Universidade do Minho

Hugo Manuel Sousa Ribeiro

Spreadsheet Smells

Tese de Mestrado
Mestrado em Informática
Trabalho efectuado sob a orientação de
Prof. Dr. João Saraiva e Dr. Jácome Cunha

Novembro 2011

2

acknowledgements

This dissertation has been influenced by many people, some over the past year, and some over my entire life. I am specially thankful to my supervisors Ph.D João Saraiva and Ph.D Jacome Cunha for their amazing skill in matter of guidance, they always knew how to put me in the right track.

Then, I must thank my parents and my brother for all the patience, support, and all the teachings given. They made me who I am today, as a student and a human being.

I mus thank my girlfriend Rita Barros for all the support, patience to put up with my bad mood, and willing to listen when I felt more below. She was the best!

I cannot forget my friends, specially Nhoca, Lautas and Paulo Lopes they had an important role helping me to relax, keep the cool, make company and gave their support when I needed.

I must also thank Software Improvement Group in general, José Pedro, Miguel Ferreira and Joost Visser for the time spent, hospitality and knowledge provided during my stay at their headquarters.

To all the personal that worked close to me during this past year, specially Christophe Peixoto, André Riboira, Jorge Mendes and others, that helped me when some problem showed and I tried the answer with them. And to all the people that I didn't mention but here and there helped me to go through this past year.

Resumo

Olhando para as folhas de cálculo como uma linguagem de programação faz dela a linguagem mais usada em todo mundo. Na verdade alguns estudos dizem que os chamados de programadores não-profissionais excedem em grande número os programadores profissionais. Por causa disso e da falta de mecanismos como abstracção, encapsulamento, ou programação estruturada, 90% das folhas de cálculo têm erros. Esta dissertação apresenta um esforço feito para ajudar com este problema.

O objectivo principal desta dissertação é desenvolver uma ferramenta que permita detectar possiveis problemas em folhas de cálculo, esses problemas chamamos "smells" (uma indicação superficial que geralmente aponta para um problema mais profundo). Para isso, introduzimos alguns conceitos teoricos como metricas e smells, como por exemplo o Smell das Dependências Funcionais que adaptamos das bases de dados. Apresentámos o estudo que foi feito, mostrando os resultados obtidos pela ferramenta aplicada a um grande conjunto de folhas de cálculo, o EUSES Corpus.

Abstract

Viewing spreadsheets as a programing language makes it the most used programming language worldwide. In fact some studies performed show that the so called "end-user" programmers surpass the professional programmers by far. Because of this and the lack of support for abstraction, testing, encapsulation or structured programming, 90% of the spreadsheets in the real world have errors. This dissertation presents an effort to help with this problem.

The main goal of this dissertation is to create a technique that allows us to detect probable problems in spreadsheets, problems called smells (a surface indication that usually corresponds to a deeper problem). Thus, we first introduce some theoretic concepts like metrics and smells, such as for instance the Functional Dependency Smell that was adapted from databases. We present the study we made, showing the results obtained with the tool applied to a large set of spreadsheets, the EUSES corpus.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The spreadsheets are used worldwide by all kind of person, specially non-professional programmers, the also called "end-user" programmers [22]. An end-user can be a teacher, an engineer, a student, anyone that is not a professional programmer is considered one. These end-user programmers outnumber the professional programmers by far. In fact a study performed by Scaffidi *et al.* in 2005 estimates that only in U.S. exists 11 million of end users against only 2.75 million of professional programmers [31]. Still in this study they project to 2012 a total of 90 millions of end users from which 55 million will be from spreadsheets or databases.

The dimension of users presented by Scaffidi implies that millions of new spreadsheets are created every year. And because these end users are not professional programmers when they create a new spreadsheet they usually do not look to any principles of programming, instead all they care is getting the job done. This approach by the users, and the lack support for abstraction, testing, encapsulation, or structured programming in spreadsheets as a programming language, leads to the results presented in some studies that report that up to 90% of real-world spreadsheets contain errors [30].

A spreadsheet is a computer adaptation of a paper ledger sheet and it consists of a grid of rows and columns. It is an environment that simplifies manipulation of numbers. A spreadsheet is a digital document composed by a grid of rows and columns filled with cells containing three types of data, *Labels*, *Constants* and *Formulas*. Labels are text entries mainly used to identify items and help understanding the spreadsheet. Constants are numeric, dates or booleans entries that can be used in computation; Formulas are the entries that have an equation inside (formula) that will be used to display the resulting value.

Formulas must start with "=" symbol and constant values can be used as parameters. For instance a formula can be used to sum all values from a row or column. Formulas can also be just a reference to other cells. By doing this the cell with that formula will present the same value presented in the referenced cell.
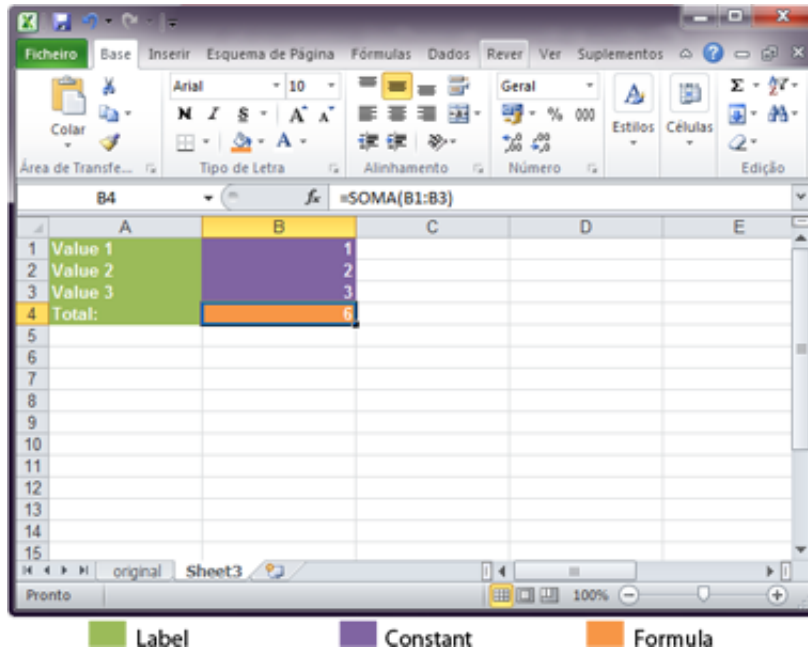
Figure 1.1: *Example of a spreadsheet.*

In spreadsheets the mapping to the cells is made by using a combination of letters (columns) and numbers (rows) for example: the code "A10" represents the first column row 10. In Fig. 1.1 we can see an example of a spreadsheet. The values in the column A (green values[1]) are Labels, the values in purple are constants (numbers) that will be used by the formula in the cell B4 (orange) to calculate their sum.

Nowadays we can find many software to work with spreadsheets, from the most common solutions desktop-based as *Excel* [5] or the open source versions of it, the *calc* from *LibreOffice* [9] or from *OpenOffice* [10], to the more recent approach brought by *Google* a web-based system like *Google Docs* [13]. This new branch of spreadsheets is just a consequence of the evolution of the technology that is rapidly advancing to full web systems.

Many studies refer the high rate of errors in spreadsheets and how this is costly to companies. Thus, in this work we look for smells in spreadsheets so we can lead to the way of improving those spreadsheets. Smells were introduced to the software engineering

---

[1]Colors are visible in digital version of this document.

by Martin Fowler [11] in 1999 and the name points directly to the concept: *something that smells is something that does not look correct.* This concept has been studied in parallel by us and by Hermmans in [7], but in different perspectives, we do not trace a parallelism between our smells and the ones introduced by Fowler.

In order to detect something that does not look correct in spreadsheets we analyze a large repository of spreadsheets; the EUSES corpus. We used the EUSES Corpus [17] since it contains more than 5000 spreadsheets. The study that we will present is a massive analyze to a randomly selected sample from the EUSES. First we selected a bunch of spreadsheets from each category of the EUSES, and then from those we selected ten sheets from each category, all the picks were random.

1.1 Structure of the dissertation

This dissertation is structured as follows:

Chapter 2: discusses the state of the art and is where we will present some previous studies done by other authors.

Chapter 3: explains the need of measuring software and some guidelines of how should we do it. This chapter also introduces the concept of metric and introduce the metrics that can be used in spreadsheets.

Chapter 4: presents the concept of software smell and how we use it in spreadsheets. In this chapter we describe in detail the smells that we will be using in the "SmellSheet Detective".

Chapter 5: is where all the evolution results will be shown. This chapter is where we present the study *per se*, we describe the sample used, explain the process, present the tool and the results obtained.

Chapter 6: is where we take all the conclusions of the work, and point out some interesting future work.

Chapter 2

State of the Art

Summary:

In this chapter we present some of the research that is related to ours. We present works about the measurements of softwares in general, spreadsheet analysis and spreadsheet errors and also present some work in the field of smells, on software and spreadsheets.

Efforts related to our research include a wide range of studies. From studies about spreadsheets errors, spreadsheet analysis, spreadsheet visualization and smells in spreadsheets, to general analysis of software or metric definitions. In this chapter, we will do a brief exposition about those studies.

Measurements:

The main purpose of Alves *et al.* [2] is to give meaning to values obtained by the use of metrics in measuring software. They present a way to define relevant thresholds so that the results obtained may be better understandable and more meaningful. Alves *et al.* work rests in three assumptions: "*i) it should respect the statistical properties of the metric, such as scale and distribution; ii) it should be based on data analysis from a representative set of systems (benchmark); iii) it should be repeatable, transparent and straightforward to execute.*"

Still in the field of measurements we can underline the work made by Heitlager *et al.* [15]. In this paper, it is presented a way to measure the maintainability of a software that is based on the standard ISO 9126 [18]. This model is not just theoretical, they really

use it in the analysis made by their company "Software Improvement Group".

Spreadsheet Analysis:

Related to spreadsheet analysis we must stress the work of Bergar [3] who presents a list of complexity metrics to be used in spreadsheets. This work does not provide any justification for the metrics chosen. Hodnigg *et al.* [16] defend that the comprehension of a spreadsheet may be simplified by a good technique of visualization, so they use complexity measures as an indicator of a proper visualization. They divide their metrics in three groups: *general metrics*, where they consider the number of formulas or the number of non-empty cells; *formula complexity*, where they include metrics such as the chain calculation or the Fan-in and Fan-out of a cell; and finally the metrics they call *further complexity arguments* where they measure the existence of any scripts, *e.g.* VBA or python, the existence of user-defined functions or external sources.

Smells:

Fowler [11] was the first to introduce the concept of smell and to create a list of 22 smells pointing a possible solution for each one of them. In the sequence of Fowler study, Mantyla *et al.* [21] has created a taxonomy for the smells listed by Fowler so they could be easier to understand. They created five groups of smells, the bloaters, the object-oriented abusers, the change preventers, the dispensables and the couplers.

Still in smells but now to spreadsheets we should refer the very recent work of Hermans *et al.* [7] that used smells to detect weaknesses in spreadsheets. They also make a relation between their smells for spreadsheets with those that Fowler listed, presenting, like Fowler did, a possible way of refactoring. Their work differs from the work we present in this thesis in the fundamental approach to define spreadsheet smells: while Hermans adapt Fowler smells to the spreadsheet realm, we analyze a large corpus and based on that we define spreadsheet specific smells.

Spreadsheet Errors:

In this specific field, we refer Powell *et al.* [26], [27] studies, where they dissects errors in spreadsheets, which type of errors occur, their consequences, which ones are more common, how to prevent them and how to detect them.

