for better support in locating references.

Duplicated Formula

Findings In the evaluation we found that the cases in which duplicated formulas are detected can be divided into two distinct categories.

- *Sharing Subtrees:* Different formulas are sharing a subtree, and there is an opportunity for refactoring.
- *Rare Formulas:* There is one formula that differs slightly from its neighbors, and therefore shares a subtree with these neighbors.

Figures 5.4 and 5.5 illustrate the two categories. In Figure 5.4 the highlighted formula (in cell B13) shares the subtree *SUM(B7: B11)* with the formula in cell B12. The same subtree occurs twice, so it might be better to replace *SUM(B7:*

B11) in B13 with a reference to B12. In Figure 5.5 however something different is happening. The selected formula (E4) shares a subtree with the other formulas in the same row, each summing up the values of the three cells above it.

However, there is a small difference with the other formulas, which is the '+0.1', denoting the formula as *rare*, it is not like other formulas in the worksheet. Excel itself recognizes the risk of this type of formulas. This is one of the possible errors that Excel marks with a green triangle in case a formula in a row or column differs from its direct neighbors. Others are the referencing of an empty cell, and numbers formatted as text.

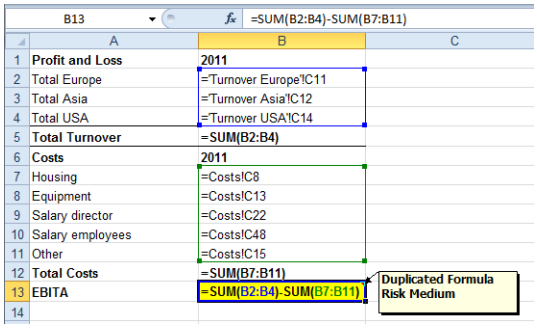


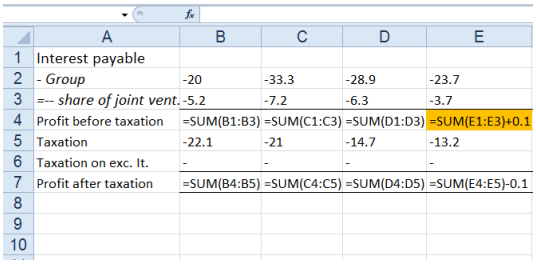
Figure 5.4: *A formula with duplication*

In our ten spreadsheets, we encountered two cases of a Rare Formula. In both of them, a formula was marked as having a lot of duplication, turned out to differ from the other formulas in its column, while the participants stated that this was actually wrong. Thus, the smell indicated an actual error in the spreadsheet.

Note that Excel was able to mark only one of these cases as possibly dangerous: Excel spots discrepancies between directly adjacent cells, whereas one of these errors involved cells disconnected from each other.

Opinions differed on the six cases in which sharing subtrees were encountered. Four of the subjects understood that having the references at one place made the structure of the spreadsheets better. However the remaining two saw no harm in the duplication of formulas. This is notable, since with source code many people agree that duplication should be avoided.

With respect to the risk maps, we noticed that the current implementation of the pop up as does not yet provide enough information: It only marks the formula that shares subtrees with many formulas, but does not indicate with what cells the subtrees are shared. This resulted in participants looking through formulas in the spreadsheet to find the formulas that shared a subtree. A possible solution could be to include the list of sharing formulas in the pop up, or create an Excel plug in that highlights the sharing formulas when a formula suffering form duplication is selected. We will address this in future work.



	A	B	C	D	E
1	Interest payable				
2	- *Group*	-20	-33.3	-28.9	-23.7
3	=-- *share of joint vent.*	-5.2	-7.2	-6.3	-3.7
4	Profit before taxation	=SUM(B1:B3)	=SUM(C1:C3)	=SUM(D1:D3)	=SUM(E1:E3)+0.1
5	Taxation	-22.1	-21	-14.7	-13.2
6	Taxation on exc. It.	-	-	-	-
7	Profit after taxation	=SUM(B4:B5)	=SUM(C4:C5)	=SUM(D4:D5)	=SUM(E4:E5)-0.1
8					
9					
10					

Figure 5.5: *A rare formula*

Conclusions Rare formulas can reveal true weaknesses and even errors in spreadsheets, and spreadsheet users agree with that.

However, the refactoring of duplicate pieces of formulas — in source code refactoring very common— is not considered to be am improvement to all spreadsheet users.

Long Calculation Chain

Findings This smell triggered most discussion with the five subjects whose spreadsheets were diagnosed with this smell.

The main reason was the fact that the risk maps do not provide enough information to understand this smell immediately. When a certain cell suffers from the Long Calculation Chain smell at the 70% level, this means that the path from this formula to the beginning of the longest calculation chain is at least 5 steps. The cells that are included in this calculation chain were not shown in the pop up. This led to spreadsheet users repeatedly stepping through formulas to check whether a formula indeed had a long calculation chain; and whether that was necessary.

Two of the subjects found that the calculation chain (in one case 10, in the other 17 steps) was indeed long, and that some restructuring would improve readability. The other three subjects found that, although the number of steps was high, this was necessary in order to calculate the needed values. We did notice that it is easier to trace and understand the different calculation steps when they are located in the same row or column. When we asked the five subjects about this, they concurred. This means there is a need for an additional metric based on the location of the cells involved in the calculation chain. We will look into this in future research.

Furthermore there is the trade off between Multiple Operations and Multiple References on one the hand, and Long Calculation Chain on the other. When discussing this phenomenon with the five subjects, we learned that they felt in need of guidance where the right balance is. Hence, better support for managing this trade off is needed. This might be done with our risk maps or with other interfaces to help users to find the proper balance between the formula smells.

Conclusions Long Calculation chains are relatively common, but are difficult to refactor for spreadsheet users. Hence more support to help users to understand and refactor this smell is necessary.

Conditional Complexity

Findings This metric was the least common in the case study, similar to the finding in the evaluation of the EUSES corpus. In the two spreadsheets in which it was located, the risk maps easily helped in locating the Conditional Complexity smell. When the users selected the cells suffering from the smell, they learned from the pop up that nested conditionals were found in the formula.

The two subjects understood and even apologized for the detected smell, stating "I know this formula is too hard, I was trying to get something to work, and then it just remained like that". Both subjects were well aware of the fact that nesting more than two conditional formulas was not such a good idea.

Conclusions The Conditional Complexity smell is in fact already known to spreadsheet users. Apparently there is some notion among the spreadsheet users that conditional operations are complex and should be handled with some care, probably explaining the low occurrence of this smell.

5.10 Answers to research questions

With the results of the EUSES analysis and the case studies, we revisit the research questions.

R_1 *What spreadsheet smells are most common, and why?* In both evaluations we have seen that Multiple Operations and Multiple References are the most common smells, and from the second evaluation we have learned that this is often caused by the modification of a spreadsheet, sometimes under time pressure. Since there is little awareness of the risks of Multiple Operations, spreadsheet users seem not to be concerned too much about maintainability of formulas. They keep extending formulas with more operations and more references, causing formulas to become long and complicated.

R_2 *To what extent do formula smells expose threats to spreadsheet quality?* We found two actual faults in a spreadsheet by looking at the Duplication Smell. With respect to the other smells, the concern caught is lack of understandability. Spreadsheet users found that our current smell detection strategies reveal the formulas that are the least maintainable. These formulas will be time consuming to change, and changes made will be more error prone.

R_3 *To what extent are risk maps an appropriate way to visualize spreadsheet smells?* The strengths of risk maps include their simplicity and the fact that the visualization is shown within the context of the spreadsheet. Seven subjects explicitly stated they liked the risk maps, posing statements like "these colors draw my attentions to the formulas that deserve a second check". Users furthermore

appreciated the different levels of the smells, allowing them to inspect the worst formulas first. For the Long Calculation smell, however, additional interface elements are needed, in order to help spreadsheet users understand the cause of the smell.

Beyond risk maps, three of the subjects asked for a general score of the quality of their spreadsheet. Although we could provide them with the number of smells and their severity by looking into our database, an aggregation of the metrics is not provided by the current prototype. This could, for instance, be added by generating an extra worksheet in the spreadsheet in which overviews of all metrics are shown.

5.11 Discussion

5.11.1 Named ranges

In the current set of smells we have not taken into account *named ranges*, a spreadsheet feature allowing users to assign a name to a number of cells. We encountered named ranges in one of the case studies, where a formula that summed a named range, SUM(NamedRange), was marked as having the Many Reference smells. Initially the subject did not understand why it was marked as referring to many different ranges, since there was only one reference. The named range itself however consisted of several separate ranges. This raises the question whether we think this is smelly, and why. Note that the smells is in fact related to the named range itself —it is probably not a good idea to create a named range consisting of multiple ranges— rather than to the formula referencing the named range.

5.11.2 Applicability of the risk maps

Our risk map visualization exhibits limitations if the smell in question addresses concerns not exclusively contained in the smelly formula itself. This explains why some subjects were dissatisfied with the pop-ups of Long Calculation Chain and Duplicated Formulas, which essentially require information from cells outside the smelly formula itself. In future research we will explore how to present smells at different levels of abstraction in one integrated view.

5.11.3 Spreadsheet evolution

While performing the case study, subjects shared that spreadsheets tend to undergo many changes during their life time (an observation also made in [Her11]), and that these changes can lead to a degradation of formula quality. This is an issue that warrants further investigation, calling for a longitudinal study of spreadsheet quality, and opening up the possibility of spreadsheet quality monitoring tools.

5.11.4 Threats to validity

A threat to the external validity of our quantitative evaluation concerns the representativeness of the EUSES Corpus spreadsheet set. This set, however, consists of 4223 spreadsheets covering 11 different domains. Furthermore, it has been used in many other studies and is collected from practice.

A threat to the external validity of our qualitative evaluation concerns the representativeness of the selected set of employees of Robeco and their spreadsheets. However other papers [Hen94; Pan06] report on industrial spreadsheet stories similar to the ones we found at Robeco, so their practice seems representative. Further studies are however needed to generalize our findings.

With respect to internal validity, one of the threats is the fact that we did not pick a random sample of people. This effect can be decreased by using a larger test group in future experiments. We however believe the current test group serves as a good reference group, as the persons varied in age, function and daily tasks with spreadsheets. By working with practitioners we tried to maximize the realism of our evaluation, which unfortunately comes at the price of reduced repeatability.

5.12 Related work

Research efforts related to ours include papers that provide spreadsheet design guidelines. Raffensberger [Raf09], for instance advises to merge references that occur only once. He furthermore states that unnecessary complex formulas with many operations and parenthesis should be avoided. Rajalingham *et al.* [Raj00] also propose guidelines to improve spreadsheet quality, which they base on principles of software engineering.

Secondly, there are papers that address common errors in spreadsheets, like [Aya00; Pan98], together with their causes. Powell *et al.* for instance [Pow09] names conditional formulas (which is one of our smells) among the top three of commonly occurring spreadsheet error categories.

Furthermore there is related work on finding anomalies on spreadsheets, for instance the work on the UCheck tool [Abr07a; Cha09; Erw09]. UCheck determines the type of cells, and locates possible anomalies based on this type system. UCheck uses a similar visualization, with colors in the spreadsheet, to indicate found anomalies.

We ourselves have worked on spreadsheet smells in previous work [Her12b]. In that paper we focused on detecting smells between worksheets, like high coupling. That paper followed our earlier work, in which we worked on the visualization of spreadsheets by means of class diagrams [Her10] and dataflow diagrams [Her11].

This chapter differs from our previous work by focusing on detecting smells in spreadsheet formulas. Recently, other work on spreadsheet smells has been published that aims at smells in values, such as typographical errors and values that do not follow the normal distribution [Cun12]. Other recent work by Ba-

dame and Dig [Bad12] also confims the presence of smells in the EUSES corpus and furthermore suggests an approach to support spreadsheet users in removing formula smells by refactoring.

5.13 Concluding remarks

The goal of this research is to investigate the applicability of code smells to spreadsheet formulas as a means to asses and improve spreadsheet quality.

To that end we have created a list of formula smells, based on our experiences with spreadsheets, related work in spreadsheet guidelines and literature on code smells.

We then defined a set of metrics for detecting five formula smells and presented the visualization of these smells with the spreadsheet risk map. We have evaluated the metrics and the risk map with a qualitative and quantitative evaluation. The quantitative evaluation was performed on the spreadsheets from the EUSES corpus. The qualitative evaluation was with spreadsheets from ten professional spreadsheet users from industry.

The key contributions of this chapter are as follows:

- An analysis of the risky types of spreadsheet formulas (Section 5.2) and corresponding metrics (Section 5.3)
- A method to detect (Section 5.4) and visualize (Section 5.5) those smells
- An evaluation of these formula smells on the EUSES corpus (Section 5.8 and with ten professional spreadsheet users and their spreadsheets (Section 5.9).

We have found that spreadsheet formula smells occur frequently, and can pose a real threat to spreadsheet understandability, and can even detect actual errors. Spreadsheet users in our qualitative evaluation found that the risk maps were a good way of indicating formula smells, and that the three thresholds helped them get a feeling of the importance of the located smells.