More recently by Panko *et al.* [23] proposed a taxonomy for spreadsheet errors to help other researchers. This taxonomy is a revision of one previously proposed by him [24]. In Figure. 2.1 we can see the new proposed taxonomy.
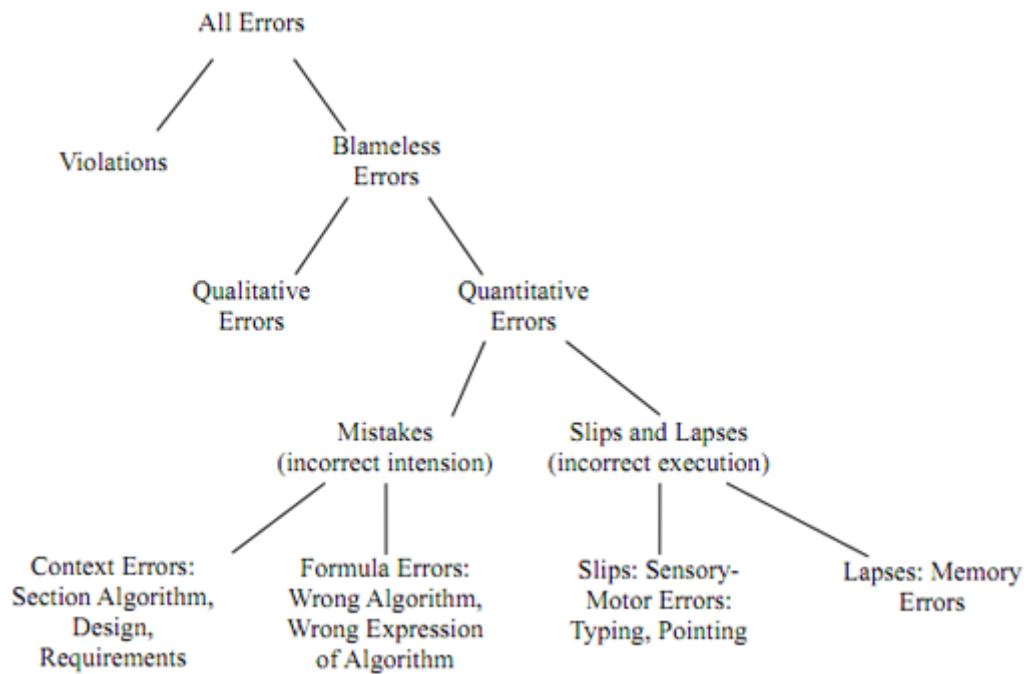
Figure 2.1: *Spreadsheet error taxonomy.*

Chapter 3

Software Quality Assessment Based on Metrics

Summary:

In this chapter we start by introducing the need of measure things daily and then we try to explain why are these measurements needed in the computer science, and how they may be done. Finally we talk about metrics for spreadsheets, where we present five groups of metrics and also some metrics that can be used to measure things in spreadsheets. These metrics are grouped by levels: cell, sheet and spreadsheet.

*" **Measurements:** is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly define rules"* Fenton *et al.* [8]

In our daily life we use measurements in almost every task that we accomplish even without realizing it. In the morning when we wake up we measure the time needed to get to work in time, we measure the time needed to warm up the coffee so it do not get to hot or to cold, if we need to pay something in a shop we measure the correct change so we do not get deceived, and in many other tasks. Measurements are built-in our everyday and we need them to understand it and properly interact with it.

In computer science this reality is transposed to our work and we use it for instance to measure the quality of software. Software quality measurement is the quantification to what extent a system possesses the desirable characteristics. This can be made through

qualitative or quantitative measures, either way a measurable set of attributes related to the desired characteristics must be stressed out. In the ISO/IEC 9126 standard [18] is described a model for software product quality that categorizes the global notion of quality into six main characteristics: functionality, reliability, usability, efficiency, maintainability, and portability. This standard is used for example, by the company SIG[1] to create their own model to measure maintainability [15]. In their model they have a ranked base approach that rates each system in five levels ++, +, o, -, - - in which ++ is the best result and - - is the worst. This rating is done by analyzing a large set of systems in order to create thresholds so they can rate the systems according their level. In the evaluation of our results we will use a similar technique to rate the smells found in the spreadsheets.

In this chapter we briefly will discuss about software metrics, why to use them, how to use them, some issues that occur when defining what to measure and identify some kinds of popular metrics used to measure software. Then, in the second part of this chapter we will present spreadsheet metrics making the link between spreadsheets and programs written in other programing languages. Furthermore, we will be presented a catalog of metrics for spreadsheets.

3.1   Program Metrics

Knowing and accepting the fact that we constantly measure things in order to control our life, I will try to explain how this measures are used in the field of computer science. Computer science involves several activities like analyzing, planing, costing, testing, implementing, maintaining, and others. Because each activity can be quite distinct, different measurements can be needed in order to properly quantify some attribute from one entity. Depending on the characteristic that we want to measure a different set of measurements will be needed.

3.1.1   Why measure things?

Departing from *Tom DeMarco's* quote *"You can't control what you can't measure!"* [6] we can agree and say that the global purpose of measure things is to control every (possible) outcome from a certain situation. But this measurements can be made to "control" many

---

[1]Software Improvement Group. http://www.sig.eu

characteristics of some product. Like we saw before we have different kinds of measures and each can serve more than one purpose, generally each one helps to understand a given property. For instance, if we want to study the maintainability of one software artifact, measure things like *Lines Of Code* (LOC), or the *Size of the methods/functions* can help us with that, instead if we want to measure the quality of software, measurements like time or effort need to be done.

Despite each metric gives an indicator of some possible attribute, when creating a more global evaluation about one entity we should not make a straightforward analysis for each individual metric, because, for instance when we have two software programs one with 1000 LOC and other with 500 LOC, say that the one with 1000 is easier to maintain may not be true. We need to see each metric as a portion of the whole, and how many more metrics better will be the final conclusions.

3.1.2 How to measure things?

Sometimes in computer science the definition of what and how something should be measured can be quite difficult to define because there are attributes that can depend on the context or interpretation. For instance, if we want to measure attributes like width or height from one person, they are quite simple to measure, but if we want to measure the beauty or the IQ the task becomes a more complicated because beauty and IQ are subjective measures and depend on how or who measures them. In order to surpass these obstacles we have to define objectives.

As we can see in Figure. 3.2 the definition of objectives always depend on the priorities from people involved. A client will need different information about a product from a manager, and because of this we have to make them as specific as possible and not let any interpretation to be made. They must be clear and simple otherwise the conclusions taken from them can also depend on interpretation. For instance, when performing a study that measures the percentage of fat women in Portugal if we just say "30% of the women from Portugal are fat", we are leaving the attribute fat to interpretation, so in order to avoid different interpretations we must explain what do we mean by fat. Instead we should say something like "30% of the women from Portugal have 10kg more than the advised in relation to their age and height".
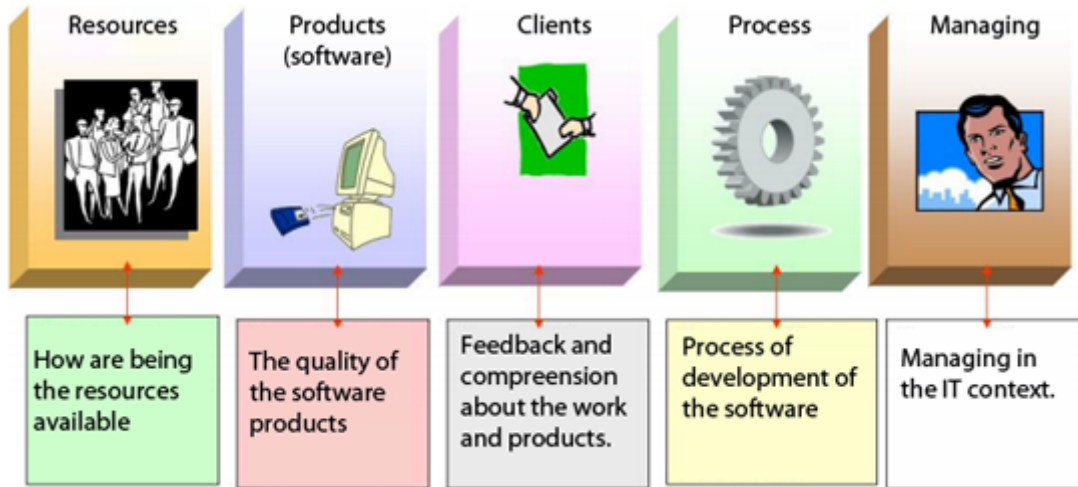
Figure 3.1:   *Different   metrics   for   different   stages.   (Adapted   from http://www.cefetrn.br/placido/disciplina/pgp/aulas/Metricas.pdf)*

3.1.3   Software Metrics

A software metric is the evaluation of a property from one software artifact by looking directly to the source code. There are many different software metrics, and each can help to measure many characteristics. If we look at the ISO/IEC 9126 standard [18] six main characteristics are used to give the notion of quality: functionality, usability, efficiency, maintainability and portability. These characteristics are divided in 27 sub-characteristics, and each sub-characteristics uses one or more metrics for its measurement. For instance, in Heitlager *et al.* [15] they explain a model to measure maintainability of a software using specific metrics: lines of code, cyclomatic complexity or code duplication.

The ISO/IEC 9126 will also be used to create the quality model used to classify some spreadsheet metrics presented in the next section.

3.2   Spreadsheet Metrics

The use of spreadsheets by non-professional programmers is well know, those that we call "end-user" programmers, raise exponentially the total number of users of spreadsheets. If we count with this "end-user" programmers, the spreadsheet programming language is the language with more programmers worldwide. But not only "end-user" programmers use spreadsheets, there are reports of many companies losing money due to errors in spread-

sheets [14], meaning that there are many companies using them.

Due the current high complexity of spreadsheets and their frequent use [29], [16], they must be looked at like any other computer program from another programing language. This means that the use of metrics to measure and quantify entities in spreadsheets can be made like in other programing language. In fact, we can easily map the structure of a spreadsheet to an object oriented programing language: we just have to look at spreadsheets as a source code file, sheets as classes, cells as methods and functions as statements. Knowing this Peixoto [25], created a quality model that we will be using to classify some spreadsheet metrics according five characteristics .
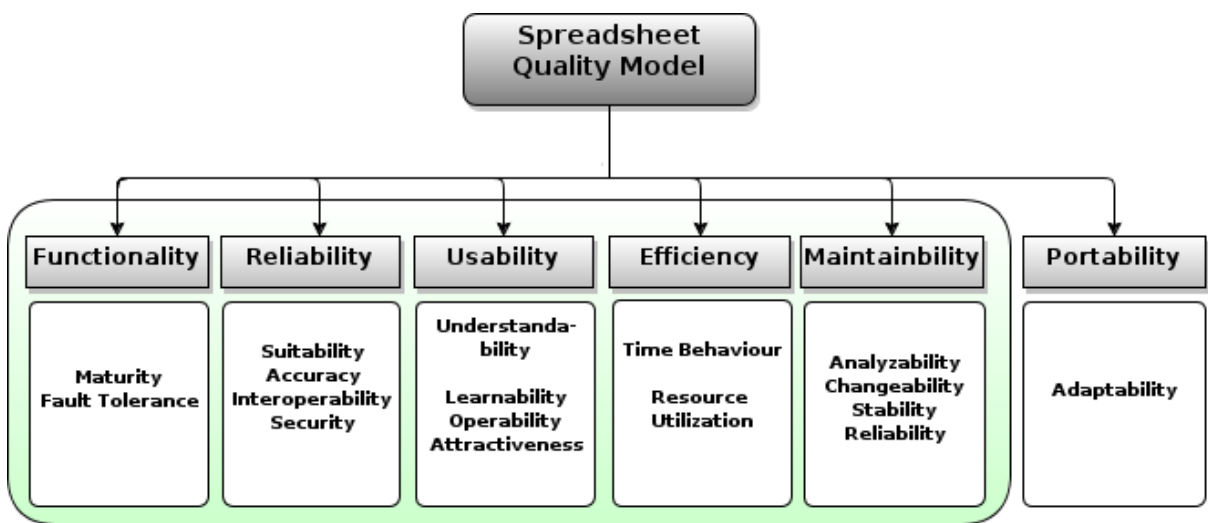
Figure 3.2: *The Software Quality Model ISO/IEC 9126.*

In the Figure. 3.2 we can see the software quality model created by Peixoto.