The current research gives rise to several directions for future work. Firstly, the definitions of the current set of metrics could be refined; as mentioned in the evaluation section, we could split the duplication metric, and add a metric for the location of cells in a long calculation chain. Secondly, some smells ask for a more elaborate visualization, for instance to indicate the balance between Multiple Operations and Long Calculation Chain. Finally, more support for formula refactoring is needed. We plan to investigate means to suggest such refactorings to the spreadsheet user, give them a preview of the result, or even perform them automatically.

Chapter 6

Data Clone Detection and Visualization in Spreadsheets

6.1 Introduction

Spreadsheets are heavily used within companies in many domains, ranging from financial to medical and from educational to logistics. It is estimated that 90% of desktops have Excel installed[Bra09] and that the number of spreadsheet programmers is bigger than that of software programmers[Sca05]. Because of their widespread use, they have been the topic of research since the nineties[Bel93]. However, most papers focus on analyzing and testing the *formulas* in a spreadsheet.

The impact of data on spreadsheet calculations has been somewhat overshadowed by this interest in formulas. However, problems with data can pose threats to a spreadsheet's integrity too. A paper by Ballou *et al.*[Bal87] phrases the problem as follows "...errors in the operational data can influence the determination of the most appropriate forecasting model" and "The manager is unlikely, however, to study the implications of errors in the data that are being projected. Clearly such errors have an impact, but it is not necessarily obvious which are potentially serious and which less". Although this paper is 25 years old, the problem statement is still very valid. In 2003, TransAlta lost US$24 Million because of a copy-paste error in a spreadsheet[1]. More recently, the Federal Reserve made a copy-paste error in their consumer credit statement which, although they did not make an official statement about the impact, could have led to a difference of US$4 billion[2]. These stories, although anecdotal, underline the fact that copy-paste errors in spreadsheets can greatly impact spreadsheet quality.

[1] http://bit.ly/cQRoy8
[2] http://bit.ly/6XwN9t

In this chapter we focus on the occurrence of copy-pasting in spreadsheets by analyzing how the detection of *data clones* can help spreadsheet users in finding errors and improving the quality of their calculations. To that end we study related work in the field of clone detection in source code and come up with an approach to detect data clones in spreadsheets. In addition to exact clones, we also detect *near-miss clones*, those where minor to extensive modifications have been made to the copied fragments[Roy09a].

Our approach is based on existing text-based clone detection techniques[Joh93], we use cell values as fingerprints and remove values that do not occur as formula and plain text. Subsequently, we group values that occur in multiple places into *clone clusters*, to detect groups of cells that are possibly copied.

Detected clones are visualized in two ways. Firstly, we generate a dataflow diagram that indicates how data is cloned between two worksheets, by drawing an arrow between boxes that represent those worksheets. This way, users can see how data is copied through worksheets and files. Secondly, we add pop-up boxes within the spreadsheet to show where data is copied, and in the case of near-miss clones, what cells differ.

This approach is subsequently validated both quantitatively and qualitatively. Firstly, we analyze the EUSES corpus[Fis05a] to calculate the precision and performance of our algorithm and to understand how often clones occur. Secondly, we perform two case studies: one with a large budget spreadsheet from our own university and a second one for a large Dutch non-profit organization, for which we analyzed 31 business critical spreadsheets.

From these three evaluations, we conclude that 1) data clones in spreadsheets are common, 2) data clones in spreadsheets often indicate problems and weaknesses in spreadsheets and 3) our algorithm is capable of detecting data clones quickly with 80% precision and supports spreadsheet users in finding errors and possibilities for improving a spreadsheet.

6.2 Related work

As stated in the Introduction, Ballou *et al.*[Bal87] described the problem of data quality in spreadsheets. More recently, O'Beirne[O'B08] states that "...much present use of spreadsheets is as data manipulation and reporting tools used to bypass the controls around IT development. " And that this "ad hoc integration, transformation, or simple cobbling together is done by the user to get what they need when they need it. This gives rise to many extracted copies of corporate data as imports or query links in spreadsheet files. These personal data stores are often referred to as 'data shadows' or 'silos' or 'spreadmarts' giving rise to 'multiple versions of the truth'." [O'B08] He furthermore cites evidence that "errors in the transfer of data from the field officer forms through to the DEFRA spreadsheet equating to an error rate of 14 percent over the year."

Clone detection in source code has been researched extensively and resulted in numerous clone detection techniques and tools. Bruntink *et al.*[Bru05] give the following overview:

Text-based techniques perform little or no transformation to the raw source code before attempting to detect identical or similar (sequences of) lines of code. Typically, white space and comments are ignored[Joh93; Duc99].

Token-based techniques apply a lexical analysis (tokenization) to the source code and, subsequently, use the tokens as a basis for clone detection[Kam02; Bak95].

AST-based techniques use parsers to obtain a syntactical representation of the source code, typically an abstract syntax tree (AST). The clone detection algorithms then search for similar subtrees in this AST[Bax98].

PDG-based approaches go one step further in obtaining a source code representation of high abstraction. Program dependence graphs (PDGs) contain information of a semantical nature, such as control and data flow of the program. Kommondoor and Horwitz[Kom01] look for similar subgraphs in PDGs in order to detect similar code. Krinke[Kri01] first augments a PDG with additional details on expressions and dependencies, and similarly applies an algorithm to look for similar subgraphs[Kom01],[Kri01].

Other related efforts are Roy[Roy09a; Roy10], who created NiCad, a parser-based and language specific tool that detects both exact and near-miss clones with high precision and recall. Roy and Cordy [Roy10] compared several open source systems and study, among other properties, the occurrence of near-miss clones versus exact clones. They detected significantly higher numbers of near-miss clones than exact clones in the systems under evaluation. For more information on existing code clone detection techniques and tools, we refer the reader to the comprehensive survey by Roy *et al.*[Roy09b].

Our approach is text-based and is most similar to that of Johnson[Joh93], where we use cell values as fingerprints.

Recently, there have been other efforts to apply clone detection to artifacts other than source code, Alalfi *et al.*[Ala12] have successfully applied clone detection to SimuLink models. For this purpose they adapted NiCad and were able to efficiently find exact, renamed and near-miss clones in SimuLink models. To the best of our knowledge, no approach to detect data clones in spreadsheets exists.

Finally, there is our own work on spreadsheet analysis. Previously, we have worked on an algorithm to visualize spreadsheets as dataflow diagrams[Her11], and subsequently on detecting inter-worksheet smells in those diagrams[Her12b]. Recently we have also worked on detecting smells in spreadsheet formulas[Her12c]. This paper is a continuation of our research on smells in spreadsheets, however we shift our focus to detecting and visualizing clones in spreadsheet data. A tweet-sized paper on clones in spreadsheets was recently accepted in Tiny Transactions on Computer Science[Her12a].

6.3 Motivation

In our work with spreadsheet users, we often see that they copy and paste data from one spreadsheet to the other, from to worksheet to the other, and even within worksheets. When data is copy-pasted, or *cloned*, the spreadsheet might become more prone to errors, similar to the effect clones have on a software system.

Research in the field of source code analysis has analyzed the negative effect of clones on quality and maintenance. Mayrand *et al.*[May96] showed that duplicated code fragments can increase maintenance effort. Furthermore a study by Jurgens *et al.* that analyzes industrial code shows that inconsistent changes to code duplicates are frequent and lead to severe unexpected behavior[Jür09]. However, not all clones are harmful, Kapser en Godfrey [Kap08] show that clones can also have a positive impact on maintainability.

Strictly, copy-pasting data in spreadsheets is not necessary: most spreadsheet systems have a feature where data can be linked from one worksheet to another and even from one file to another. In Excel, this can be done by either typing the location of the file in a formula, followed by the name of the worksheet and the cell(s), or by opening both worksheets or spreadsheets and creating the link by clicking, just as one would do with any formula.

So why would spreadsheet users resort to copy-pasting data from one spreadsheet file to the other, if most spreadsheet systems have a 'better' way to do this? In our experience, this practice can have several reasons. Firstly, sometimes users are not aware of a way to link spreadsheets, they do not know how to use the link-formulas. Secondly, users are often unaware of the risks of copy-pasting data and it seems the easiest way.

We do not aim at changing the spreadsheet users's behavior, since that would, most likely, involve changing the process around a spreadsheet and that would be hard to implement in a company. We rather follow an approach that allows users to proceed as they normally would, but mitigate the risks by detecting and visualizing the copy-paste relationships. This enables users to take the appropriate measures in a similarly pragmatic way.

Therefore the aim of this chapter is to *both understand the impact of copy-pasting, and to develop a strategy to automatically detect data clones in spreadsheets.* We refine this goal into three research questions.

R1 How often do data clones occur in spreadsheets?

R2 What is the impact of data clones on spreadsheet quality?

R3 Does our approach to detect and visualize data clones in spreadsheets support users in finding and understanding data clones?

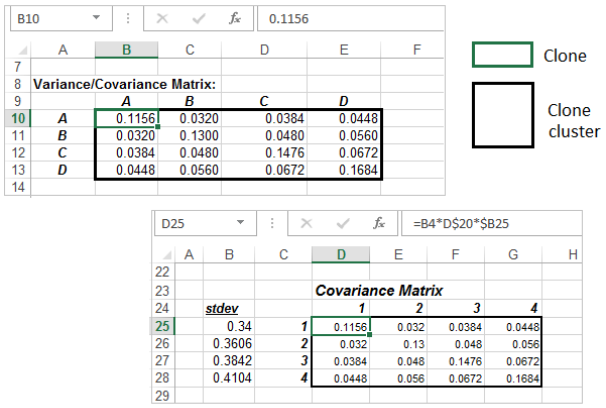


Figure 6.1: *Cells B10 and D25 form a clone. Since all their neighboring cells are also clones, we detect two clone clusters in this example.*

6.4 Data clones

The type of clones we are interested in are formula results that are copied as data for other parts of the spreadsheet. This type of copying is easy in Excel: with 'paste special' a user can copy values only.

We hypothesize that this type of cloning is risky: when formulas or their input is updated and their values change, this results in an extra maintenance task: updating the copies of the formulas too. If this is forgotten, errors could occur. We therefore look for tuples of a formula cell and a constant cell (a cell with a fixed value in it) that contain the same value. We call such a tuple of cells a *clone* and corresponds to what is called a *clone pair* in the clone detection literature.

It can, of course, happen by coincidence that a formula and a cell contain the same value. However, if a group of neighboring cells are all copies, or are all copied, we are probably dealing with data that has been copied by the user. We call such a group a *clone cluster*. In more detail: a clone cluster consists of cells that are all either formula cells or constant cells, and all of them are contained in a clone.

We call two clone clusters *matching* if they contain the same values. Figure 6.1 shows an example of a clone, in the small green rectangle, and two matching clone clusters in the gray rectangles.

Finally, we distinguish between clones clusters of which all values match and *near-miss* clone clusters, which contain cloned cells, but also cells that differ from each other. Near-miss clone clusters can occur when by a user updates a copied cell, but not does not update the original cell.

6.5 Data clone detection

In this section we describe the algorithm with which we detect clone clusters. This algorithm consists of 5 steps, as shown in the overview in Figure 6.2.

6.5.1 Algorithm

In the first step, *cell classification*, we divide the cells into data cells, formula cells and empty cells. For data cells, we only consider cells containing a number. Although strings can be the result of a formula, such as string concatenation or a lookup function, we do not take them into consideration, since strings are usually used for labels and descriptions in spreadsheets. In Figure 6.2 formula cells are colored pink and data cells containing a number are colored green. Note that this classification differs slightly from the cell classification algorithms we have used in previous papers[Her10; Her11]. In our current version, all cells containing a number are considered data cells, and not only cells that are used as input for formulas. Since we are looking for clones, the data cells do not necessarily have to be used in calculations.

In the second step, *lookup creation*, a lookup table of all cells is created, with the cell value as key and a list of locations as the value. Shown in Figure 6.2 the value 0.4101 occurs only in Eff4!B28 and 0.1156 occurs in Problem Data!B10 and Eff4!D25. This step is similar to the creation of fingerprints in Johnson's clone detection approach[Joh93].

The third step, *pruning*, removes all values from the lookup table that do not occur both in a formula and a constant cell, since these cells can never be part of a clone, conform our definition. In the example shown in Figure 6.2, 0.4101 is removed, since it only occurs in a formula and not in a constant cell.

In the subsequent fourth step called *cluster finding*, the algorithm looks for clusters of neighboring cells that are all contained in a clone, and that are all either formula cells or constant cells. The clusters are found by starting with a cell that is still contained in a value of the lookup table. In Figure 6.2, we start with cell Problem Data!B10 that contains 0.1156. Subsequently, this cell's neighbors are inspected. If these neighbors are contained in the lookup table too, the cluster is expanded and their neighbors are inspected. The fourth step of the algorithm results in a list of formula clusters and a list of constant clusters.

In the fifth and final step, *cluster matching*, each formula cluster is matched with each constant cluster. For two clusters to match, they have to contain the same values, i.e. all values in one cluster also have to occur in the second cluster. If one cluster is bigger than the other, all values of the smaller cluster have to be found in the second cluster. Furthermore, there may not be a formula link between the formula cells and the cloned value cells, since we do not consider a link (i.e. =Sheet2!B1) to be a clone. In Figure 6.2, the two gray clusters match, since all values of the left cluster match with values on the right.

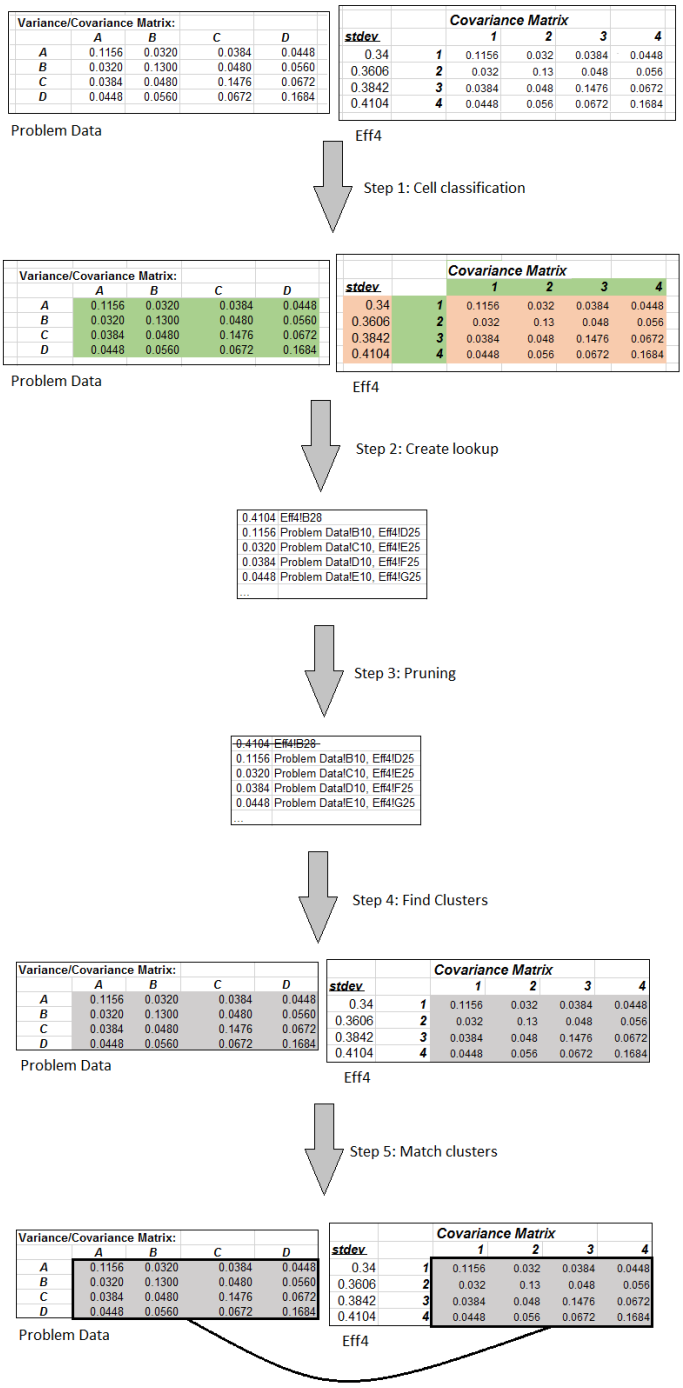


Figure 6.2: *Overview of the approach consisting of the five different steps: cell classification, lookup creation, pruning, cluster finding and cluster matching.*

6.5.2 Parameters

The algorithm takes four parameters as input:
StepSize is used in the fourth step of the algorithm and indicates the search radius in terms of numbers of cells. Setting it to 1 means we are only looking for direct neighbors, with step size 2, a 'gap' of 1 cells is allowed in a cluster.
MatchPercentage is used in the final step, when clusters are matched. This percentage indicates what percentage of the cells has to match. Setting it to 100% means the values have to match exactly, lower percentages allow for the detection of near-miss clones.
MinimalClusterSize sets the minimal number of cells that a cluster has to consist of. Very small clusters might not be very interesting, hence we allow for a minimal threshold.
MinimalDifferentValues represents the minimal number of different values that have to occur in a clone cluster. Similar to small clusters, those clusters consisting of a few different values will be of less interest.
Furthermore, a user of the algorithm can indicate whether clones are found *within* worksheets, *between* worksheets, between spreadsheets or a combination of those.

6.6 Clone visualization

We visualize the detected clones in two ways. Firstly, we generate a dataflow diagram that shows the relationship between worksheets that contains clones. Secondly, we add a pop-up to both parts of a clone indicating the source and the copied side of a clone.

The two visualizations serve a different purpose. The dataflow diagram is useful for *understanding* the relationship between worksheets and show how data is copied between worksheets in a spreadsheet or between multiple spreadsheets.

The pop-ups within the spreadsheet, on the other hand, are useful when maintaining a spreadsheet. Whether it is updating the copied side of a clone or refactoring the copy into a link, the pop-ups support the spreadsheet user in selecting the right cells.

6.6.1 Dataflow diagrams

In previous work[Her11; Her12b] we have developed a tool to generate a dataflow diagram from a spreadsheet to represent dependencies between worksheets. In this visualization, worksheets are visualized as rectangles, while arrows are used to indicate a formula dependency between two worksheets.

We consider the copying of data from one worksheet to another as a dependency between worksheets and hence we decided to show this dependency in our original dataflow diagrams too. We show data clone dependencies with a dashed arrow to show the difference with formula dependencies which are shown with solid arrows.

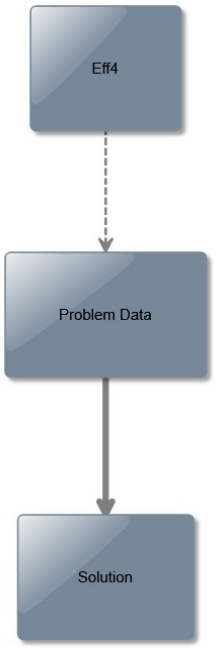


Figure 6.3: *Screenshot of the clone detection dataflow diagram corresponding to our running example hw4a.xls*

Figure 6.3 shows the dataflow diagram corresponding to the spreadsheet hw4a.xls shown in Figure 6.1. In this spreadsheet data is copied from formulas in the worksheet *Eff4* to the worksheet *Problem Data*, this is shown in the diagram with a dashed arrow.

6.6.2 Pop-ups

As described above, we add pop-ups to the spreadsheets to both the source and the copied side of the clone. These pop-ups, that are shown when the user clicks a cell, warn the user that data has been copied, so he can update the copy in case of a change to the formulas. Furthermore provides an easy way for the user to improve the design of the spreadsheet, by changing the copy-paste relationship into a link. By creating a link, the dependencies are made explicit and future changes will automatically be propagated.

Figure 6.4 shows an example of a pop-up indicating a detected clone cluster. On the formula side, we show where the data is copied and on the data side, we indicate the source.

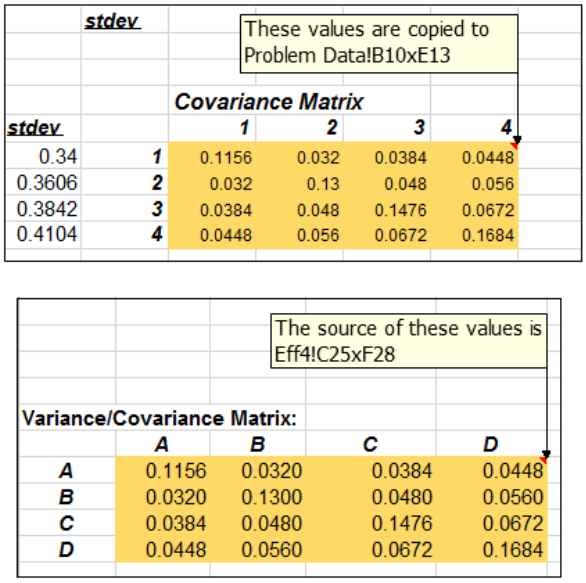


Figure 6.4: *Screenshot of the clone detection pop-up showing the copy-paste dependency for our running example hw4a.xls*

6.7 Implementation

Our current approach for the detection of data clones in spreadsheets is implemented into our existing spreadsheet analysis tool Breviz[Her11; Her12b]. Breviz is implemented in C# 4.0 using Visual Studio 2010. It utilizes the Gembox component to read Excel files.[3] Breviz reads an Excel file and executes the above described clone detection algorithm, either within a spreadsheet or among multiple files, and subsequently generates the dataflow diagram and a copy of the spreadsheet with pop-ups.

Breviz, including the data clone analysis, is available as-a-service, by uploading a spreadsheet to Infotron's website.[4]

6.8 Evaluation overview

To evaluate our approach, we performed both a quantitative and a qualitative analysis. First, we analyzed a subset of the EUSES corpus[Fis05a] to determine how well our algorithm performs and to learn how often data clones exist in this corpus. The corpus consists of more than 4000 spreadsheets from 11 different domains collected from practice.

Secondly, we studied two different real-life cases. The first case study was conducted at the South-Dutch foodbank, where employees keep track of all logistics using spreadsheets. For the second case study we evaluated a spreadsheet used by our university to calculate the budget for a large ($>$ 25 Million Euro) research proposal. With the qualitative analyses we aim to determine whether detected data clones actually pose a threat to spreadsheet quality.

6.9 Quantitative evaluation

6.9.1 Goal

The aim of the first evaluation is to answer research question 1 "How often do data clones occur in spreadsheets?" and to preliminarily evaluate the performance of our algorithm both in terms of execution time as in terms of the precision of the algorithm.

6.9.2 Background

In this evaluation, we used spreadsheets from the EUSES corpus[Fis05a]. This corpus contains real-life spreadsheets from 11 different domains, ranging from educational to financial, and from inventory to biology. It was created in 2005 and has since then been used by several researchers to evaluate spreadsheet algorithms,

[3] http://www.gemboxsoftware.com/spreadsheet/overview
[4] http://app.infotron.nl

among which [Abr06] and [Cun09a]. Of the 4223 spreadsheets in the corpus, 1711 spreadsheets contain formulas.

6.9.3 Setup

To reach our goal, we ran our data clone detection algorithm on those 1711 spreadsheets, for different values of the MinimalClusterSize and MinimalDifferentValues parameter.

Since we do not have a validated benchmark, we focus on matching exact clones. Evaluating the correctness of near-miss clones without the owners of the spreadsheets would leave too much room for speculation. Hence we set MatchPercentage to 100% for the quantitative study. In the qualitative studies, we will take near-miss clones into consideration. Furthermore, we do not search for clones between the files of the EUSES corpus. Since the spreadsheets are unrelated, matches between spreadsheets would always be false positives.

For each detected clone, we manually determine whether this is a real clone or a false positive. We do this by inspecting clones and determining whether 1) the detected clone clusters indeed share the same data values, 2) one of the detected clone clusters consists of formulas and the other of constant cells and 3) headers of the found clones match to decide whether the clones indeed concern the same conceptual data. This way we calculate the precision of our approach. We calculate this precision as the percentage of spreadsheets in which we verified a clone, divided by the total number of spreadsheets in which a clone is detected by the algorithm, rather than measuring it as the number of verified clones divided by the number of detected clones. We do this because we found that some spreadsheets contain as many as 25 clones, all of which are very similar and this could skew the results.

Since we do not know what spreadsheets in the corpus contain clones, we cannot not analyze the recall of our algorithm at this point. It would be both time-consuming and speculative to inspect all 4000+ spreadsheets in the corpus by hand to check whether they contain data clones. We plan to analyze recall in a future study on a smaller set of spreadsheets of which we can contact the creators.

The results of our analysis are all available online in the FigShare corpus[5], to enable other researchers to replicate our results.