3.2.1 Functionality

This characteristic measures the ability of software to satisfy the user needs. This depends in four sub-characteristics:

- Suitability: measures if the spreadsheet has the right properties for the purpose it is meant. It is measured with metrics *Number of incongruences* and *Number of references to Blank Cells in Formulas*;

- Accuracy: is a faithful measurement or representation of the true, correctness. It is

measured using metrics as *Number of Output Cells with Errors/Bad Content*, *Number of incongruences* and *Number of Blank Cells referenced in Formulas*;

- Interoperability: is the ability of two or more Sheets or components to exchange information and to use the information that has been exchanged. This uses metrics as *Data been exchanged between Sheets*, *Quantity of rightful formulas*, *Total of Cells with references* and *Total of references*;

- Security: verifies the existence of protected cells, sheets or even the entire workbook. This uses metrics as *Protected Formulas*, *Protected Cell for Data only for reading* and *Use Password to lock Workbook/Worksheets*.

3.2.2   Reliability

This is the is capacity from a software to maintain its quality after a period of time and under specific conditions, it depends in three sub-characteristics;

- Maturity: evaluates if the spreadsheet is fully developed, for this metrics like: *Number of Labeled Rows/Columns that are empties*, *Number of Blank Cells in a matrix*, *Number of Blank Cells* and *Difference between the Sheets that the Spreadsheet have*, and the ones been used, are used.

- Fault Tolerance: is the property to continue operating properly in the event of one or more faults within some of its components, for this we use metrics like *Number of Cells been referenced (Directly or indirectly) by many other Cells* and *Number of Complex Formulas*.

- Recoverability: is the capacity to restore a previous state, this characteristic does not apply because a spreadsheet can not restore itself.

3.2.3   Usability

Usability is the capability of a software being understood to the users, this characteristic depends in four sub-characteristics;

- Understandability: is the capacity of a spreadsheet being understood, for this we measure *Different colors for different types of Data*, *Separate Input, Computation and Output* and *Number of Cells*.

- Learnability: evaluates the easiness for the user to use the spreadsheet, for this we use metrics *Number of Cells*, *Different colors for different types of Data*, *Separate Input, Computation and Output*, *Number of Complex Formulas* and *Amount of Data being exchanged between Sheets*.

- Operability: evaluates the capacity of work with the spreadsheet, for this we measure if it has *Data Validation Drop Down Lists* and *Separated Inputs, Computation and Outputs*.

- Attractiveness: measures how attractive is to the user, for this we measure the use of colors, the existence of *Data Validation Drop Down Lists* and *Separated Inputs, Computation and Outputs*.

3.2.4   Efficiency

Efficiency is the ratio resources/performance of a software, this characteristic depends in two sub-characteristics.

- Time Behavior: estimates the computing time. For this we use the *Number of V-Lookup's* and other search formulas, and *Number of Complex Formulas*.

- Resources Utilization: estimates the resources needed. For this we use the *Number of V-Lookup's*, the *Amount of Blank-Cells* and *Number of Complex Formulas*.

3.2.5   Maintainability

Maintainability is the ease which a software can be modified/updated. This characteristic depends in four sub-characteristics.

- Analyzability: measure the capacity to analyze a spreadsheet, in order to conclude the effort needed for diagnosis deficiencies. In this we measure the *Number of Cells*, if the data is or not well organized, the *Number of References* and the *Number of Formulas*.

- Changeability: evaluates the ease of change of a spreadsheet and concludes the effort needed for that modifications. In this characteristic we have to measure *How Well is the Data Organized*, the *Number of Referenced Cells* and the *Number of Cells*.

- Stability: evaluates how stable is a spreadsheet. For this we use the *Number of Complex Formulas* and the *Number of Cells Referenced by Other Cells (directly or indirectly)*.

- Testability: evaluates how well can we test a spreadsheet. For this we only use the *Number of Formulas*.

In Peixoto's dissertation he also discusses portability, but we will not use it in the classification.

These classifications were also done in three levels, cell, sheet and spreadsheet. From many of the metrics shown in Table A.1, Table A.2, Table A.3(Appendix A) we can infer new ones. For instance to measure the data density from one sheet we can somehow use the "#Cells" and the "#Blank Cells" to obtain a valid measure.

Cell Level: At the cell level Table A.1 we only have a few metrics, some of the metrics identified were the Fan-in, Fan-out, that together represent all the references from a cell, we also have references to empty cells and constants in formulas.

Sheet Level: At the sheet level we have a bigger set than on the cell because it uses all the results from the cell. Besides the metrics from the cell level we also have those in Table A.2.

Spreadsheet Level: At the spreadsheet level the set grows because, like at the sheet level, it inherit metrics from cell level, on spreadsheet happens the same but from the sheet level.

Chapter 4

Bad Smells

Summary:

In this chapter we first introduce the notion of smell in software presenting a taxonomy that uses a catalog of smells previously created by Fowler. We then discuss spreadsheet smells and present in detail our catalog of smells, the ones that we will use in the "SmellSheet Detective". In this catalog we have statistical smells, type smells, input smells, functional dependencies smells and the ochiai smell.

*"**Code Smell:** A code smell is a surface indication that usually corresponds to a deeper problem in the system."* Martin Fowler Website

The notion of "bad smell" was introduced to the computer science by Fowler in [11], and it emerged in his book because he felt the need to define when and where to apply internal structure improvements to a software. Since that was a complicated task he used "bad smells" as a flag to do it. A "bad smell" is an indicator of some possible bigger problem, like usually we say, something that smells.

This "new" notion is just an helpful tool and still depends on some previous criteria definition to be used. For instance the notion of what is a too big method can depend on the product, who analyzes it and the purpose of the analysis. Like the author says *"no set of metrics rivals informed human intuition"* [11].

In spite of the fact that this concept was thought to object oriented programing language, I think we can say that it fits in spreadsheets like a glove because in spreadsheets we almost

never can tell if something is an error. Most of the times we point to something that does not feel right or something that could be done in a better way.

In the first part of this chapter we will see a list of smells that was introduced by Fowler in [11] but organized by Mantyla *et al.* [21] taxonomy. Then we will talk about smells in spreadsheets, were we will introduce the list of smells that we created and explain in detail how each one of them works.

4.1   Software Smells

Martin introduced the list of software smells in 2000. In 2003 Mantyla created a taxonomy to group them so it would be easier to understand them. This taxonomy is a posterior improvement of the taxonomy made to his thesis[1].

Group	Smells
The bloaters	Long Methods
	Large Classes
	Long Parameter List
	Long Methods
	Primitive Obsession
The Object-Orientation Abusers	Switch Statements
	Temporary Field
	Refused Bequest
	Alternative Classes with Different Interfaces
The Change Preventers	Divergent Change
	Shotgun Surgery
	Parallel Inheritance Hierarchies
The Dispensables	Lazy Class
	Data Class
	Duplicate Code
	Dead Code
	Speculative Generality
The Couplers	Feature Envy
	Inappropriate Intimacy
	Message Chains
	Middle Man

Table 4.1: Mantyla Taxonomy

---

[1]http://www.soberit.hut.fi/mmantyla/badcodesmellstaxonomy.htm

4.2 Spreadsheet Smells

The smells mentioned by Fowler are in single flat list, but Mantyla has created a taxonomy for all these smells. Similarly to what Mantyla has done we also grouped our smells by categories, five to be more specific: *Statistical Smells*, *Type Smells*, *Content Smells*, *Functional Dependencies Based Smell*, *Ochiai Semlls*. We will see how are they used by in our application.

	A	B	C	D	E	F	G	H	I	J
1	code	upc	description	size	store_nr	week	qty	price	on Sale	profit
2	653	1111140009	DOVE DISH LIQUID		101	385	2	1.99		26.78
3	653	1111140009	DOVE DISH LIQUID	42 OZ	101	386	o	1.99		26.78
4	653	123	DOVE DISH LIQUID	42 OZ	101	387	4	1.99		37.83
5	653	1111140009	DOVE DISH LIQUID	42 OZ	101	388	4	1.99		37.83
6	653	1111140009		42 OZ	102	391	6	2.19		38.49
7	653	1111140009	DOVE DISH LIQUID	42 OZ	102	392	8	2.19		33.47
8	654	1111143002	SUNLIGHT DISH LIQUIDS	64 OZ	100	383	9	2.99		21.97
9	654	1111143002	SUNLIGHT DISH LIQUID	64 OZ	100	384	21	2.99		21.97
10	654	1111143002	SUNLIGHT DISH LIQUID	64 OZ	100	385	10	2.99		21.97
11	654	1111143002	SUNLIGHT DISH LIQUID	64 OZ	100	386	6	2.99		21.97
12	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	102	391	35	3.39	S	25.16
13	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	102	392	4	3.79		23.82
14	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	383	0	0.00		0.00
15	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	384	0	0.00		0.00
16	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	385	1	3.79		23.82
17	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	386	1	3.49		17.27
18	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	387	0	0.00		0.00
19	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	388	0	0.00		0.00
20	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	389	2	3.49		17.27
21	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	390	0	0.00		0.00
22	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	391	10	3.39	S	25.16
23	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	392	1	3.49		17.27

Empty Cells Strings Distance Patterns Standard Deviation 95.4% Functional Dependencies

Figure 4.1: *Motivation example for smells*

In the Figure 4.1 we present a spreadsheet, slightly adapted from one in the EUSES repository, where all kind of smells can be observed. All those smells will be automatically detected and flagged out by our tool "SmellSheet Detective"(Section 5.3).

4.2.1 Statistical Smells

This groups smells that are calculated through some kind of statistical analysis. In this category we only have the *Standard Deviation* smell. The standard deviation smell detects cells that are outside the normal distribution.

Detection

Most of the times when we fill a spreadsheet with numeric values we organize them either by row or column, and many times we introduce wrong values without noticing. So, the **Standard Deviation** smell is detected by analyzing the spreadsheets row (column) by row (column) and flagging the values outside the normal distribution of 95,4% (two Standard deviations). In the detection of this smell neither formulas nor labels are taken into account.

Example

If we look to the Figure 4.1 we can see, for instance, that in the column B the standard deviation is of 2.369E8. Then the normal distribution values acceptable should be within [5.868E8, 1.534E9] and so in the cell B4 we detect a smell because it contains the value 123 that is outside that interval.

4.2.2   Type Smells

In this group of smells we have *Empty Cell* and *Pattern Finder*. They are both in this category because in both of them is made an analyze to the type of cell: Label, Number, Formula or Empty Cells.