6.9.4 Findings

[5]http://figshare.com/authors/FelienneHermans/98650

	Minimal Different Values										
Minimal Size	3	4	5	6	7	8	9	10	11	12	13
5	**54.8%**	59.1%	63.7%	-	-	-	-	-	-	-	-
6	54.2%	59.2%	62.9%	70.1%	-	-	-	-	-	-	-
7	53.8%	59.1%	62.5%	69.5%	70.9%	-	-	-	-	-	-
8	56.1%	60.2%	63.6%	70.1%	71.6%	72.9%	-	-	-	-	-
9	56.6%	60.6%	64.3%	71.2%	72.9%	74.6%	**81.7%**	-	-	-	-
10	55.1%	58.6%	62.3%	69.7%	71.4%	73.3%	80%	79.2%	-	-	-
11	56.3%	57.7%	60.9%	68.3%	70.2%	71.4%	78.4%	77.6%	78.3%	-	-
12	56.6%	58.1%	60.6%	67.8%	69.6%	70.9%	78%	77.1%	77.8%	81%	-
13	56.9%	57.4%	61%	66.7%	69.2%	70.6%	76.6%	75.6%	76.7%	80%	80%

Table 6.1: *Results of the EUSES evaluation showing the precentage of spreadsheets containing a data clone*

	Minimal Different Values										
Minimal Size	3	4	5	6	7	8	9	10	11	12	13
5	**86**	81	72	-	-	-	-	-	-	-	-
6	77	74	66	61	-	-	-	-	-	-	-
7	70	68	60	57	56	-	-	-	-	-	-
8	64	62	56	54	53	51	-	-	-	-	-
9	60	57	54	52	51	50	**49**	-	-	-	-
10	54	51	48	46	45	44	44	42	-	-	-
11	49	45	42	41	40	40	40	38	36	-	-
12	47	43	40	40	39	39	39	37	35	34	-
13	41	39	36	36	36	36	36	34	33	32	32

Table 6.2: *The number of spreadsheets in EUSES containing a data clone, for varying values of MinimalDifferentValues and MinimalSize*

Precision

Using MinimalClusterSize 5 and MinimalDifferentValues 3, which we consider the lowest meaningful values, our algorithm detects 157 spreadsheet files in the EUSES corpus that contain clones. Manual inspection showed that of these detected files, 86 contain verified clones. This 86 is highlighted in Table 6.2. 86 files out of 157 detected files with clones leads to a precision of 54.8%, as highlighted in Table 6.1.

In this table, one can find the precision for different values of MinimalClusterSize and MinimalDifferentValues. Combinations where MinimalDifferentValues is bigger than MinimalClusterSize are not allowed, since there cannot be more different values in a clone cluster than cells.

As illustrated by Table 6.1, the precision rises for higher values of the two parameters, especially the parameter MinimalDifferentValues is of influence, as we suspected. Highest precision (81.7%) is obtained with both parameters set to 9, this value is also highlighted in Table 6.1. In that case we still detect 49 clone files, which amounts to 57% of all 87 spreadsheets that contain verified clones in the EUSES test set (highlighted in Table 6.2).

False positives

The biggest category of false positives is a group of data that happen to occur at multiple places in a spreadsheet. For instance in a spreadsheet used for grades, we detect several groups of the numbers 1 to 10. If some are calculations and others are input data, this is detected as a clone. Especially for low values of MinimalClusterSize and MinimalDifferentValues, both below 6, this occurs frequently, since chances that small groups of values are found in multiple places are higher. A second category of false positives is a clone that is actually a header: spreadsheet users use formulas to describe their data, such as a department code

or a year. If in one case they use a formula and in another case they use a constant value, this is detected as a clone. Another type of false positives are clones consisting of array formulas that have the same value as other formulas in the worksheet. Gembox, the third party library we use to read spreadsheets, is not able to recognize array formulas, so they are read as being values.

Performance

Running the clone detection algorithm over the 1711 spreadsheet files in the EUSES corpus which contain formulas total took 3 hours, 49 minutes and 14 seconds (13,754 seconds in total). This means analyzing one file takes an average of 8.1 seconds.

Clone occurrence

In total, there are 1711 spreadsheets in the EUSES corpus that contain formulas, which means around 5% of all spreadsheets with formulas contain verified clones. Although not directly comparable, papers on clone analysis on source code estimate that 10 to 30% of lines of code are duplicated. For instance, Baker[Bak95] reported that around 13% - 20% of large code bases can be clones. Lague *et al.*[Lag97] found that, when considering function clones only, between 6.4% - 7.5% of code is cloned. Baxter et al.[Bax98] have reported 12.7% cloning and Mayrand et al.[May96] have estimated that industrial source code contains between 5% and 20% duplication.

Observations

While we cannot yet conclude something about the impact of data clones on spreadsheet quality, we noted several interesting similarities in this evaluation.

Firstly, we saw that a common pattern for cloning is the use in overviews and reports. In this scenario, spreadsheet users use one worksheet to calculate values, and copy them to a second worksheet to create a summary, a graph or a report sheet. Since many of these spreadsheets did contain links between worksheets, we do not think this use is due to the fact that the user did not know how to create links.

Secondly, we saw that copies are used to sort. In this scenario, a whole column is copied, sometimes directly next to the column with values, but the copy is sorted. This use might be due the way sorting in combination with links is implemented in Excel. When one sorts a column that has links, the formulas get sorted too. Users might prefer to make a copy to keep their formulas intact.

Finally, an unexpected observation was that in some cases the format of the cells that were clones did not match. For instance, the original formulas were typed currency, while the copied cells were typed as a percentage. Even without knowing the context of this spreadsheet, we can conclude that one of the cell formats must be wrong. This practice is error-prone, especially in the case of

dates. When a date is typed as a number, Excel will show a the number of days this day is removed from January 1, 1900, since Excel uses the 1900 date system. This way a user can easily overlook the fact that this value represents a date. In future work we plan to work on the detection of these mismatching clones.

6.10 The Case studies

After we performed the quantitative evaluation and we were convinced of both the applicability of our approach and the frequency with which clones occur in practice, we conducted two case studies to investigate the implications of data clones in spreadsheets.

6.10.1 Goal

The goal of the two case studies is to answer research questions 2 and 3: to learn more about the impact of data clones and to evaluate our data clone detection and visualization approach.

6.10.2 Setup

To reach this goal, we have analyzed real-life spreadsheets in both case studies: we ran the data clone detection algorithm and subsequently we presented the results to the spreadsheet owners. Next, we went over all detected clones with them and asked them the following questions:

- Is this a real clone, in other words: did you copy this data?
- Did this clone lead to errors or problems?
- Could this clone be replaced by a formula link?

Furthermore, we asked them the following questions about clones and about our approach:

- Do you know why no direct links were used initially?
- How did the pop-ups help you in understanding the found data clones?
- How did the dataflow diagrams help you in understanding the found data clones?

6.10.3 Background

The following describes the background of the two case studies.

Foodbank A foodbank is a non-profit organization that distributes food to those who have difficulty purchasing it. We ran our case study at the distribution center of the foodbank that supplies 27 local foodbanks. In 2011 the distrubution centre processed an average of 130.000 kilograms of food per month. To keep track of this process, they use a set of spreadsheets. The figures of incoming food from sponsor and food sent out fo local foodbanks should balance, since no food is supposed to remain in the distribution center.

In January 2012, the distribution center of the foodbank approached us and asked whether we could help them improve their spreadsheet administration, since they observed that the total result did not balance and food remained in the center, or went missing.

Initially, we did not know what caused their problems, but when we learned about the copy-pasting practice that was used, we suspected that clone detection might help to locate the errors. We asked the foodbank whether they would be interested in participating in a study of a new feature we were developing, with which they agreed.

Subsequently, we received 31 spreadsheet files from the foodbank, to check whether clones might be the source of problems. One of those spreadsheets was the distribution list, while the other 30 were lists of a specific region.

Delft University In April of 2012 Delft University of Technology participated in a grant proposal, for which a budget spreadsheet had to be created. This particular spreadsheet calculates, among other things, the salary cost of different types of employees. These salaries are raised every year, because of inflation, and the creator of this spreadsheet calculated the salaries once and copied them to different places in the spreadsheet.

The author involved in this proposal noticed this duplication and asked this employee whether he would want to participate in a study on cloning in spreadsheets and this employee agreed.

6.10.4 Findings

In this subsection we describe the results of both case studies.

Foodbank In the first study, we searched for data clones in the test set of the foodbank by running the prototype tool over the 31 spreadsheets. We used parameters 9 for MinimalClusterSize and MinimalDifferentValues, since they were the optimal choice in the quantitative analysis. Furthermore we set Matching-Percentage to 80% and StepSize to 2 to enable the detection of near-miss clones. Running the algorithm took 3 hours, 9 minutes and 39 seconds, which amounts to 6 minutes per file. Performance was worse than in the EUSES case, since in this analysis, all clones of all files had to be compared with each other, since we were searching for clones between files too.

With these settings, we detected 145 clones, of which 61 were near-miss clones, in other words, they had a matching percentage of less than 100%. Furthermore, in this case we only searched for clones between spreadsheets, since we knew that there would only be clones between files. We discussed the found clones one by one with three employees of the foodbank and checked whether the found clones were actual clones. Only one of the found clones was identified as a false positive: in that case, by coincidence two files contained similar values.

Subsequently, we studied the near-miss clones in more detail: were they really errors that had been made in copy-pasting? We found that all cases were 'wrong' in the sense that the values should match. The employee that we discussed the results with stated "these should always match, I don't understand why they do not." However, in many cases, the updates made to the copies were the right values, but in 25 of the detected 61 near-miss clones were actual errors that the employees were not aware of before our analysis. While checking the near-miss clones, we also found that one of the exact clones was actually an error: here the data had been copied into the wrong column. The foodbank employees stated that all found clones could, in theory, be replaced by direct links. No direct links were used initially, since the employee who created the spreadsheets, was not very proficient in spreadsheets at that time. She started with a few spreadsheets and copying the values would not be much of a problem. When the collection of spreadsheets got bigger, it became increasingly more difficult to make the change.

Later on, another employee was put in charge of handling the spreadsheets. She stated: "Since I did not build all the sheets, I am always a bit afraid to adapt formulas. Since I can see the links in the pop-ups that you created, I feel more safe, since I know it will do the right things."

This sentiment is shared even by the original creator of the spreadsheets, saying "The problem with managing multiple sheets is that you never know if changing one cell will mess up other sheets or files." Especially for the current maintainer of the spreadsheets, seeing files that were not linked was insightful. "I assumed this region was already copied into the total sheet, but in the diagram I see it is not. I should fix that right away."

After the employees fixed the clones that we found, the overall results balanced as they should, which we considered a very good result and strong evidence that data clones can indeed be the cause of errors.

Delft University In the case study for the Delft University, we studied the budget spreadsheet, consisting of 15 worksheets. We again used MinimalCluster-Size 9 and MinimalDifferentValues 9 and set the MatchingPercentage to 80% and StepSize to 2. In this case study, we searched for clones between worksheets, since there was just one spreadsheet. Running the algorithm took 3 seconds.

We found 8 exact clones, which all turned out to be real clones, i.e. they were copied by the spreadsheet creator. When we asked him why he used the clones instead of links, he stated that the spreadsheet was a draft version and that it seemed easier to simply copy the values. Although these clones did not lead to problems in this case, the spreadsheet owner did state that if he had

to share the spreadsheet with others, he thought he should replace the clones with links. He stated that our analysis would be very useful in improving the spreadsheet and removing the clones: "This scan is very useful. You can just copy-paste and the system indicates where you should make links". In this case study, the visualization was interesting, since there were two worksheets that were both connected by a solid arrow, indicating formula dependencies, and a dashed arrow which shows a clone dependency. We consider this as extra risky, because the spreadsheet's user might think all values are linked upon seeing one of the formula dependencies.

6.11 The research questions revisited

In this section, we revisit the research questions.

R1: How often do data clones occur in spreadsheets? We learned from the both EUSES case and the case studies that clones occur often in spreadsheets. Around 5% of all spreadsheets in the EUSES corpus contain clones.

R2: What is the impact of data clones on spreadsheet quality? From the two case studies, we learned that clones can have two different types of risks. We learned that clones matching 100% mainly impact the users perspective of spreadsheets ("I did not know these values were copied from that source"), while near-miss clones really causes trouble ("this value should have been updated months ago").

R3: Does our approach to detect and visualize data clones in spreadsheets support users in finding and understanding data clones? In both studies we saw that employees were not always aware of what values were copied from what sheets to what others. Even creators of the spreadsheets did not know all relations by heart. The dataflow visualizations aided users in quickly getting an overview of the, otherwise very hidden, copy dependencies ("the system indicates where you should make links").

6.12 Discussion

Our current approach to finding clones helps spreadsheet users to understand how data is copied throughout worksheets and spreadsheets and furthermore supports them in improving erroneous relations. In this section, we discuss a variety of issues that affect the applicability and suitability of the proposed approach.

6.12.1 Relative settings for parameters

In the current evaluations, we have used fixed settings for the parameters: we set MinimalDifferentValues and MinimalClusterSize both to 9, irrespective of the spreadsheet. However, it could improve performance of the algorithm if these parameters were calibrated using properties of the spreadsheets. For instance, in

a spreadsheet with only 8 rows, no clones will ever be found. Although the evaluations showed that using fixed settings leads to useful results, taking spreadsheet properties into account might improve the algorithm even further.

6.12.2 Headers

In previous work, we have worked on the extraction of metadata from spreadsheets[Her10]. Other authors have worked on this as well[Abr04; Erw02]. We could use extracted header information, such as column or row names to gain more confidence in detected clones. If clones are found below column headers with the same name, chances are bigger that clones are are 'real' clones and concern the same conceptual data.

6.12.3 Copied data

In addition to copying the results of formulas, a spreadsheet user can also copy data to different places of the spreadsheet. This is a different type of cloning in spreadsheets that we have not yet considered in the current tool. We hypothesize that copy-pasting of data might also be error-prone, however this calls for more research. Furthermore, there is be the challenge of detecting the origin of the clone, similar to origin analysis in source code[God02; Zou03]. We see this as an interesting avenue for future research.

6.12.4 Clone genealogy

The current approach analyzes cloning as it occurs in a spreadsheet at a given point in time. However, it would also be very interesting to understand how clones are created and adapted. When spreadsheets are be placed under version control, such as provided by Microsoft's Sharepoint, for example, it will be possible to also track the history of a set of clones, similar to clone genealogy work in source code analysis[Kim05; Bak11].

6.12.5 Spreadsheet maintenance support

We learned from the case study at the foodbank that, when spreadsheets become larger and more complex, their users become more anxious to make structural changes. Although this does not relate to cloning directly, updating a clone into a formula link is one of those changes that users fear might mess up the entire sheet. This means that spreadsheets need better support for previewing a change, such as some refactoring tools offer, or the option to calculate what cells will be affected by a certain change.

6.12.6 Threats to validity

A treat to the internal validity of our quantitative evaluation is the fact that we validated the precision of the algorithm manually. We have however described our approach and made our results publicly available, so our results can be replicated. A threat to the external validity of our quantitative evaluation concerns the representativeness of the EUSES corpus. However, the EUSES corpus is a large set that is collected from practice and has been used for numerous spreadsheet papers. The external validity of the qualitative evaluation might suffer from this threat, however these spreadsheets too are collected from practice and available online to enable other researchers to replicate our results.

6.13 Concluding remarks

The goal of this chapter is to investigate the risks that data clones pose to spreadsheets. To that end we have defined data clones and described a way to detect and visualize them. We have subsequently evaluated data clones in two ways, with a quantitative evaluation on the EUSES corpus and two real-life case studies in which we found that data clones are common and can lead to real errors.

The key contributions of this chapter are as follows:

- The definition of data clones in spreadsheets (Section 6.4).
- An approach for the automatic detection (Section 6.5) and visualization (Section 6.6).
- An implementation of that approach into our existing spreadsheet analysis toolkit Breviz (Section 6.7).
- A quantitative evaluation of the proposed clone detection algorithm on the EUSES corpus (Section 6.10).
- A real-life evaluation with 31 spreadsheet from a Dutch non-profit organization and 1 from academia (Section 6.9).

The results of our evaluations show that around 5% of spreadsheets contain data clones and that these clones lead to actual errors such as in the case of the foodbank. The current research gives rise to several directions for future work. Firstly , the algorithm could be improved to get a better precision. Also, in a new study, we will analyze the recall of the algorithm and on detecting clone that do not match in their number format, since these might be even more error-prone than the data clones we detect currently. Furthermore, the case study learned us that impact analysis of changes in spreadsheets could help to increase a spreadsheet user's confidence, either before the change or directly after. This is a very interesting avenue for further research. Finally, taking the metadata into account might strengthen the clone detection algorithm.

Chapter 7

Conclusion

7.1 Contributions

The aim of this research is to analyze and visualize the *design* of a spreadsheet. To that end we have looked at different aspects of a spreadsheet: the metadata, the formulas, the organization and finally the data, in order to obtain a complete view.

The contributions of this dissertation are:

- A pattern language to express common spreadsheet patterns (Chapter 2)
- A method to automatically recognize these patterns and transform them into class diagrams (Chapter 2)
- A detailed survey analyzing the information needs of spreadsheet users (Chapter 3)
- A method to extract dataflow diagrams form spreadsheets (Chapter 3)
- The definition of four inter-worksheet smells, based on known code smells (Chapter 4)
- An approach for the automatic detection of the inter-worksheet smells (Chapter 4)
- An analysis of the risky types of spreadsheet formulas (Chapter 5)
- An approach for the automatic detection of the formula smells (Chapter 5)
- The definition of data clones in spreadsheets (Chapter 6)

- An approach for the automatic detection and visualization of clones (Chapter 6)
- Detailed evaluations of all the above, where possible performed in an industrial setting
- A tool called Breviz in which all extraction methods are integrated

7.2 The different aspects of spreadsheets

In this section we present an overview of the different aspects we described in the Introduction—data, metadata, formulas and organization—and to what extent our approach is able to analyze them.

7.2.1 Metadata

The first attempt we made at obtaining information from spreadsheets concerned the extraction of metadata. We did this by creating a two-dimensional pattern language and describing well-known spreadsheet patterns in this language, such that they could be recognized and transformed into class diagrams.

As shown in Chapter 2, our proposed method is able to extract perfect class diagrams from 40% of the tested spreadsheets. 26% contained minor flaws such as missing fields, while 22% of the diagrams were incomplete. Only 12% of extracted diagrams were not meaningful.

From these results, although limited in number, we conclude that it is possible to automatically extract metadata from spreadsheets. Interaction with a spreadsheet user, such as allowing the user to select metadata or to validate extracted diagrams could improve precision of this approach.

However, one of the aims of this work was to use the extracted class diagrams as a basis for reimplementing the underlying spreadsheet. Although we found that this is technically possible, when testing this idea with users, we found that migrating spreadsheets was not often feasible, for several reasons:

- Spreadsheet users love the flexibility of their spreadsheet and are not easily convinced by benefits of a tailormade application, such as speed or safety.
- IT departments in large companies do not have the manpower, nor the interest to rebuild spreadsheets, which they deem often as as uninteresting end-user artifacts, despite their documented impact in a company.
- To actually conduct a migration, additional information, beyond a class diagram representation, is needed.

This led us to pursue the idea of understanding and refining spreadsheets, rather than aiming to rebuild them.

7.2.2 Organization

We thus directed our attention to the organization of a spreadsheet, since we concluded—after performing a grounded study with 27 spreadsheet users—that the way that spreadsheet users divide their data into tables and worksheets was an important factor in understanding a spreadsheet.

As shown in Chapter 3, subjects in our qualitative study expressed that the global view, showing the worksheets and their relationships, helped them in seeing the *idea* behind the spreadsheet. "The global view reminds me of the plan I had when building this spreadsheet. Normally, I just walk through the worksheets from left to right, but that is not the logical way."

When we observed spreadsheet users interacting with these visualizations, we found that they used them, not only for understanding spreadsheets, but also for assessing their quality. This was a very interesting phenomenon that we did not anticipate when we created the dataflow diagrams. We noticed that especially the global view was used to this purpose. This is why we formalized this formerly informal assessment of spreadsheet quality by introducing *inter-worksheet smells*. Since we asserted that worksheets and their connections strongly resemble classes and their relations, we decided to use inter-class code smells as our starting point. Chapter 4 shows that annotating the dataflow diagrams helps users to judge spreadsheet quality even more. One of the subjects stated, upon being confronted with the annotated dataflow diagram: "I should really take some time to improve these formulas, since this is already confusing for me, so it must be terrible for others".

Although there are still limitations in our approach, such as analyzing pivot tables and VBA code, the experiments have shown that dataflow visualizations are a suitable way to help users in understanding the high-level design of their spreadsheets, while automatic smell detection helps them to see weaknesses in this design.

7.2.3 Formulas

While researching the topic of smells, we noticed that users also felt the need to judge individual formulas. This is why we expanded our idea of spreadsheet smells into the realm of spreadsheet formulas. Again we took our inspiration from existing code smells, yet this time at the intra-class level. With this approach, we were able to locate complex formulas in a spreadsheet. Users in the subsequent case study confirmed that the selected formulas were indeed the least maintainable and could be improved: "I understand why this formula is selected by your system, it is quite long." Another subject, when looking at a formula that referred to no less than 17 ranges said "this formula is a real puzzle". The study showed that our smells reveal formulas that are too complex. These formulas will be time consuming to change, and changes made will be more error prone. We did not only find formulas that users deemed tricky, our method found two actual faults

in a spreadsheet by looking at the Duplication Smell.

From the results of this study we can conclude that applying code smells to spreadsheet formulas is a good way to detect flaws in a spreadsheet's design, at the detailed level of formulas.

7.2.4 Data

The final aspect of spreadsheets we analyzed was the data in the spreadsheet. We noticed that spreadsheet users resorted to copy-pasting their data (rather than using the more robust creation of formula links). They do so for a variety of reasons:

- Some users simply do not know how to create formula links, especially links between files can be tricky to get right.

- Users want a spreadsheet to be self-contained, for instance because it can then be emailed more easily, since the receiver does not need access to the source data. Since spreadsheets are emailed around very often, this is a scenario that is quite likely.

- In some cases, a spreadsheet owner wants to continue to develop a spreadsheet, while also allowing others to use the data. For instance, a user might want to share the preliminary results so colleagues can do analysis on them, while still having the freedom to update these results.

These reasons result in the fact that many spreadsheets contain copy-pasted data. This can diminish understandability, since the real source of data is concealed and can furthermore be error-prone, since in the case of an update, this update has to be performed on all copies. This is why we decided to add automatically detection of data clones to our analysis toolkit.

When validating our clone detection approach, we found that both exact clones (those matching 100%) and near-miss clones occur in spreadsheets. Both pose different threats. Exact clones mainly impact the user's perspective of spreadsheets ("I did not know these values were copied from that source"), while near-miss clones really cause trouble ("this value should have been updated months ago"). We learned that cloning can be a source of errors, as even creators of the spreadsheets did not know all relations by heart. In one of the case studies, our approach was able to detect and repair severe errors in spreadsheets used for the inventory of the south-Dutch foodbank. This has shown that clone detection in spreadsheets can solve real-life spreadsheet problems.

7.3 Reflecting on methodology

7.3.1 Methods used in this dissertation

In this research we have tried to emphasize doing research in a real context. We have done so firstly, by making use of the well-known EUSES Corpus, for the in evaluations in Chapters 2, 4 and 5.

Secondly, we have employed *Grounded theory*. Grounded theory is a qualitative research method that aims at obtaining a theory from data, especially observations. As such, it is highly suitable to discover problems that exist for participants. In a grounded theory study, the researcher works with a general area of interest rather than with a specific problem. Through the use of grounded theory, we aimed at finding problems that are really relevant to industrial spreadsheet users.

Finally, we have performed industrial case studies to validate our different approaches, as described in Chapters 3, 4, 5 and 6. We choose this approach to deeply understand how our solutions could be useful for real spreadsheets and real spreadsheet users.

Where possible, we have used a mixed method approach to evaluate our approaches. In particular, we have used the explanatory sequential study. When using this type of mixed method, one first performs a quantitative evaluation to get an initial insight into a problem, followed by a qualitative study in which these insights are deepened with users.

7.3.2 Impact

While performing the research, we have experienced that starting with a grounded theory approach is an excellent way to make a connection with real problems, especially in an applied field like software engineering. By connecting with people and taking an interest in their professional interests and problems, we were able to develop methods that meet their needs. We believe that the use of grounded theory has positively contributed to the adoption of our approaches in the company that we performed our studies at. Both at Robeco and at the Dutch Food Bank, our software is still used.

With respect to the mixed method, we also conclude that this method is very applicable to software engineering studies. In many cases it is possible to both analyze the data and interview users or perform a case study. By combining the methods, threats to validity from performing only one study are mitigated. For instance: just analyzing the data might not reveal the true problems and has the risk that the observer has to make assumptions when interpreting the results, while only performing a case study might lead to generalization that are not grounded. In our opinion, the combination is empirically strong and furthermore makes for nice and readable papers, since heavy statistics can be interleaved with quotes from the case studies.

7.3.3 Risks

Of course, doing research so closely together with industry also has its risks. It is possible companies view the researcher as a professional software engineer and demand deliverables such as working software, as opposed to research prototypes. If the company feels that the researcher is not producing anything useful, this could negatively impact the research project.

Another threat is the possibility that the topic under research is only relevant to the one company where the case study is performed. We have mitigated this risk by communicating early and often about our work, to stay convinced that the studies we were doing at Robeco also mattered to other people in other companies.

Furthermore, when performing multiple case studies with the same organization, there is a risk of aptitude treatment interaction. We certainly had some 'fans' at Robeco, who were frequent users of the tool and felt very positive about the research, making them less objective test subjects. In all tests, however, we got enough criticism to assert that even very positive employees were still able to point at weaknesses in the approach. Still, this phenomenon is something we must certainly keep taking into account in future research.

7.4 Revisiting the research questions

Given the ideas and results described in the previous chapters, we now direct our attention to the main research question of this dissertation.

RQ1 To what extent are methods and techniques from software engineering applicable to the spreadsheet domain?

In this dissertation, we have taken methods from software engineering applied them to spreadsheets. In particular, we have investigated class diagrams (Chapter 2), data flow diagrams (Chapter 3), code smells (Chapter 4 and 5) and clone detection (Chapter 6).

One of the core reasons that we were able to apply methods from software engineering to spreadsheets successfully was the fact that we were able to apply these concepts to spreadsheets without additional efforts on either the spreadsheets, such as preparing them for analysis, or their users, such as training them. This resulted in very low overhead for the spreadsheet users: they did not have to change their way of working with spreadsheets, they could just work as they were used to, while getting additional information when needed.