- **Empty Cell**

  Many times we forgot to fill some cells in spreadsheets that should be filled. In order to detect some of these cells we created the *Empty Cell* smell. The *Empty Cell* locates all the empty cells in the middle of others non empty cells, this means that if we have an empty line we will not find that line, or if we just have one label in the line we will see it just as a label.

  Detection

  In the detection of the *Empty Cell* what we do is select all the possible windows of cells from each row (column) and verify if in that window there is an empty cell. In the Figure 4.2 we can see in green the cases where the smell is detected. For this smell we used a window of five cells because after looking to many spreadsheets we
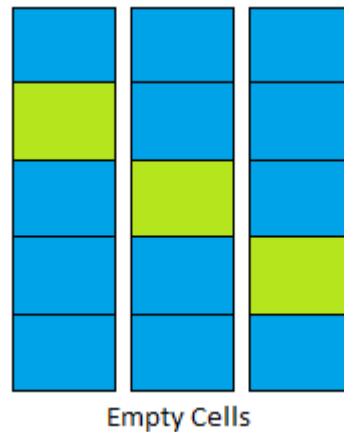
Figure 4.2: *Window of search in Empty Cell Smells*

thought that was an acceptable size, but in order to guarantee that is the best size more tests should be made.

Example 4.2.1

Taking the example spreadsheet from Figure 4.1 we can see that the cells D2 and C6 have empty cells, and they are in the middle of cells with labels fulfilling what we said above, to have at least four cells surrounding it. Still from the same Figure we can see that more white cells exist, for instance in column I there are plenty, but these cells do not fulfill the condition. The analysis for this example is by column, if we look by row all the empty cells from the column I would be detected.

- **Pattern Finder**

  The need of this smell came from the fact that many times we introduce values (numeric or labels) in the middle of formulas to simplify in the moment and then when reusing the spreadsheet we may forget to correct that "problem". In order to detect those cells we created the *Pattern Finder*, but not only for formulas. The *Pattern Finder* finds patterns in the sheet and if in some row we have only Numbers and in the middle of those numbers we find a Label/Empty Cell/Formula we point that cell as a smell.

Detection

The detection of the *Pattern Finder* are quite similar to the one for the *Empty Cell*. In fact they almost overlap each other because in this we also detect empty cells. The major differences between them is that in this smell we detect not only empty cells but also every other kind of cells, and, in this smell we use a smaller window, just four cells. We chose to use a smaller window because the occurrence of patterns without empty cells is quite smaller.

Once again more tests should be done in order to guarantee that this is a good size for the window.
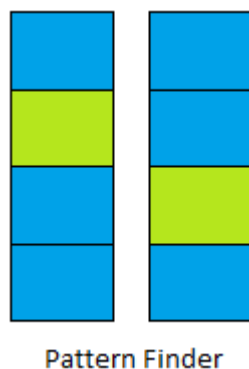
Pattern Finder

Figure 4.3: *Window of search in Pattern Finder Smells*

In the Figure 4.3 we can see in detail how this detection is made.

Example 4.2.2

Taking the example spreadsheet from Figure 4.1 we can see that the cell G3 contains the value "o" so is from the type Label, and this cell is surrounded by Numbers creating a window of four cells like the one shown in Figure 4.3. This means that G3 is a smell, maybe that "o" is a typo and it should be a "0" (zero).

4.2.3   Content Smell

In this category we will have the smells found through the analysis of the content of the cell. We have putted in this category the *String Distance* smell and *Reference to Blank Cells* smell.

- **String Distance**

  When writing in a computer many times we make typo errors, so, to detect those we create the *String Distance* smell. In the *String Distance* smell we compare two strings and find if the minimum number of edits needed to transform one string into the other one.

  Detection

  The detection of this smell is made by using the algorithm created by Vladimir Levenshtein [20] in 1966, *Levenshtein Distance*.

  The *Levenshtein Distance* compares two strings and finds the minimum number of edits needed to transform one string into the other. So in order to do it we have to apply the *Levenshtein Distance* each string from a row (column) to all the others in the same row (column) and verify if the result is 1.

  At first no verification to the strings was being done, and this was a problem because, for instance if we had a spreadsheet with a row with alphabet all cells would be pointed out. So, we limited the comparison only to strings longer than three characters. The election of this value was made by looking into some spreadsheets and selecting the string length producing best results.

  Example 4.3.1

  For *String Distance*, in the Figure 4.1 we can see a example of this smell. In the row C, the word in the cell C8 "SUNLIGHT DISH LIQUIDS" is the plural of the word in the cells C9 to C11 pointing to some probable typing error.

- **Reference to Blank Cells**

  When we have a big spreadsheet with many formulas sometimes having a formula using an empty cell in the calculation may lead to problems in the output, so in order to detect those cells we created the *Reference to Blank Cells*.

  In the *Reference to Blank Cells* smell we identify if there is any formula with references to empty cells.

Detection

The detection of the *Reference to Blank Cells* smell is made by walking all the formulas from the spreadsheet gathering all their references. Then, we just verify if each of the gathered reference is a reference to an empty cell flagging those that are.

Example 4.3.2

For *Reference to Blank Cells*, no example is presented in Figure 4.1 because it displays the values and not the formulas used to compute each values. Thus, let us introduce a new example to illustrate this smell.

SOMA					=(SOMA(B5:E5)/4)*0,4+F5*0,6	
	A	B	C	D	E	F	G
1	Student Name	Ex 1	Ex 2	Ex 3	Ex 4	Exam 1 (%)	Nota Final
2	Ana Rita	102,73	100,00	100,00	100,00	95,80	97,753
3	Anabela Gonçalves	105,00	96,29	100,00	100,00	93,10	95,989
4	Abel Pinheiro	93,18	88,88	100,00	100,00	86,10	89,866
5	Afonso Santos	86,36	74,07	100,00		88,90	=(SOMA(B5:E5)/4)*0,4+F5*(
6	Bruno Ribeiro	82,27	96,29	100,00	100,00	81,90	86,996
7	Catarina Silva	97,73	92,59	100,00	97,85	97,20	97,137
8	Diana Melo	97,73	92,59	100,00	100,00	91,70	94,052
9	Eduardo Lopes	105,00	100,00	100,00	100,00	95,80	97,98
10	Filipe Machado	84,10	100,00	100,00	100,00	90,30	92,59

Figure 4.4: *Reference to Blank Cells Example*

So if we look to the Figure 4.4 we can see that it is in column G where the final grade of the students is calculated, the formula used to calculate the final grade is the one presented in G5 (=(SOMA(B:E)/4)*0,4+F*0,6) and in E5 we have an empty cell. Our "SmellSheet Detective"tool automatically detects this smell.

4.2.4   Functional Dependencies Based Smells

Similarly to what happens in databases, the existence of poor data, can be very costly for the companies who use them. Regarding that and because the problem as already been approached in the data mining field, we tried to adapt techniques used to identify dirty values in databases to spreadsheets.

In [4] it is described a technique to identify dirty values using Conditional Functional Dependencies (CFD), and this is one of the techniques that we will be using. Before we

start analyzing the algorithm let us introduce the concept of Functional Dependency (FD).

> "***Functional Dependency:*** *are fundamental constraints that define the relation between attributes.*" [19]

This means that one attribute A in the relation R only points to another attribute B, A→B. In other words this means that B is functionally dependent upon A, and every time that we have the attribute A it will imply the attribute B.

A CFD is the same as a FD but instead of use all the data, it only uses part of the data.

Birth Country	Nationality
China	Chinese
Spain	Spanish
Portugal	Portuguese
Portugal	Portuguese
Portugal	Portuguese
Portugal	Portuguese

Birth Country	Nationality
China	Chinese
Spain	Spanish
Portugal	Portuguese
Portugal	Greek
Portugal	Portuguese
Portugal	Portuguese

Table 4.2: Functional Dependency Example 1.

Table 4.3: Functional Dependency Example 2.

In the Table 4.2 we can see that we have a relation between the Country of birth and the Nationality, more specifically we can see that when we have a citizen born in "Portugal" it is called as "Portuguese", so we can say that we have the FD *BirthCountry → Nationality*, so we have *Portugal → Portuguese*, *China → Chinese* and *Spain → Spanish* as CFD's. In the Table 4.3 we have *Portugal → Portuguese*, *Portugal → Greek*, *China → Chinese* and *Spain → Spanish* as CFD's.

Next we will see how to apply this in spreadsheets.

Detection

In order to detect this smell some steps have to be taken:

1st Step - "Data collection" The first step towards the detection of this smell is the collection of all the data from the spreadsheet. In this step we gather all the information contained in the all the cells from the spreadsheet. During this step we have to do an extra handling when dealing with formulas cells.

Because formulas are always different from cell to cell, even if they perform the same calculations we have to transform these "absolute" formulas into relative formulas. This

means that, for instance, if we had the cell A11 with the absolute formula SUM(A1:A10) the correspondent relative formula would be SUM(R[-10]C:R[-1]C). With this all the formulas that perform the same operations and with the same range but in different cells they will have the same relative formula (same value for calculations).

2nd Step - "Matching Data" After the data collection we do an attribute lattice match, meaning that we match every column (row) with each other until we have a maximum of four matched columns (rows), depending on how we read the data from the spreadsheet.
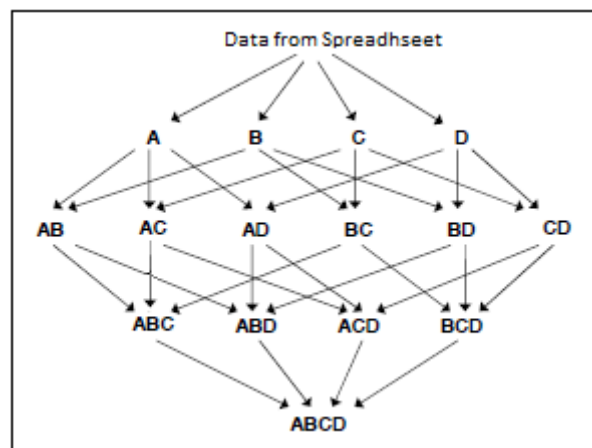
Figure 4.5: *Attribute match lattice.*

To better understand how this match is done we should look tho the Figure 4.5 where each letter represents the name of one column. For instance the data from column A, matched with the data from the column B creates the new group of data AB that contains a subset where the data from both are the same. For instance, if we have A = $\{\{1, 2, 3\}, \{4, 5, 6\}$ and B =$\{\{1, 2\}, \{3\}, \{4, 5, 6\}$ then AB=$\{\{1, 2\}, \{3\}, \{4, 5, 6\}$.

3º Step - "Identify Dirty Values" After matching all the data we find all possible rules of the form $[A, B] \rightarrow C$. In this step we have to make some manual configurations to adjust the quality of the dirty values returned. We have to define the support (θ), that is the minimum number of times that a FD must occur in order to be considered, and the maximum error frequency (α), that unlike the support, is the maximum number of times that a FD can occur in order to be considered a possible problem. These two variables are

going to be used as follow: first we walk all the $x_i \subset X$ and for each one we get the mapped $y_{i's} \subset Y$ that we will use to verify if $\frac{|y_{max}|}{N} \geq \theta$ and $\frac{|y_s|}{N} \leq \alpha$. In here the X and Y are the partitions, for instance for the example in the Figure 4.6 if we have the candidate (X,Y) = ((A,B,C,D),(A,B,C,D,E)), X would be the partition from (A,B,C,D) that in this case is $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ and Y would be the partition of (A,B,C,D,E) that in this case is $\{\{1, 2\}\{3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}\}$. To better understand this you should read the paper [4]. In our tests we always used $\theta = 0.2$ and $\alpha = 0.1$.