Furthermore, the methods and techniques we have applied, kept their original value. The extracted dataflow diagrams, for example, proved to have the same value to spreadsheet users as extracted dependency diagrams have to software engineers: they quickly show how a system is structured and provide assistance when learning the underlying code. With spreadsheet smells and the clone detection we were able to find weak points and even errors in the spreadsheets, similar to code smells and code clones.

This gives credibility to the hypothesis that other methods from software engineering too can be applied to spreadsheets. Methods that spring to mind are testing, version control or invariants. However, for these methods, spreadsheet users will have to put in extra effort, such as adding tests, commit messages and assertions, which makes it harder to get spreadsheet users to adopt these methods. It remains an open question how to transfer these software engineering principles to spreadsheets in a non-intrusive manner.

RQ2 How can we visualize the underlying design of a spreadsheet in order to support spreadsheet understanding?

To answer this research question, we have explored how class diagrams and data flow diagrams can be used to support spreadsheet understanding, as described in Chapter 2 and 3.

The visualizations were useful for spreadsheet users, again since they did not have to change their normal way of working, while being able to get design information in a diagram. Since the diagrams can be automatically extracted, they are available almost immediately when support for understanding is necessary.

Our evaluations have shown that the resulting diagrams can be used for understanding a spreadsheet's design, to support re-implementing it and to ease the transfer of a spreadsheet from one employee to another. In the current implementation, however, some design elements are not covered: such as pivot tables, VBA code and array formulas.

RQ3 To what extent does the visualization and analysis of the design of a spreadsheet support users in assessing quality of their spreadsheet?

For the final research question, we have aimed at measuring quality, based on code smells and clone detection to spreadsheets. We have looked both at smells between worksheets, based on inter-class smells (Chapter 4), smells within worksheets, based on intra-class smells (Chapter 5). In both cases we have used metrics to detect the smells, which were calibrated by setting thresholds based on the EUSES Corpus [Fis05b]. Furthermore, we have applied clone detection to capture common copy-paste errors and vulnerabilities (Chapter 6).

With the above approached, we have been able to detect serious maintainability problems and even actual errors in industrial spreadsheets.

However, the approach has its limitations. We mainly focus on maintainability and not on other important aspects, like correctness or performance. These factors are correlated less with the design of a spreadsheet and hence our method is less suitable for detecting these kinds of problems.

Following from all the above, the overall conclusion of this dissertation are: **(i) methods from software engineering form an excellent source of inspiration for analysis and (ii) visualization algorithms for spreadsheets and spreadsheet users are supported by the analysis and visualization of their spreadsheet's design.**

7.5 Future work

The current research gives rise to several avenues for future work.

Beyond formulas and data

The current approach focuses on the visualization and analysis of spreadsheet formulas and data. However, there are several other constructs in spreadsheets we do not yet analyze, like VBA code, pivot tables and graphs. An interesting direction here would be to research *how formulas and data are combined with more complex spreadsheet constructs.*

Some spreadsheet models, for instance, use both VBA code and formulas, making it hard for spreadsheet users to understand how this VBA code fits into the design of the spreadsheet. Future work that would research the possibility to visualize the way in which VBA code and formulas are combined would be of great interest for more complex spreadsheets containing VBA. The challenge would of course be technical—parsing the VBA code and detecting dependencies between worksheets in the code—but even more user centered: how to visualize the mutual dependencies between formulas and other constructs in an automatic and easy comprehensible way.

In all these concepts design smells could also be defined and detected, creating another viable direction for future research.

Spreadsheet evolution

In the case studies we have seen that spreadsheets are not simply created and then used immediately. Similar to software, they evolve and stay alive for several years, 5 years on average as shown in Chapter 3. This gives rise to the question *how spreadsheets evolve over time.* By analyzing a large number of different versions of one spreadsheet, we could shed more light on how spreadsheets are created and maintained in general. For instance, by creating a data flow diagram of each version and subsequently combining them into an animation, we could visualize the history of this spreadsheet. Especially studying how smells and clones appear and disappear over time could give us valuable insight on how spreadsheet errors are created.

Furthermore analyzing previous versions of one spreadsheet could support the spreadsheet's users in understanding the current version of a spreadsheet better. For instance, knowing who created a formula and what other cells where edited at the same time, might help the user to understand the context of this formula.

Change impact analysis

In the final two experiments, we have seen that spreadsheet users, while updating their spreadsheet, have trouble predicting how their spreadsheet will behave after the change. This is why *helping users understand how their spreadsheet will behave*

after a change is an interesting direction for future work. Here the focus will be to help users to prevent errors in the future and not to analyze the current state of the spreadsheet.

As one of the subjects in the smells study stated: "if I move this, what will happen to the other formulas? I would like to preview that". In the clone study a different subject said: "The problem with managing multiple sheets is that you never know if changing one cell will mess up other sheets or files."

One way to overcome this is the introduction of spreadsheet tests, so the user will have confidence that despite the change, the spreadsheet still works as it is supposed to. This however would require additional effort from the spreadsheet user.

A second option that we believe to be less demanding for the spreadsheet users, is the calculation of the impact of the change. We could show the users exactly what cells have been impacted by the update, or even predict what cells will be impacted before a change has even been performed, so they can check only those cells for correctness. This direction again would have a technical component, but would also present challenges on how to communicate the impacted cells to the spreadsheet users in a clear and simple way.

Appendix A

Breviz

The extraction methods we have described in the above chapters are combined into one web application called Breviz. Currently, the dataflow extraction as described in Chapter 3, the smell detection from Chapters 4 and 5 and the clone detection (Chapter 6) are incorporated into Breviz. Only the class diagram extraction as described in Chapter 2 is currently not part of Breviz, as that visualization is mainly aimed at programmers, whereas Breviz is a real end-user tool.

Breviz is available as a service on `app.infotron.nl`. A spreadsheet user can simply upload his spreadsheet and a report will be generated in the browser. By means of the report, the user gets an overview of the extracted information.

The report is divided into different categories. Firstly, the user gets a list of formulas that are marked *smelly* by Breviz. In the case where a refactoring is obvious, such as the *Feature Envy* smell, where the proposed refactoring is to move the formula, Breviz also lists this refactoring, as shown in Figure A.1.

Figure A.1: *Breviz suggesting refactorings.*

Next, detected data clones are listed, as depicted in Figure A.2.

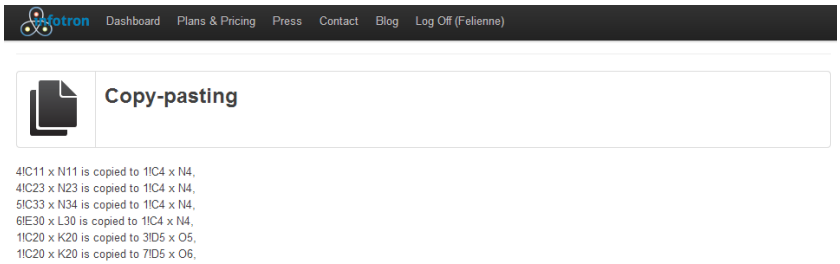


Figure A.2: *Breviz showing where data is cloned.*

Finally, the generated dataflow diagram is shown. Each rectangle represents a worksheets and an solid arrow represents formula dependencies between worksheets. Dashed arrows represent data has been copied between worksheets. This can be seen in Figure A.3.

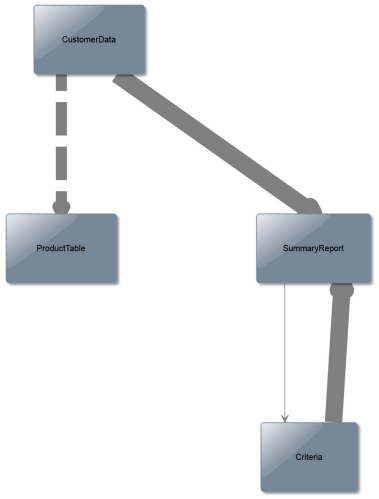


Figure A.3: *A dataflow diagram, including dashed arrows that represent copying.*

In addition to the report, users can download an annotated copy of his spreadsheet, in which smelly and cloned cells are colored and marked with a popup, forming the *risk map* as described in Chapter 5. Breviz is implemented in C# 4.0 using Visual Studio 2010. It utilizes the Gembox component to read Excel files. [1]

[1] http://www.gemboxsoftware.com/spreadsheet/overview

Bibliography

[Abr04] R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. In *Proc. of VL/HCC '04*, pp. 165–172. 2004.

[Abr05a] R. Abraham and M. Erwig. How to communicate unit error messages in spreadsheets. In *Proc of WEUSE '05*, pp. 1–5. 2005.

[Abr05b] R. Abraham, M. Erwig, S. Kollmansberger, and E. Seifert. Visual specifications of correct spreadsheets. In *Proc. of VL/HCC '05*, pp. 189–196. IEEE Computer Society, 2005.

[Abr06] R. Abraham and M. Erwig. Inferring templates from spreadsheets. In *Proc. of ICSE '06*, pp. 182–191. 2006.

[Abr07a] R. Abraham and M. Erwig. Ucheck: A spreadsheet type checker for end users. *Journal of Visual Languages and Computing*, vol. 18:pp. 71–95, 2007.

[Abr07b] R. Abraham, M. Erwig, and S. Andrew. A type system based on end-user vocabulary. In *Proc. of VL/HCC '07*, pp. 215–222. 2007.

[Abr09] R. Abraham and M. Erwig. Mutation operators for spreadsheets. *IEEE Transactions on Software Engineering*, vol. 35(1):pp. 94–108, 2009.

[Ado08] S. Adolph, W. Hall, and P. Kruchten. A methodological leg to stand on: Lessons learned using grounded theory to study software development. In *Proc. of CASCON '08*, pp. 166–178. 2008.

[Ahm03] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi. A type system for statically detecting spreadsheet errors. In *Proc. of ASE '03*, pp. 174–183. 2003.

[Aho86] A. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[Ala12] M. H. Alalfi, J. R. Cordy, T. R. Dean, M. Stephan, and A. Stevenson. Models are code too: Near-miss clone detection for simulink models. In *Proc. of ICSM '12*. 2012. To appear.

[Alv10] T. L. Alves, C. Ypma, and J. Visser. Deriving metric thresholds from benchmark data. In *Proc. of ICSM '10*, pp. 1–10. IEEE Computer Society, 2010.

[And04] W. Anderson. *A Comparison of Automated and Manual Spreadsheet Detection*. Master's thesis, Massey University, Albany, New Zealand, 2004.

[Aur10] S. Aurigemma and R. R. Panko. The detection of human spreadsheet errors by humans versus inspection (auditing) software. In *Proc. of EuSpRIG '96*. 2010.

[Aya00] Y. Ayalew, M. Clermont, and R. T. Mittermeir. Detecting errors in spreadsheets. In *Proc. of EuSpRIG '00*, pp. 51–62. 2000.

[Bad12] S. Badame and D. Dig. Refactoring meets spreadsheet formulas. In *Proc. of ICSM '12*. 2012. To appear.

[Bak78] T. P. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM Journal on Computing*, vol. 7(4):pp. 533–541, 1978.

[Bak95] B. S. Baker. On finding duplication and near-duplication in large software systems. In *Proc. of WCRE '95*, pp. 86–95. 1995.

[Bak11] T. Bakota. Tracking the evolution of code clones. In *Proc. of SOFSEM '11*, pp. 86–98. 2011.

[Bal87] D. P. Ballou, H. L. Pazer, S. Belardo, and B. D. Klein. Implication of data quality for spreadsheet analysis. *DATA BASE*, vol. 18(3):pp. 13–19, 1987.

[Bax98] I. D. Baxter, A. Yahin, L. M. de Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Proc. of ICSM '98*, pp. 368–377. 1998.

[Bec11] L. Beckwith, J. Cunha, J. P. Fernandes, and J. Saraiva. An empirical study on end-users productivity using model-based spreadsheets. In *Proc. of the EuSpRIG '11*, pp. 87–100. July 2011.

[Bel93] D. Bell and M. Parr. Spreadsheets: A research agenda. *SIGPLAN Notices*, vol. 28(9):pp. 26–28, 1993.

[Bir77] R. S. Bird. Two dimensional pattern matching. *Information Processing Letters*, vol. 6(5):pp. 168–170, 1977.

[Bra09] L. Bradley and K. McDaid. Using bayesian statistical methods to determine the level of error in large spreadsheets. In *Proc. of ICSE '09, Companion Volume*, pp. 351–354. 2009.

[Bre04] A. Bregar. Complexity metrics for spreadsheet models. In *Proc. of EuSpRIG '04*, p. 9. 2004.

[Bru05] M. Bruntink, A. van Deursen, R. van Engelen, and T. Tourwé. On the use of clone detection for identifying crosscutting concern code. *TSE*, vol. 31(10):pp. 804–818, 2005.

[Bur99] M. Burnett, A. Sheretov, and G. Rothermel. Scaling up a what you see is wat you test methodology to spreadsheet Grids. In *Proc. of VL '99*, pp. 30–37. 1999.

[Bur03] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace. End-user software engineering with assertions in the spreadsheet paradigm. In *Proc. of ICSE '03*, pp. 93–103. 2003.

[Cha96] M. Chatfield and R. Vangermeersch. *The History of Accounting-An International Encyclopedia*. Garland Publishing, New York, 1996.

[Cha09] C. Chambers and M. Erwig. Automatic detection of dimension errors in spreadsheets. *Journal of Visual Languages and Computing*, vol. 20:pp. 269–283, 2009.

[Che86] P. Cheney, R. I. Mann, and D. L. Amoroso. Organizational factors affecting the success of end-user computing. *Journal of Management Information Systems*, vol. 3:pp. 65–80, July 1986.

[Cle03] M. Clermont and R. Mittermeir. Analyzing large spreadsheet programs. In *Proc. of WCRE '03*, pp. 306–315. 2003.

[Cle04] M. Clermont. *A Scalable Approach to Spreadsheet Visualization*. Ph.D. thesis, Universitaet Klagenfurt, 2004.

[Con97] D. Conway and C. Ragsdale. Modeling optimization problems in the unstructured world of spreadsheets. *Omega*, vol. 25(3):pp. 313–322, 1997.

[Cre03] J. W. Creswell. *Research design : qualitative, quantitative, and mixed method approaches*. Sage Publications, 2nd edn., 2003.

[Cun09a] J. Cunha, J. Saraiva, and J. Visser. Discovery-based edit assistance for spreadsheets. In *Proc. of VL/HCC '09*, pp. 233–237. 2009.

[Cun09b] J. Cunha, J. Saraiva, and J. Visser. Discovery-based edit assistance for spreadsheets. In *Proc. of VL/HCC '09*, pp. 233–237. IEEE, 2009.

[Cun09c] J. Cunha, J. Saraiva, and J. Visser. From spreadsheets to relational databases and back. In *Proc. of PEPM '09*, pp. 179–188. 2009.

[Cun12] J. Cunha, J. P. Fernandes, J. Mendes, and J. S. Hugo Pacheco. Towards a catalog of spreadsheet smells. In *Proc. of ICCSA'12*. LNCS, 2012.

[Dav96] J. S. Davis. Tools for spreadsheet auditing. *International Journal of Human Computer Studies*, vol. 45(4):pp. 429–442, 1996.

[Duc99] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In *Proc. of ICSM '99*, pp. 109–118. 1999.

[Eng05] G. Engels and M. Erwig. Classsheets: automatic generation of spreadsheet applications from object-oriented specifications. In *Proc. of ASE '05*, pp. 124–133. 2005.

[Erw02] M. Erwig and M. M. Burnett. Adding apples and oranges. In *Proc. of PADL '02*, pp. 173–191. 2002.

[Erw09] M. Erwig. Software engineering for spreadsheets. *IEEE Software*, vol. 26:pp. 25–30, September 2009.

[Fis02] M. Fisher, M. Cao, G. Rothermel, C. R. Cook, and M. M. Burnett. Automated test case generation for spreadsheets. In *Proc. of ICSE '02*, pp. 141–151. 2002.

[Fis05a] M. Fisher and G. Rothermel. The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. *ACM SIGSOFT Software Engineering Notes*, vol. 30(4):pp. 1–5, 2005.

[Fis05b] M. Fisher and G. Rothermel. The EUSES spreadsheet corpus: A shared Resource for supporting experimentation with spreadsheet dependability mechanisms. In *Proc. of WEUSE '05*, pp. 47–51. 2005.

[Fow99] M. Fowler. *Refactoring: improving the design of existing code.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Fre66] R. S. Freedman. *Introduction to Financial Technology.* Academic Press, 20066.

[Gab10] P. Gabriel. *Software Languages Engineering: Experimental Evaluation.* Master's thesis, Universidade Nova de Lisboa, 2010.

[Gan77] C. Gane and T. Sarson. *Structured Systems Analysis: Tools and Techniques*. McDonnell Douglas Information, 1977.

[Gia96] D. Giammarresi and A. Restivo. Two-dimensional finite state recognizability. *Fundamenta Informaticae*, vol. 25(3):pp. 399–422, 1996.

[Gla67] B. Glaser and A. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Transaction, 1967.

[God02] M. Godfrey and Q. Tu. Tracking structural evolution using origin analysis. In *Proc. of IWPSE '02*, pp. 117–119. 2002.

[Gol47] H. Goldstein and J. von Neumann. *Planning and Coding Problems of an Electronic Computing Instrument*. McMillan, 1947.

[Gon10] O. Gonzalez. *Monitoring and Analysis of Workflow Applications: A Domain-specific Language Approach*. Ph.D. thesis, Universidad de los Andes, 2010.

[Gro08] D. M. Groenewegen, Z. Hemel, L. C. L. Kats, and E. Visser. WebDSL: A domain-specific language for dynamic web applications. In *Proc. of OOPSLA '08*, pp. 779–780. 2008.

[Hen94] D. G. Hendry and T. R. G. Green. Creating, comprehending and explaining spreadsheets: a cognitive interpretation of what discretionary users think of the spreadsheet model. *International Journal of Human-Computer Studies*, vol. 40(6):pp. 1033–1065, 1994.

[Her10] F. Hermans, M. Pinzger, and A. van Deursen. Automatically extracting class diagrams from spreadsheets. In *Proc. of ECOOP '10*, pp. 52–75. 2010.

[Her11] F. Hermans, M. Pinzger, and A. van Deursen. Supporting professional spreadsheet users by generating leveled dataflow diagrams. In *Proc. of ICSE '11*, pp. 451–460. 2011.

[Her12a] F. Hermans. Exact and near-miss clone detection in spreadsheets. *TinyToCS*, vol. 1(1), 2012.

[Her12b] F. Hermans, M. Pinzger, and A. van Deursen. Detecting and visualizing inter-worksheet smells in spreadsheets. In *Proc. of ICSE '12*, pp. 441–451. 2012.

[Her12c] F. Hermans, M. Pinzger, and A. van Deursen. Detecting code smells in spreadsheet formulas. In *Proc. of ICSM '12*. 2012. To appear.

[Her13] F. Hermans, B. Sedee, M. Pinzger, and A. van Deursen. Data clone detection and visualization in spreadsheets. In *Proc. of ICSE '13*. 2013. To appear.

[Hod08] K. Hodnigg and R. Mittermeir. Metrics-based spreadsheet visualization: Support for focused maintenance. In *Proc. of EuSpRIG '08*, p. 16. 2008.

[Hol09] S. Hole, D. McPhee, and A. Lohfink. Mining spreadsheet complexity data to classify end user developers. In *Proc. of ICDM '09*, pp. 573–579. CSREA Press, 2009.

[Jan00] D. Janvrin and J. Morrison. Using a structured design approach to reduce risks in end user spreadsheet development. *Information & Management*, vol. 37(1):pp. 1–12, 2000.

[Joh93] J. H. Johnson. Identifying redundancy in source code using fingerprints. In *Proc. of CASCON '93*, pp. 171–183. 1993.

[Jür09] E. Jürgens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *Proc. of ICSE '09*, pp. 485–495. 2009.

[Kam02] T. Kamiya, S. Kusumoto, and K. Inoue. CCfinder: A multilinguistic token-based code clone detection system for large scale source code. *TSE*, vol. 28(7):pp. 654–670, 2002.

[Kap08] C. J. Kapser and M. W. Godfrey. "cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering*, vol. 13(6):pp. 645–692, 2008.

[Kim05] M. Kim and D. Notkin. Using a clone genealogy extractor for understanding and supporting evolution of code clones. In *Proc. of MSR '05*. 2005.

[Kni00] B. Knight, D. Chadwick, and K. Rajalingham. A structured methodology for spreadsheet modelling. *Proc of EuSpRIG '00*, vol. 1:p. 158, 2000.

[Knu63] D. Knuth. Computer-drawn flowcharts. *Communication of the ACM*, vol. 6(9):pp. 555–563, 1963.

[Ko10] A. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrence, H. Lieberman, B. Myers, M. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck. The state of the art in end-user software engineering. *ACM Computing Surveys*, 2010.

[Kol02] R. Kollman, P. Selonen, E. Stroulia, T. Systä, and A. Zündorf. A study on the current state of the art in tool-supported uml-based static reverse engineering. In *Proc. of WCRE '02*, pp. 22–. 2002.

[Kom01] R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In *Proc. of SAS '01*, pp. 40–56. 2001.

[Kri01] J. Krinke. Identifying similar code with program dependence graphs. In *Proc. of WCRE '09*, pp. 301–309. 2001.

[Kru06] S. E. Kruck. Testing spreadsheet accuracy theory. *Information & Software Technology*, vol. 48(3):pp. 204–213, 2006.

[Lag97] B. Laguë, D. Proulx, J. Mayrand, E. Merlo, and J. P. Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In *Proc. of ICSM '97*, pp. 314–321. 1997.

[Lan05] M. Lanza, R. Marinescu, and S. Ducasse. *Object-Oriented Metrics in Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[Lev75] V. I. Levenshtein. On the minimal redundancy of binary error-correcting codes. *Information and Control*, vol. 28(4):pp. 268–291, 1975.

[Lyn94] M. Lynne and M. Keil. If we build it, they will come: Designing information systems that people want to use. *Sloan Management Review*, vol. 35(4), 1994.

[Mar79] T. D. Marco. *Structured Analysis And System Specification*. Prentice Hall PTR, 1979.

[Mar96] M. N. Marshall. Sampling for Qualitative Research. *Family Practice*, vol. 13(6):pp. 522–526, 1996.

[Mar01] R. Marinescu. Detecting design flaws via metrics in object-oriented systems. In *Proc. of TOOLS '01*, pp. 173–182. IEEE Computer Society, 2001.

[Mat64] R. Mattessich. *Simulation of the Firm through a Budget Computer Program*. R.D. Irwin, Illinois, 1964.

[May96] J. Mayrand, C. Leblanc, and E. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *Proc. of ICSM '96*, pp. 244–. 1996.

[McC03] A. McCallin. Designing a grounded theory study: Some practicalities. *Nursing in Critical Care*, vol. 8:pp. 203–208, 2003.

[Mit02] R. Mittermeir and M. Clermont. Finding high-level structures in spreadsheet programs. In *Proc. of WCRE '02*, p. 221. 2002.

[Moh10] N. Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. L. Meur. Decor: A method for the specification and detection of code and design smells. *IEEE Transactions on Software Engineering*, vol. 36(1):pp. 20–36, 2010.

[Nar90] B. Nardi and J. Miller. The spreadsheet interface: A basis for end user programming. In *Proceeding of The IFIP Conference on Human-Computer Interaction (INTERACT)*, pp. 977–983. North-Holland, 1990.

[Nov01] N. Novelli and R. Cicchetti. Fun: An efficient algorithm for mining functional and embedded dependencies. In *Proc. of ICDT '01*, pp. 189–203. 2001.

[O'B08] P. O'Beirne. Information and data quality in spreadsheets. In *Proc. of EuSpRIG '08*. 2008.

[O'B10] P. O'Beirne. Spreadsheet refactoring. In *Proc. of the EuSpRIG '10*. 2010.

[Olb09] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka. The evolution and impact of code smells: A case study of two open source systems. In *Proc. of ESEM '09*, pp. 390–400. 2009.

[Pan94] R. R. Panko and R. P. H. Jr. Individual and group spreadsheet design: Patterns of errors. In *Proc. of HICSS '94*, pp. 4–10. 1994.

[Pan96] R. R. Panko and R. P. Halverson Jr. Spreadsheets on trial: A survey of research on spreadsheet risks. In *Proc. of HICCS '96*, pp. 326–. 1996.

[Pan98] R. R. Panko. What we know about spreadsheet errors. *Journal of End User Computing*, vol. 10(2):pp. 15–21, 1998.

[Pan06] R. Panko. Facing the problem of spreadsheet errors. *Decision Line*, vol. 37(5), 2006.

[Pow09] S. Powell, K. Baker, and B. Lawson. Errors in operational spreadsheets: A review of the state of the art. In *Proc. of HICCS '09*, pp. 1–8. IEEE Computer Society, 2009.

[Raf09] J. Raffensperger. New guidelines for spreadsheets. *International Journal of Business and Economics*, vol. 2:pp. 141–154, 2009.

[Raj00] K. Rajalingham, D. Chadwick, B. Knight, and D. Edwards. Quality control in spreadsheets: a software engineering-based approach to spreadsheet development. In *Proc. HICSS '00*, pp. 133–143. 2000.

[Ron89] B. Ronen, M. Palley, and H. L. Jr. Spreadsheet analysis and design. *Communication of the ACM*, vol. 32(1):pp. 84–93, 1989.

[Ros87] A. Rosenfeld. Array grammars. In *Proc. of International Workshop on Graph-Grammars and Their Application to Computer Science*, vol. 291 of *LNCS*, pp. 67–70. Springer-Verlag, 1987.

[Rot97] G. Rothermel. Testing strategies for form-based visual programs. In *Proc. of ISSRE '97*, pp. 96–107. 1997.

[Rot98] 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 *Proceedings of the International Conference on Software Engineering (ICSE)*, pp. 198–207. 1998.