4º Step - "Ranking" In the previous step we obtain a list with many dirty values some of them can point to the same cells but using different CFDs, so in this step we sum the number of occurrences for each cell and rate them by frequency. This step is made to help us prioritizing the results.

Example 4.4

	A	B	C	D	E	F	G	H
1	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	102	391		3,39
2	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	102	392	4	3,79
3	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	383	0	0,00
4	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	384	0	0,00
5	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	385	1	3,79
6	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	386	1	3,49
7	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	387	0	0,00
8	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	388	0	0,00
9	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	389	2	3,49
10	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	390	0	0,00
11	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	391	10	3,39
12	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	392	1	3,49

Figure 4.6: *Conditional Functional Dependency Example*

For instance if we take a look to the example in Figure 4.6 we can see that the data from the columns A, B, C and D is all the same. But in the column E we have two different values, "102" and "103". So, for the pink area we have the rule $[A, B, C, D] \rightarrow E_{103}$. This is the y_{max} and happens ten times, so we can calculate the support, $\theta = \frac{10}{12} = 0.8$. For the light blue area we have the rule $[A, B, C, D] \rightarrow E_{102}$, this happens two times and with this we can calculate the frequency, $\alpha = \frac{2}{12} = 0.16(6)$. After these calculations we just have to validate if they are according our reference values, the ones that we present in step three and we found as a smell the cells E1 and E2.

4.2.5 Ochiai Smells

The Ochiai smell is the application of the ochiai algorithm developed by Abreu *et al.* [1]. This algorithm is based in the ochiai similarity coefficient known from the biology domain and was introduced by Abreu *et al.* [1] in the context of fault localization, and previously adapted to spreadsheets by Riboira in the research for the SSaaPP project[2].

Detection

The algorithm adapted by Riboira receives as arguments a list of the cells detected by our smells and the spreadsheet being analyzed and with those it will calculate the probability of error from each cell and its dependents. By dependents we mean the references to other cells. For instance if we have a cell detected that is a formula that uses other cells the calculations of this smell will give ratings to all the cells implied.

Example 4.5

Figure 4.7: *Ochiai Example*

In the Figure 4.7 is presented an example where a pattern smell occurs: in cell C6 the pattern of constant values is broken by a formula (Section 4.2.2). So, applying the ochiai to that cell we have the output presented in the Figure 4.8

For this example where we only give the cell C6 to the ochiai as a smell, the output tell us that the cells A4, A5 and A6 have an 71% rate of being the source of the problem, and the cell C6 and B6 have 100%. This means that the cells in column A are used by the others to compute their values.

---

[2]http://ssaapp.di.uminho.pt/

celulas ochiai: Sheet1!C6
[...\TESTE REF NULLS.xls]
NEW 'Sheet1'!A4 (71%) <<< WARNING!
NEW 'Sheet1'!A5 (71%) <<< WARNING!
NEW 'Sheet1'!A6 (71%) <<< WARNING!
NEW 'Sheet1'!B6 (100%) <<< WARNING!
      'Sheet1'!C6 (100%) <<< WARNING!

Figure 4.8: *Output of ochiai to the spreadsheet in Figure 4.7.*

Chapter 5

Evaluation

Summary:

In this chapter we start by presenting the sample of spreadsheets that we will use in the study and we explain from where and how we select them. Then, we explain the classification model that will be used to classify the smells found. After we present the tool "SmellSheet Detective" and explain how it works, we explain its architecture and how it was built.

In the end we show and explain the results: these results start from the general to the particular, meaning that we show the global results and then we walk each category from the sample used and present results for each selected spreadsheet. In this process we analyze six spreadsheet categories, namely: database, financial, grades, homework, inventory and modeling.

"True genius resides in the capacity for evaluation of uncertain, hazardous, and conflicting information." Winston Churchill

A successful evaluation of the results is always dependent on the data used on it: if we have poor data or an insufficient sample the results and conclusions taken may not be the more accurate. Knowing this, we searched for a big and representative set of spreadsheets to use in our analysis, the EUSES Corpus [17]. This set has already been used by others to perform analysis in spreadsheets [7].

5.1   EUSES Corpus

The EUSES Corpus is a repository built through a Google search by the name of the six major categories (Database, Financial, Grades, Homework, Inventory and Modeling), leaving an initial sample of ≈5600 spreadsheets that after some cleaning process made by the authors (for example, by removing the unusable and the duplicated ones) narrow the sample to ≈4500 spreadsheets. From these 4500 we randomly selected 180 where categories and properties are presented in Table 5.1.

Category	Vertically	Horizontally	Poor Data	Total
Database	42	6	10	60
Financial	20	22	18	60
Grades	13	2	1	15
Homework	15	0	1	16
Inventory	9	1	4	14
Modeling	12	1	2	15

Table 5.1: *Data orientation in spreadsheets.*

More than half of the sample are spreadsheets from financial and database because these two categories were the ones we found more important and with the best data.

5.2   Classification Model

After the selection of the sample to be analyzed we had to define how the results would be measured and classified. For this we inspired ourselves in the technique used by the Software Improvement Group [15] and classified the smells in four levels:

- **Level 1:** In this level fit the cells that are week smells, things that may be a smell;

- **Level 2:** In this level fit all the cells that we are almost certain that are smells but due to the lack of understanding of the sheet we cannot guarantee it as a smell.

- **Level 3:** In this level goes all the cells that we are sure that are smells.

- **Not Smell:** In this category goes all the wrong detections, because the tool did not work properly, or because the detection seem to be on purpose.

The creation of this standard, made possible an uniform classification of the spreadsheets.

5.3 SmellSheet Detective - The Tool

In the analysis of the selected spreadsheet sample we used the tool "SmellSheet Detective"which implements the smells introduced in the Chapter 4. This implementation was made in Java using the Google web toolkit [12] (GWT), the Apache POI [28] library and the Google libraries to work with spreadsheets from the Google Docs.

The use of Google in the application is because the technologies are evolving in the direction of the browser-based approach, and thus we had to keep up and do something according to it. Thus we built this tool that can work directly with a Google docs account [13].
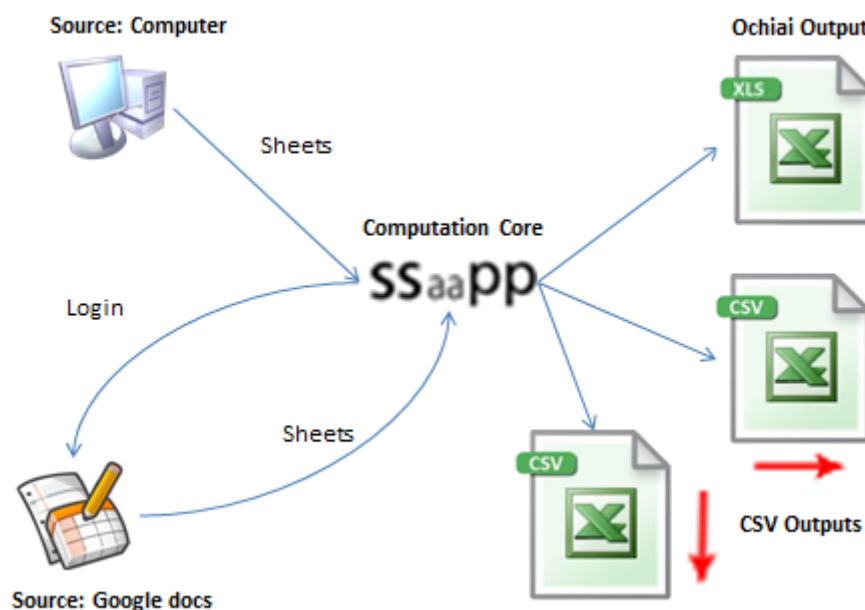
Figure 5.1: *"SmellSheet Detective"Architecture.*

Only having the Google docs as source of spreadsheets give rise to a security issue, because many of the spreadsheets may be confidential and thus we don't want to share them with Google. For instance if one owns a company and wants to analyze the spreadsheets used by it one may feel that is giving secrets away. In order to surpass this problem we also have the possibility to directly analyze spreadsheets by uploading them to the tool

instead of login in Google account. This functionality is just a few steps away of being full integrated.

With this we can describe the tool architecture as we see in Figure 5.1, where we have three major nodes: Spreadsheet source, Computation Core and CSV outputs.

In the Spreadsheet source, as we mentioned before we have the Google docs or direct upload. These two are the source from where the spreadsheets will be analyzed. If we use the Google docs source, we have to login in with our Google account credentials and select which spreadsheet to analyze. If we use the direct upload source, we just have to browse the spreadsheet in our computer and select it.

In the Computation Core is where all the smells defined will be applied to the selected spreadsheet(s).

Finally in the CSV outputs is were the outputs are created. Here we have three outputs, two csv files and one xls file. In the csvs we have the results for each smell by analyzing the data in two different ways, vertically and horizontally. In the xls file is where it will be presented the results of the ochiai analysis mentioned in Chapter 4.

5.4   Results

In the representation of the results instead of full spreadsheets, we selected ten sheets per category. This selection was made randomly. In some cases and since we were choosing the sheets randomly, we got empty sheets. In these cases we choosed another sheet to replace it. This happened just a few times because most of the spreadsheets have data; even if in some of them there was few data in the sheets or if they looked like a word document they had something to analyze. During the analysis of spreadsheets we realize that the organization of the data could be done in two ways: either we can put the data organized horizontally or vertically. This means that if the data was organized horizontally the values would be co-related column by column. If we look to the Figure 5.2 we see a financial spreadsheet where the expenses are presented for many years, putting these years side by side. So the relation that makes sense to evaluate is between those values side by side.

In the Figure 5.3 we see a grade spreadsheet that unlike what happens to the Figure 5.2 the values that make sense to relate are the grades for each problem. So in this case we have a vertical organization.

	2000	2001	2002
Cash & Short-Term Investments	18,03	37,563	38,532
Net Patient Receivables	56,358	57,109	63,122
Current Assets	89,556	110,749	117,957
Net Fixed Assets	92,695	91,922	96,831
Accumulated Depreciation	134,611	148,063	161,941
Total Assets	312,096	342,919	339,828
Current Portion of Long Term Debt	5,196	6,115	6,484

Figure 5.2: *Horizontal Organization Data Example. (Adapted from FIN_hospitaldataset2002 spreadsheet).*

ID	Prob#1	Prob#2	Prob#3	Prob#4	Exam#1	Exam#2	Final Exan	Final %
3427	9	15	17	5	75	97	80	84,52941
7925	9	14	21	5	102	88	92	94,11176
4577	9	13	19	5	77	80	67	76,22941
3427	10	15	19	5	82	90	84	86,81176

Figure 5.3: *Vertical Organization Data Example. (Adapted from GRD_483_grades_web spreadsheet).*

Because of these two types of organization we decided to classify the sample used. To do so, we opened all the spreadsheets one by one and that make us realize that most of the spreadsheets had poor data in it: some were empty forms and others some sort of manuals or catalogs were the data was mostly labels, so, we decided to identify some of these but keeping a very permissive criteria because in spite we thought the data was poor, it was still possible to analyze them. This lead to the fact that many of the spreadsheets characterized as horizontal/vertical still hadn't relevant data. The numbers of this classification can be seen in the Table 5.1.

Next we will see the results for each of the six categories from the EUSES corpus. During the presentation of the results first we will see the individual outputs for each sheet and highlight some of the more relevant results. In the presentation of the results we will hide the smells that had no cells flagged.

Then we will discuss the global results where we will group the results of all sheets and make a statistical analysis to them. This statistical analysis will be done by level, and smell and the values presented will always be an approximation because they will remain accurate enough and will simplify the reading.

5.4.1   Database Sheets

With 60 spreadsheets the *Database* category is one of the two biggest samples that we have used. Because this category mostly included spreadsheets similar to databases, most of the cells were numeric and label values. The caption of each table is the name of the sheet with the following format: *DB_spreadsheet name sheet name*. In the next sections the tables are named likewise, just changing the first letters to meet the corresponding category. Also, in the tables when the values occur followed by "matches" means that instead of number of cells are the number of combinations founded, this only happens in the string distance smell because the smell compares the cells two by two.

Results

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Std. Dev. Cells	9	0	0	1
F.D. Cells	2	1	0	0

Table 5.2: DB_basicdata HAFO Permits.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	2	0	0	21
Patterns	2	0	0	38

Table 5.3: DB_cattrainchecklist Sheet1.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Std. Dev. Cells	0	0	0	1
String Dist.	0	0	0	6 Matches

Table 5.4: DB_duck94_otherdata RHBSept.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	2	0	0	0
Patterns	2	0	0	0
Std. Dev. Cells	1	0	0	0
String Dist.	0	0	0	6 Matches

Table 5.5: DB_SteamTool 3. STEAM SYSTEM PROFILING.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	5	0	0	9
Patterns	5	0	0	9
F.D. Cells	2	4	1	0

Table 5.6: DB_NamingConventionDataS#A855B Summary.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	1	0	0
Patterns	0	1	0	0
String Dist.	0	0	0	6 Matches

Table 5.7: DB_rmomatrix MATRIX.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	15
Patterns	0	0	0	29
Std. Dev. Cells	0	0	0	18
F.D. Cells	0	2	0	0

Table 5.8: DB_ADC%20Databases ADC_data_Table1.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	1
Patterns	0	0	0	2
Std. Dev. Cells	0	0	1	3

Table 5.9: DB_Population Pop.l1.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
String Dist.	0	0	0	6 Matches

Table 5.10: DB_dab1 Sheet3.

Besides the results showed above, we also analyzed the sheet *"DB_2004_admin_plan Goal C"* but because in this sheet all the results obtained were zero we did not create a table for it.

From the results above, some got our attention. For instance, in Table 5.4 and Table 5.10 none of the detected cells were considered smells, making these sheets the ones with worst percentage results in the category. On the other hand, Table 5.2 with ≈85% of the cells flagged level 1 and ≈8% as level 2 is where we find the best percentage results for the category.

Another thing we must notice is that in spite of for Table 5.4 the percentage results are the worst, the Table 5.3 and Table 5.8 the number of not smells is substantially higher.

Discussion

Now we will see the general results for the *Database* category and discuss some of the more relevant findings.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell	Total
Empty Cells	7	1	0	45	53
Patterns	7	1	0	78	86
Std. Dev. Cells	10	0	0	23	33
String Dist.	0	0	1	24	25
F.D. Cells	4	7	1	0	12
Ref2null	0	0	0	0	0
Total:	28	9	2	170	209

Table 5.11: Database Result Totals.

In the Table 5.11 is presented the total values for each smell and the total values by level from all the sheets seen previously.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	13%	2%	0%	85%
Patterns	8%	1%	0%	91%
Std. Dev. Cells	30%	0 %	0%	70%
String Diff.	0%	0%	4%	96%
F.D. Cells	33%	58%	8%	0%
Ref2null	0%	0 %	0%	0%
Total:	13%	4%	1%	81%

Table 5.12: Database Statistical Result by Smell.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	25%	11%	0%	26%
Patterns	25%	11%	0%	46%
Std. Dev. Cells	36%	0 %	0%	14%
String Diff.	0%	0%	50%	14%
F.D. Cells	14%	78%	50%	0%
Ref2null	0%	0 %	0%	0%

Table 5.13: Database Statistical Result by Level.

Observing the Tables 5.11, 5.12 and 5.13 the first thing that we notice is that no references to empty cells have been found. This happens because in this category, from the ten selected sheets, the data was mainly label and numeric values, not leaving room for this smell.

Another thing we can see in the tables is that the patterns and the empty cells are the smells that happen more often and they have some overlap results. This happens because these two smells have much in common. In fact the patterns will found almost all the results that the empty cells do, and some more.

Still from the tables we can see that for the functional dependencies smell none of the findings was considered not a smell. This is easy to understand why: like we have talked in Section 4.2.4 the algorithm used to detect these smells was adapted from the databases, and being this category the databases category was expected that the results were well behaved.

Also if we look to Table 5.12 we can see that most of the detected cells (≈81%) were not considerate as smells, but we must consider that from these ≈81% of cells ≈72% were either empty cells or patterns that were almost overlapped smells. This happens so much because in this category many sheets were really like databases where null values are allowed and in the spreadsheets were represented by empty cells. Still from these ≈81% cells ≈14% where string differences where the difference was in a numeric value.

5.4.2   Financial Sheets

Like the *Database* category the *Financial* is the other big sample with a total of 60 spreadsheets. But unlike the *Database* this category is more complex in terms of types of data having a little bit of all cell types (Formulas, Constants and Labels).

Results

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	4
Patterns	2	0	0	0
String Diff.	0	0	0	7 Matches
Ref2null	6	0	0	0

Table 5.14: FIN_CMSAauditreport2002_2003 Sheet1.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	2	0	0
Patterns	0	2	0	0
Std. Dev. Cells	0	0	0	18
String Diff.	0	0	0	3 Matches
F.D. Cells	0	0	3	0
Ref2null	0	0	4	0

Table 5.15: FIN_FinalAnnexFSSN06001 Annex3.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	2
Patterns	0	0	0	2
Ref2null	8	0	0	0

Table 5.16: FIN_Cost%20Statement Blad1 (2).

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	3	0	0	6
Patterns	4	0	0	30
Std. Dev. Cells	0	0	0	1
String Diff.	0	0	0	3 Matches

Table 5.17: FIN_Financial%20Compariso#A7ED8 Financial Comparison Analysis.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Std. Dev. Cells	2	0	0	5
Ref2null	0	0	1	6

Table 5.18: FIN_finrpt00 Five Year Review.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	6
Patterns	0	0	0	6
Std. Dev. Cells	0	0	0	9

Table 5.19: FIN_income-statement Income Statement.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Ref2null	1	1	0	0

Table 5.20: FIN_MATHCOUNTS%20Financial Sheet1.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	4
Patterns	0	0	0	16

Table 5.21: FIN_hospitaldataset2002 MEMORIAL.

Beside the sheets presented above we also analyzed the sheets *"FIN_clienttemplate Financial Diagnostics"* and *"FIN_financial-greece_el GR"*, but since the results obtained were all zero no tables were created.

From the result tables presented we can notice that, for instance, for Table 5.19 and Table 5.21 all the flagged cells were considered not smells making them the sheets with the worst results from the sample. On the other hand Table 5.20 all the cells were correctly flagged and therefore this is the sheet with the best results.

Other result that we must highlight is the pattern smell in Table 5.17 where 30 cells were categorized as not smell. This value is quite high specially if we compare it to the rest of the values.

We also must underline the standard deviation smell in Table 5.15 where we also obtained a quite high value again, specially if compared to the other values.

Discussion

Now we will see the general results obtained in the *Financial* category and point some interesting facts that can be extracted from them.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell	Total
Empty Cells	3	2	0	22	27
Patterns	6	2	0	54	62
Std. Dev. Cells	2	0	0	33	35
String Diff.	0	0	0	13	13
F.D. Cells	0	0	3	0	3
Ref2null	9	1	7	6	23
Total:	20	5	10	128	163

Table 5.22: Financial Result Totals.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	11%	7%	0%	81%
Patterns	10%	3%	0%	87%
Std. Dev. Cells	6%	0 %	0%	94%
String Diff.	0%	0%	0%	100%
F.D. Cells	0%	0%	100%	0%
Ref2null	39%	4 %	30%	26%
Total:	12%	3%	6%	79%

Table 5.23: Financial Statistical Result by Smell.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	15%	40%	0%	17%
Patterns	30%	40%	0%	42%
Std. Dev. Cells	10%	0 %	0%	26%
String Diff.	0%	0%	0%	10%
F.D. Cells	0%	0%	30%	0%
Ref2null	45%	20 %	70%	5%

Table 5.24: Financial Statistical Result by Level.

Unlike what happens in the previous category in which we did not find one type of smells, in the *Financial* category we have results for every type of smell. Still, by observing Table 5.22, Table 5.23 and Table 5.24 there are some facts that can be observed. One thing that can be easily observed from the tables is that in the string distance smell all the flagged matches were considered not smells. In spite of the detected smells were all wrong, they can be easily corrected because all of them were either numeric values with a dollar mark associated or index labels where the difference was in the number of the index.

Another thing that we can see is that unlike the string distance smell, in the functional dependencies smell all the flagged cells are level 3. This may lead to conclude that this smell works very well for this category but because all the cells came from the same sheet (Table 5.15) we just can say that for that sheet this smell worked perfectly. In fact in this

sheet it helped to identify some cells that just by looking were really hard to detect.

Another smell that seems to behave nicely is the reference to empty cells where only ≈26% of the detected cells were considered not smell. One of the things that lead to this is that in this category there are used many ranges and some of them go out of boundaries. Also, many times they make use of empty cells that should be filled. Another thing is the existence of some sheets that are forms to fill with data.

We can also notice that the patterns represent ≈42% of the not smells being the highest value. One explanation for this is that most of the sheets have calculations in the middle of the data creating blocks of cells with formulas mixed with values.

5.4.3 Grades Sheets

The *Grades* category is built by a set of 15 spreadsheets where, like the name suggests, most of them are used to grade/classify something. Therefore the data in it has a little bit of every type of cells, but the main percentage of cells are numeric values.

Results

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	1	0	0	0
Patterns	1	0	0	1
Std. Dev. Cells	1	0	0	7

Table 5.25: GRD_483_grades_web grades.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	9
Patterns	0	0	0	9
Std. Dev. Cells	1	0	0	4

Table 5.26: GRD_as474gradestopost lab grades.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Std. Dev. Cells	0	0	0	97
String Diff.	0	0	0	14 Matches
F.D. Cells	0	0	0	81

Table 5.27: GRD_2003FP785DZ 2003FP785DZ.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	19
Patterns	0	0	0	26
String Diff.	0	0	3	1319 Matches
Ref2null	36	0	0	0

Table 5.28: GRD_dss-2001 Sheet1.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	2
String Diff.	0	0	3	3 Matches
F.D. Cells	0	0	1	0
Ref2null	0	0	0	18

Table 5.29: GRD_firsttrimester Sheet1.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
String Diff.	0	0	0	15 Matches

Table 5.30: GRD_99execgrades Exec Grades.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	14	0	0	0
Patterns	15	0	0	0
Std. Dev. Cells	0	0	0	4
F.D. Cells	19	0	0	0
Ref2null	0	8	0	0

Table 5.31: GRD_CRJ%20230_Spring%20grades ClassRosterExportServlet.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	1	0	0
Patterns	0	1	0	0
Std. Dev. Cells	0	0	0	12
String Diff.	0	0	0	1 Match
F.D. Cells	0	0	0	8
Ref2null	0	0	1	0

Table 5.32: GRD_anat1f03post post1.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	9
Patterns	0	5	0	9
Std. Dev. Cells	0	0	0	8
String Diff.	0	0	1 Match	1 Match
F.D. Cells	0	5	0	0

Table 5.33: GRD_2000_places_School Sheet1.