[Rot00] K. J. Rothermel, C. R. Cook, M. M. Burnett, J. Schonfeld, T. R. G. Green, and G. Rothermel. Wysiwyt testing in the spreadsheet paradigm: an empirical evaluation. In *Proc. of INCSE '00*, pp. 230–239. 2000.

[Roy09a] C. K. Roy. Detection and analysis of near-miss software clones. In *Proc. of ICSM '09*, pp. 447–450. 2009.

[Roy09b] C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, vol. 74(7):pp. 470–495, 2009.

[Roy10] C. K. Roy and J. R. Cordy. Near-miss function clones in open source software: an empirical study. *Journal of Software Maintenance*, vol. 22(3):pp. 165–189, 2010.

[Saj00] J. Sajaniemi. Modeling spreadsheet audit: A rigorous approach to automatic visualization. 2000.

[Sca89] D. Scanlan. Structured flowcharts outperform pseudocode: An experimental comparison. *IEEE Software*, vol. 6(5):pp. 28–36, 1989.

[Sca05] C. Scaffidi, M. Shaw, and B. A. Myers. Estimating the numbers of end users and end user programmers. In *Proc. of VL/HCC '05*, pp. 207–214. 2005.

[Shi99] K. Shiozawa, K. Okada, and Y. Matsushita. 3D interactive visualization for inter-cell dependencies of spreadsheets. In *Proc. of INFOVIS '99*, pp. 79–83. 1999.

[Sla89] R. Slater. *Portraits In Silicon*. The MIT PRess, 1989.

[Str72] G. Stromoney, R. Siromoney, and K. Krithivasan. Abstract families of matrices and picture languages. *Computer Graphics and Image Processing*, vol. 1(3):pp. 284–307, 1972.

[Tas98] A. Tashakkori and C. Teddlie. *Mixed methodology : combining qualitative and quantitative approaches*. Sage, Thousand Oaks, California, 1998.

[Win01] W. Winston. Executive education opportunities. *OR/MS Today*, vol. 28(4), 2001.

[Yin02] R. K. Yin. *Case Study Research : Design and Methods*. SAGE Publications, 2002.

[Zhu89] R. F. Zhu and T. Takaoka. A technique for two-dimensional pattern matching. *Commununications of the ACM*, vol. 32(9):pp. 1110–1120, 1989.

[Zou03] L. Zou and M. W. Godfrey. Detecting merging and splitting using origin analysis. In *Proc. of WCRE '03*, pp. 146–154. 2003.

Summary

Analyzing and Visualizing Spreadsheets

Spreadsheets are used extensively in industry: they are the number one tool for financial analysis and are also prevalent in other domains, such as logistics and planning. Their flexibility and immediate feedback make them easy to use for non-programmers.

But as easy as spreadsheets are to build, so difficult can they be to analyze and adapt. This dissertation aims at developing methods to support spreadsheet users to understand, update and improve spreadsheets. We took our inspiration for such methods from software engineering, as this field is specialized in the analysis of data and calculations. In this dissertation, we have looked at four different aspects of spreadsheets: metadata, structure, formulas and data.

Metadata

In spreadsheets, users can mix real data, such as 'Mekelweg 4' with so-called *metadata*: data that is used to describe data, such as 'Address'. When you want to know what a spreadsheet does, this metadata can be more relevant than the actual data. This is why we have developed an approach to extract this metadata. We detect metadata by looking at common places where it is usually found, such as in the first row or column. These common places are called 'patterns' and we defined four different version of them. If a pattern in found, it can subsequently be transformed into a class diagram, which describes the spreadsheet metadata in a structured way. These class diagrams can be used to re-implement, refine or improve the corresponding spreadsheets. Initially, these diagrams were intended to be used by (non end-user) software developers. However, it was when applying this research in practice that I found that representing spreadsheets with diagrams could also support end-users.

Structure

This is why subsequently, we started to investigate the *information needs* of end-users. In this research, we found that the most important information need of spreadsheet users is to get a quick overview of how the worksheets of a spreadsheet are connected. Dataflow diagrams proved to be a very viable method of visualizing the dependencies between worksheets, as shown by a case study we performed at Robeco, a large Dutch investment bank. As one of the subjects in our case study said:"The global view reminds me of the plan I had when building this spreadsheet. Normally, I just go through the worksheets from left to right, but that is not the logical way."

Formulas

When we observed spreadsheet users interacting with these visualizations, we found that they used them not only for understanding spreadsheets, but also for assessing their quality. This is why we formalized this formerly informal assessment of spreadsheet quality by introducing *inter-worksheet smells*. Since we asserted that worksheets and their connections strongly resemble classes and their relations, we decided to use inter-class code smells as our starting point. The subsequent evaluation showed that annotating the dataflow diagrams helps spreadsheet users to judge spreadsheet quality even more. One of the subjects stated, upon being confronted with the annotated dataflow diagram: "I should really take some time to improve these formulas, since this is already confusing for me, so it must be terrible for others".

While researching the topic of smells, we noticed that users also felt the need to assess individual formulas for maintainability. This is why we expanded our idea of spreadsheet smells into the realm of spreadsheet formulas. Again we took our inspiration from existing code smells, yet this time at the intra-class level. With this approach, we were able to locate complex formulas in a spreadsheet. Users in the subsequent case study with 10 professional users, again conducted at Robeco, confirmed that the selected formulas were indeed the least maintainable and could be improved: "I understand why this formula is selected by your system, it is quite long." Another subject, when looking at a formula that referred to no less than 17 ranges said "this formula is a real puzzle".

Data

Finally, we researched the applicability of *clone detection* within spreadsheets. Cloning, or copy-pasting, can diminish understandability, since it can be unclear for the user what the real source of data is. Furthermote cloning can be error-prone, since in the case of an update, this update has to be performed on all copies. When validating our clone detection approach, we found that both exact clones (those matching 100%) and near-miss clones occur in spreadsheets and that both pose different threats. Exact clones mainly impact the user's perspective

of spreadsheets ("I did not know these values were copied from that source"), while near-miss clones really cause trouble ("this value should have been updated months ago"). We learned that cloning can be a source of errors, as even creators of the spreadsheets did not know all relations by heart. In one of the case studies, our approach was able to detect and repair severe errors in spreadsheets used for the inventory of the south-Dutch foodbank. This has shown that clone detection in spreadsheets can solve real-life spreadsheet problems.

Conclusion

We found that methods from software engineering can be applied to spreadsheets very well, and that these methods support end-users in working with spreadsheets.

Felienne Hermans

Samenvatting

Analyse en visualisatie van spreadsheets

Spreadsheets worden heel veel gebruikt in het bedrijfsleven: ze zijn de nummer 1 tool voor financiële analyse, maar worden ook in andere gebieden gebruikt, zoals logistiek en planning. Omdat ze flexibel zijn en je je resultaten meteen ziet, zijn ze makkelijk te gebruiken voor niet-programmeurs.

Maar zo makkelijk als spreadsheets te bouwen zijn, zo lastig kunnen ze zijn om te doorgronden en aan te passen. Het doel van dit proefschift is om spreadsheet gebruikers te helpen bij het begrijpen, aanpassen en verbeteren van hun spreadsheets. We hebben hiervoor gekeken naar methodes in *software engineering*, want dit gebied specialiseert zich in het analyzeren van data en berekeningen. In dit proefschrift hebben we gekeken naar vier verschillende aspecten van een spreadsheet: metadata, structuur, formules en data.

Metadata

In spreadsheets kunnen gebruikers echte gegevens zoals 'Mekelweg 4', mixen met *metadata*: data die data beschrijft, zoals 'Adres'. Als je wilt weten wat een spreadsheet doet, kan deze metadata je soms meer vertellen dan de data zelf. Daarom hebben we gewerkt aan een methode om deze metadata te extraheren uit een spreadsheet. We doen dat door te kijken naar plaatsen waar deze meestal gevonden wordt: aan het begin van de rij of kolom. Deze plaatsen noemen we patronen en we hebben vier van deze patronen opgesteld om metadata te kunnen vinden. Als een patroon gevonden is, kan het worden vertaald in een klassediagram, dat de metadata op een gestructureerde manier beschrijft. Deze klassendiagrammen kunnen vervolgens gebruikt worden om het spreadsheet te herbouwen, te verfijnen of te verbeteren.

Initieel waren deze diagrammen bedoeld voor programmeurs, zodat zij de

spreadsheets konden verbeteren. Maar toen we ze gingen gebruiken in de praktijk, bleken de diagrammen ook voor eindgebruikers heel nuttig.

Structuur

Daarom gingen we ons vervolgens meer richten op eindgebruikers. We begonnen met het in kaart brengen van de *informatiebehoefte*: wat willen spreadsheetgebruikers eigenlijk weten als ze aan het werk zijn met een spreadsheet. We kwamen erachter dat het het belangrijkste is voor gebruikers om snel een overzicht te krijgen van de relaties tussen de tabbladen van een spreadsheet. Dataflowdiagrammen, zoals beschreven in Hoofdstuk 3 waren een goede manier om afhankelijkheden tussen tabbladen weer te geven, zagen we in een case study bij Robeco: "Dit diagram doet me denken aan het plan dat ik had toen ik dit sheet aan het maken was. Normaal loop ik van links naar rechts door de tabbladen, maar dat it niet zo logisch."

Formules

Toen we gebruikers bestudeerden die met onze diagrammen aan de slag gingen, zagen we dat ze die niet alleen gebruikten voor het begrijpen van spreadsheets, maar ook voor het beoordelen van de kwaliteit. Als er een 'warrig' plaatje bij een spreadsheet hoorde, vonden ze dat spreadsheet ook minder goed, zelfs zonder het bijbehorende sheet bekeken te hebben. Daarom hebben we deze intuïtie geformaliseerd en uitgedrukt in *inter-worksheet smells*, zwakke plekken in het de relatie van de tabbladen. We hebben hierbij *code smells* als uitgangspunt genomen, omdat de verhouding tussen tabbladen sterk doet denken aan die tussen klassen in source code. De evaluatie toonde aan dat deze diagrammen spreadsheet gebruikers helpen bij het beoordelen van de kwaliteit van een spreadsheet. Een van de deelnemers zei: "Ik moet echt iets aan deze formules doen, als ze voor mij al zo ingewikkeld zijn, dan moet het voor andere gebruikers helemaal vreselijk zijn".

Toen we aan de 'smells' werkten, merkten we dat gebruikers ook graag individuele formules wilden beoordelen op kwaliteit. Daarom hebben we het idee van de smells uitgebreid naar het niveau van formules. Weer namen we code smells als uitgangspunt, maar deze keer de smells binnen een klasse. Hiermee konden we de meest complexe formules in een spreadsheet opsporen. De evaluatie met 10 gebruikers, wederom bij Robeco, bevestigde dat de formules die onze methode selecteerde ook daadwerkelijk ingewikkeld waren. Een van de deelnemers zei, over een formule die naar maar liefst 17 verschillende celbereiken verwees: "deze formule lijkt wel een puzzel".

Data

Tenslotte hebben we onderzoek gedaan naar het opsporen van *clones* in spreadsheets. Clones, of copy-pasten, kan een negatieve impact hebben op het gemak waarmee een spreadsheet begrepen kan worden, omdat het niet duidelijk is wat

het origineel is en wat de kopie. Het is ook foutgevoelig, want in het geval van een verandering moeten alle kopiën aangepast worden. We hebben een methode ontwikkeld om clones op te sporen, gebaseerd op clone detectie in source code. Toen we deze methode evalueerden, zagen we dat zowel exacte clones (die 100% overeenkomen) en near-miss clones (clones met kleine verschillen) voorkomen en dat beiden op een andere manier problemen veroorzaken.

Exacte clones verminderen het begrip van een spreadsheet ("ik wist niet dat deze waardes gekopieerd waren"), terwijl near-miss clones fouten onthullen ("deze waarde had allang geüpdatet moeten worden"). Uit deze studie leerden we dat clones gevaarlijk zijn, omdat zelfs de makers van de sheets vaak de copy-paste-relaties niet kenden. In één van de case studies wisten we met onze aanpak meerdere fouten te vinden en te verhelpen in het voorraadbeheersysteem van de voedselbank Zuid-Nederland. Dit laat zien dat clone detectie echte spreadsheet problemen kan oplossen.

Conclusie

We concluderen dat methoden uit de software engineering goed toepasbaar zijn op spreadsheets en dat deze methoden eindgebruikers ondersteunen bij het werken met spreadsheets.

Felienne Hermans

Curriculum Vitae

Félienne Frederieke Johanna Hermans

February 4, 1984
Born in Oudenbosch

1996-2002
High School (Gymnasium)
Mencia de Mendoza Lyceum, Breda
Nature & Science profile with Latin

2002-2007
B.Sc. in Computer Science
Eindhoven University of Technology

2006-2008
M.Sc. in Computer Science & Engineering
Eindhoven University of Technology
Specialization: Design and Analysis of Systems
Thesis: Proving SN-infinity automatically

2008-2012
Ph.D. in Software Engineering
Delft University of Technology

2010-present
Founder
Infotron