Besides the sheets presented above we also analyzed the sheet *"GRD_Engineering Engines"*, but since the results obtained were all zero no table was created.

Looking to the results presented, there are some values that must be highlighted. The first one appear in Table 5.28 in the string distance smell: we got 1319 matches that were

not considered as smells. This was the highest value found in all the results from all the categories.

Other thing we must notice is that in Table 5.27 and Table 5.30 all the flagged cells were considered not smells, making of these two sheets the ones with worst results from the category. Unlike these two sheets, the Table 5.31 is the sheet with best results for this category. With only ≈7% of the cells (standard deviation) flagged not being considered as smell.

Discussion

Now we will see the general results obtained in the *Grades* category and point some interesting facts that can be extracted from them.

Level / Smell	Level 1	Level 2	Level 3	Not Smell	Total
Empty Cells	15	1	0	40	56
Patterns	16	5	0	45	66
Std. Dev. Cells	2	0	0	132	134
String Dist.	0	0	4	1353	1357
F.D. Cells	55	5	1	89	150
Ref2null	0	8	1	18	27
Total:	88	19	6	1677	1790

Table 5.34: Grades Result Totals.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	27%	2%	0%	71%
Patterns	24%	8%	0%	68%
Std. Dev. Cells	1%	0 %	0%	99%
String Dist.	0%	0%	0%	100%
F.D. Cells	37%	3%	1%	59%
Ref2null	0%	30%	4%	67%
Total:	5%	1%	0%	94%

Table 5.35: Grades Statistical Result by Smell.

Level Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	17%	5%	0%	2%
Patterns	18%	26%	0%	3%
Std. Dev. Cells	2%	0 %	0%	8%
String Dist.	0%	0%	67%	81%
F.D. Cells	63%	26%	17%	5%
Ref2null	0%	42 %	17%	1%

Table 5.36: Grades Statistical Result by Level.

Once again, in this category all the smells have been found in the sample. If we pay attention to Table 5.34, Table 5.35 and Table 5.36 there are some values that we must look into. The first one is the high number of flagged matches in the string distance smell: with 1353 matches it represents ≈100% of the results for that smell. This happens due to the results already highlighted above where we present the results for this category. The explanation for this value is simple and manageable because from these 1353 matches, 1347 were comparisons between numeric values read as strings. From all the 1353 detected, only one is for sure a false detection. This problem was in the implementation of the smell. For instance, when he verifies if is a numeric value if the value starts with empty space the value would be read as a string.

Like the string distance, the standard deviation also has very poor results in this category, with ≈99% of the cells detected not being considered as smells. We also have an explanation for this. Because the grades of a student can be almost random, some go from 0 to 100, and thus the really bad or really good scores will be flagged as out of the standard and so from all the cells we detect we only gave importance to those that were 0 or out of the range considering every other cell as correct.

At last, we must notice once again the close results obtained by the empty cells and the patterns.

5.4.4   Homework Sheets

The *Homework* category is built by a set of 16 spreadsheets that were some sort of home-works. The spreadsheets in this category were so distinct that some of them could easily be placed in some of the other categories: one could fit in *Grades*, other that could fit in *Database* and even in *Financial*.

Results

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	3
Patterns	0	0	0	3
Std. Dev. Cells	5	0	0	0
F.D. Cells	1	0	0	0

Table 5.37: HOME_finalGRADES Writing Assng.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	2
Patterns	0	0	0	2
String Dist.	0	0	0	93

Table 5.38: HOME_AClassSchedule2003 Schedule.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	7
Patterns	0	0	0	12
String Dist.	0	0	0	100 Matches
F.D. Cells	0	0	0	1

Table 5.39: HOME_D6 D6.2.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	27
Patterns	0	0	0	28

Table 5.40: HOME_comments02 Sheet2.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
String Dist.	1 Match	0	0	7 Matches

Table 5.41: HOME_2101_Homework 2101.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
String Dist.	0	0	0	23 Matches

Table 5.42: HOME_cgs1540 Sheet1.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Patterns	0	0	0	9
Std. Dev. Cells	0	0	0	8

Table 5.43: HOME_Econ%20homework%20one Sheet1.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	1	0	0	1
Patterns	1	1	0	1
Std. Dev. Cells	0	0	0	1

Table 5.44: HOME_Fin_Eval-Budgets-Web 1997.

Level Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	1
Patterns	0	0	0	1
String Dist.	0	0	0	7 Matches

Table 5.45: HOME_cis105Winter2004calendar January 2004.

Like in the previous categories we also got one sheet where the results were all zero, the *"HOME_embaform statement of corporate support"*, and therefore no table was created.

Looking to the tables above there are some facts that we must highlight. In Tables 5.38, 5.39, 5.40, 5.42, 5.43 and 5.45 all the detected cells were considered as not smell.

We also can see that the best percentage result obtained in this category is for the Table 5.37 and Table 5.44 with only ≈50% of the smells being correctly accepted as ones.

Discussion

Now we will see the general results obtained in the *Homework* category and point some interesting facts that can be extracted from them.

Level Smell	Level 1	Level 2	Level 3	Not Smell	Total
Empty Cells	1	0	0	41	42
Patterns	1	1	0	56	58
Std. Dev. Cells	5	0	0	9	14
String Dist.	1	0	0	230	231
F.D. Cells	1	0	0	1	2
Ref2null	0	0	0	0	0
Total:	9	1	0	337	347

Table 5.46: Homework Result Totals.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	2%	0%	0%	98%
Patterns	2%	2%	0%	97%
Std. Dev. Cells	36%	0 %	0%	64%
String Dist.	0%	0%	0%	100%
F.D. Cells	50%	0%	0%	50%
Ref2null	0%	0 %	0%	0%
Total:	3%	0%	0%	97%

Table 5.47: Homework Statistical Result by Smell.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	11%	0%	0%	12%
Patterns	11%	100%	0%	17%
Std. Dev. Cells	56%	0 %	0%	3%
String Dist.	11%	0%	0%	68%
F.D. Cells	11%	0%	0%	0%
Ref2null	0%	0 %	0%	0%

Table 5.48: Homework Statistical Result by Level.

Like in the *Database* category we also did not obtain any results for the reference of empty cells smell. In spite of in this category more formulas were used than in the *Database* their use was almost cell by cell and no ranges not leaving much room for mistakes at this level.

Other thing that we can easily see from the tables is that we got really high rate of not smells, ≈97% of the cells. And, from these ≈97%, the string distance smell is the one with the highest value (≈68%). This happens because like we have showed previously, the comparison between numeric values is a problem for this smell.

We must also notice once again the close results obtained by the empty cells and the patterns.

5.4.5   Inventory Sheets

Now we will present the results from *Inventory* category. With 14 spreadsheets this is the smaller category evaluated. The data in the spreadsheets from this category is quite similar to the one found in the *Database* category, being mainly a list of items with descriptions and so almost only constants are used.

Results

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	1	0	0	0
Patterns	1	0	0	0
F.D. Cells	1	1	0	0

Table 5.49: INV_1996El_Final_Files WRAP Domain.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Std. Dev. Cells	0	0	0	2

Table 5.50: INV_Inventory%20Log%20Sheet Basement.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Std. Dev. Cells	0	0	0	5

Table 5.51: INV_AssetAccountCodes ACCT Asset.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	29
Patterns	0	0	0	29
String Dist.	0	0	0	101 Matches

Table 5.52: INV_outline TOC for Pam.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	2
Patterns	0	0	0	2
String Dist.	0	0	0	2 Matches

Table 5.53: INV_InsuranceApplication-#A8A10 Page 4.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
String Dist.	0	0	0	3 Matches
F.D. Cells	0	0	0	5

Table 5.54: INV_Licensing%20Inventory#A88C0 Purchase Data.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Patterns	0	0	0	1
String Dist.	0	0	1 Match	3 Matches

Table 5.55: INV_PrimaryProduction2003 Dec.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	71
Patterns	0	0	0	72
Std. Dev. Cells	0	0	0	24
String Dist.	0	0	1 Match	3 Matches
F.D. Cells	0	0	0	4

Table 5.56: INV_ICATINV iccat tag.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Std. Dev. Cells	1	0	0	0
String Dist.	1 Match	0	0	43 Matches
F.D. Cells	0	0	0	2

Table 5.57: INV_2003-fairact II. Inherently Governmental.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	6
Patterns	0	0	0	6

Table 5.58: INV_CL2003-007_AnnexB ANNEX B.

For the first time from the sample selected we did not get any sheet with no results. Still in the Table 5.50, Table 5.52, Table 5.53, Table 5.54, Table 5.51, Table 5.58 all of the flagged cells were considered not smells making them the ones with worst percentage results. From those five Table 5.52 is the one that presents the worstest scenario.

On the other hand, in this category we also have a sheet where all the cells were considered as smell, Table 5.49 with three smells level 1 and one level 2.

Discussion

Now we will see the general results obtained in the *Inventory* category and point some interesting facts that can be extracted from them.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell	Total
Empty Cells	1	0	0	108	109
Patterns	1	0	0	110	111
Std. Dev. Cells	1	0	0	31	32
String Dist.	1	0	2	155	158
F.D. Cells	1	1	0	11	13
Ref2null	0	0	0	0	0
Total:	5	1	2	415	423

Table 5.59: Inventory Result Totals.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	1%	0%	0%	99%
Patterns	1%	0%	0%	99%
Std. Dev. Cells	3%	0 %	0%	97%
String Dist.	1%	0%	1%	98%
F.D. Cells	8%	8%	0%	85%
Ref2null	0%	0 %	0%	0%
Total:	1%	0%	0%	98%

Table 5.60: Inventory Statistical Result by Smell.

Smell \ Level	Level 1	Level 2	Level 3	Not Smell
Empty Cells	20%	0%	0%	26%
Patterns	20%	0%	0%	27%
Std. Dev. Cells	20%	0 %	0%	7%
String Dist.	20%	0%	100%	37%
F.D. Cells	20%	100%	0%	3%
Ref2null	0%	0 %	0%	0%

Table 5.61: Inventory Statistical Result by Level.

The general conclusions about this category are quite similar to the ones taken in the *Homework* category. We did not obtain any results for the reference of empty cells smell because once again the data in here was almost only constants and therefore without formulas we can not have this smell.

Like in *Homework*, in this category we also got a high rate of not smells, with ≈98% is even higher. But, instead of the fault being only from the string distance smell like in the *Homework*, the empty cells and the patters also gave a big help. These three smells had high rates because of two reasons: 1) the string distance once again had the problem with comparison between numeric values; 2) In the empty cells and patterns the problem was the same because almost all of the detected cells overlap each other, and the problem was that being this category similar to the *Database*, many of the cells make sense to be empty.

We must also notice once again the close results obtained by the empty cells and the patterns.

5.4.6 Modeling Sheets

The final category analyzed was *Modeling*. With a total of 15 spreadsheets from this category the data present in them was mostly constants but there were one or two sheets with formulas, even if very simple formulas.

Results

Level Smell	Level 1	Level 2	Level 3	Not Smell
Std. Dev. Cells	1	0	0	2
F.D. Cells	3	0	0	0

Table 5.62: MOD_skill-certificates-071103 West.

Level Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	13	0	0	0
Patterns	14	0	0	3
String Dist.	9 Matches	1 Match	0	2 Matches

Table 5.63: MOD_Analytic_work Sheet 1.

Level Smell	Level 1	Level 2	Level 3	Not Smell
String Dist.	0	0	0	7 Matches
F.D. Cells	0	0	1	0

Table 5.64: MOD_rs2002-0152att CALFED Watershed.

Level Smell	Level 1	Level 2	Level 3	Not Smell
String Dist.	0	1 Match	0	563 Matches

Table 5.65: MOD_PSCCUNYawards Sheet1.

Level Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	0	0	0	18
Patterns	0	0	0	18
String Dist.	0	0	0	37 Matches

Table 5.66: MOD_IROS2003-Program-Final Sessions.

Level Smell	Level 1	Level 2	Level 3	Not Smell
String Dist.	1 Match	0	0	19 Matches

Table 5.67: MOD_Teaching%20Evaluation#A8732 Sheet1.

Level \ Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	75	0	0	0
Patterns	79	0	0	0
Std. Dev. Cells	0	0	0	4
String Dist.	1 Match	0	0	17 Matches
F.D. Cells	0	0	0	20

Table 5.68: MOD_CancelsFullstOct02 CancelsFullLstOct02.

The results obtained for the sheets *"MOD_Draft_SIP_Timeline Summary & Timeline"*, *"MOD_Template%20-%20Config#A7D35 Types"* and *"MOD_alfabetischelijst Sheet1"* are all zero.

In this category we can see that in Table 5.66 all the flagged cells were considered not smell, so, this is the sheet where we found the worst results. Other value to highlight is the string distance in the Table 5.65, with 563 matches it is the highest value from this category.

We also got some good results, for instance in Table 5.63 where ≈88% of the cells were considered as smells, or even in Table 5.68 with ≈79%.

Discussion

Now we will see the general results obtained in the *Modeling* category and point some interesting facts that can be extracted from them.

Level \ Smell	Level 1	Level 2	Level 3	Not Smell	Total
Empty Cells	88	0	0	18	106
Patterns	93	0	0	21	114
Std. Dev. Cells	1	0	0	6	7
String Dist.	11	2	0	645	658
F.D. Cells	3	0	1	20	24
Ref2null	0	0	0	0	0
Total:	196	2	1	710	909

Table 5.69: Modeling Result Totals.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	83%	0%	0%	17%
Patterns	82%	0%	0%	18%
Std. Dev. Cells	14%	0 %	0%	86%
String Dist.	2%	0%	0%	98%
F.D. Cells	13%	0%	4%	83%
Ref2null	0%	0%	0%	0%
Total:	22%	0%	0%	78%

Table 5.70: Modeling Statistical Result Totals by Smell.

Level / Smell	Level 1	Level 2	Level 3	Not Smell
Empty Cells	45%	0%	0%	3%
Patterns	47%	0%	0%	3%
Std. Dev. Cells	1%	0 %	0%	1%
String Dist.	6%	100%	0%	91%
F.D. Cells	2%	0%	100%	3%
Ref2null	0%	0 %	0%	0%

Table 5.71: Modeling Statistical Result Totals by Level.

Like in the rest of the categories the *Modeling* category also got an highest rate of not smells, with ≈78% of not smells. But in this category this happen due the high value of not smells in the string distance that got ≈91% of not smells leaving all the others with less than 4%. This high result in this smell once again comes from the match with numeric values.

Other thing that we can see is that in the empty cells and patterns, besides the fact that they still have close results, for the first time these smells got lowest rate of not smells, with less than ≈18%.

We can also see that once again we got no results for the references to empty cells. This happen because like in the Homework category the few formulas considered are used without ranges making it harder to miss.

If we look to Table 5.69 we can see that the functional dependencies smell got all their bad results from Table 5.68 and this was because the kind of data in the sheet was not good for the algorithm because it did not seem to have dependent values.

5.4.7   Global Discussion

The exhaustive analysis made to these six categories and because we opened all the sheets and analyze them all one by one, some general and interesting facts must be highlighted. In terms of the results, the first thing we must say is that by the fact of the empty cells and the patterns smell being calculated in a very similar way, the results of one most of the times overlap the other. Other thing about the results is the fact that the sample used being gathered from the Internet and we can't ask the authors of the spreadsheets what they meant to do, or if what is done is correct or not, so many of the not smell were also because we did not know the domain of the sheets and we had no one to ask.

Level Smell	Level 1	Level 2	Level 3	Not Smell	Total
Empty Cells	115	4	0	274	393
Patterns	124	9	0	364	497
Std. Dev. Cells	21	0	0	234	255
String Dist.	13	2	7	2420	2442
F.D. Cells	64	13	6	121	204
Ref2null	9	9	8	24	50
Total:	346	37	21	3437	3841

Table 5.72: Global Result Totals.

From the results presented in Table 5.72 we can say that some results may be improved. For instance, the string distance smell that is the one with more wrong detections, from the 2420 wrongly detected smells, with an improvement on the string distance algorithm as discussed on Section 6.1 we believe that 2134 of those wrongly detected smells will not be considered so. This would lead to an improvement in this smell of $\approx 88\%$. The other smell where the results would be improved is the standard deviation smell, in this one we do not have a percentage of improvement because most of the wrong detections was due the lack of knowledge of the domain of the spreadsheet.

This analysis also showed some problems with the current implementation of the "Smell-Sheet Detective"tool. The first one was that whenever two or more cells were merged our tool just notices the existence of one leaving the others empty. Imagine that you merge A1, A2 and A3 creating a larger cell where you put the input "10", for our tool it was like A1 had input of "10" and the rest (A2 and A3) were empty. To solve this we would have to somehow identify these merged cells, and put in each of them a special mark in order to identify them during the analysis.

Another problem detected on the tool was related the high number o matches found in the string distance smell. Because we use the Levenshtein distance algorithm to find close strings and this algorithm is thought specially for strings, most of the wrong string distance detected came from the use of the algorithm in numeric strings or strings with numeric values. One solution for this problem would be the identification of the character where the strings were different and if that character was a numeric value we would ignore it.

Finally, another problem that we were able to detect, was that in some sheets many cells were the sum of others but with the result inserted by hand and not using a formula. This happens for instance in the sheet *"FIN_hospitaldataset2002 MEMORIAL"* on the cell F22, this is an interesting feature that we should add to our tool. This has happened at least in three of the sheets analyzed, being the example the one with more cases. Three times may not be a high number of occurrences, but since the analysis made was not to find those cases we believe that if we search properly more cases would be found. Also, this proposed feature is just a theoretical proposal that should be studied properly by performing some tests and analyzing the results obtained.

Rate / Smell	Usable	Not Usable	Neutral
Empty Cells	Modeling	Database; Grades; Financial; Homework; Inventory	
Patterns	Modeling	Database; Grades; Financial; Homework; Inventory	
Std. Dev.	Financial(**)	Database; Grades; Modeling; Homework; Inventory	
String Dist.(*)	Database; Grades; Financial; Homework; Inventory; Modeling		
F.D. Cells	Database; Grades; Financial; Modeling	Homework; Inventory	
Ref2null	Grades; Financial; Homework; Modeling		Database; Inventory

Table 5.73: Where to use or not the smells.

In the Table 5.73 we can see in which category each smell should be used.

(*) The string distance smell in all the categories must be improved in order to be used. These improvements will be mentioned in the Chapter 6.

(**) The standard deviation smell for the *Financial* category in order to give meaningful results must also be improved with the help of the Layout Smell that will be introduce in Chapter 6.

In the *Database* category we can say that the empty cell and the patterns smell should not be used because due the structure of the sheet they will flag many false smells. The reference to blank cells because in this type of sheets we have no formulas we may or may not use it because the result will always be null. On the other hand, the smell that can be used is the functional dependencies and the string distance smell.

In the *Financial* category we should not use the the empty cell and the patterns smell, but the rest of the smells may be used.

In the *Grades* category we should not use the empty cell, the patterns, and the standard deviation smell because due the type of data they get bad results. On the other hand the string distance, the functional dependencies and the references to blank cells may be used.

The reference to blank cells in spite of the results obtained being just ≈34% of true smells, this happens because one of the sheets analyzed (Table 5.29) was an empty form to fill and the total (≈67%) of the not smells cam from that sheet.

In the *Homework* category the the type of sheet is so distinct that for some smells their use will depend on the sheet. For instance, the functional dependencies smell we have some sheets that perform well and others that do not. On the other hand the string distance and the reference to blank cells may be used safely.

The *Modeling* category once again we have a smell with bad results but that we chose to classify as usable, like the in the grades for the reference to blank cells in this category the not smells obtained for the functional dependencies smell came all from the same sheet. On this category the only smell that we should not use is the standard deviation smell.

Finally the *Inventory* category. In this one the only smells that should be used are the string distance and the reference to blank cells.

Chapter 6

Conclusion

This thesis was mostly directed to the assembly of a technique that allowed us to perform a good and reliable analysis of spreadsheets. Thus, all the steps needed to construct such a technique and corresponding tool, "SmellSheet Detective"and results obtained by it are shown.

The main goal was the creation of smells for spreadsheets. As we can see from the Chapter 4 this was successfully accomplished because we created a catalog with seven distinct smells, even if in the case of the pattern smell and the empty cell smell they were very similar.

Because the spreadsheets used in the analysis were product of a Google search, some doubts about their quality arise. This happens because, if we think clearly, we know that companies have secrets that need to be protected and so, it would be likely that the important and more complex spreadsheets used by these companies in their daily work were not available for the general public. Thus we had to know how good were the spreadsheets in EUSES in order to rely (or not) on the results obtained. In spite of the sample selected (~200 spreadsheets) may not be the perfect sample, still it should give some accurate indicators of the status of the EUSES Corpus. From the work done in order to build Table 5.1 in Chap. 5 I can say that in the category of financial the average of spreadsheets is quite good and complex enough for future analysis, the category of database is also a good representative of databases on spreadsheets.

Still about the spreadsheets used we must say that in the financial category an interest finding was made: the fact of many spreadsheets in this category had the data organized in different way, instead of a vertical organization they had the data organized horizontally.

This happens because in the financial category often there is a reference to many years, and to have a reference of the values and to see the history of the companies they place the values of each year side by side.

After this my final thoughts about the EUSES are that in spite of it still can be used for some tests, it needs a proper cleaning in order to become a more reliable set of spreadsheets.

Other of the proposed goals was about the development of a tool.

About this we must say that in spite of the fact the current status of the tool is acceptable, during the development there was a brainstorm of ideas and new possible features were considered to enhance the results. Thus, we must say that the tool is not finished and some work needs to be done. We can say that, for instance, the use of the standard deviation smell must be ruled by the type of data and data organization and that the use of functional dependencies does not make sense for spreadsheets with almost only numeric type cells.

Finally about the results obtained we can start by saying that, for the statistical smell, the results lead to conclude that his use would be more accurate if we could apply it to regions from a spreadsheet where the data were related instead of doing it for all the sheet. This way the values used by the smell would always be related to each other.

In the type smells that had the empty cells smell and the pattern smell, maybe we could either ignore the empty cells in the pattern smell or ignore the results from the empty cells smell because often they overlap results. Still, the type smells were mostly helpful to find forgotten cells in the spreadsheets.

The input smells were the string distance smell and the reference to empty cells smell. For the string distance in spite of the fact that when theorizing it sounded great, the results presented showed us the other way, this smell still has some work to be done. This work will be mentioned in the Section 6.1

About the references to empty cells happens exactly the opposite, all the results found were considered important finding because references to empty cells only make sense if the spreadsheet is a form, other than that may lead to latter problems.

The functional dependencies smell was the more hard-working smell, obtained by adapting an algorithm used in databases it would be expected that this smell worked better in the spreadsheets from databases. This not always happens, in fact many spreadsheets in the financial category have better results. This can be easily explained by the fact that this smell when analyzes a formula cell, instead of use the value calculate by the formula it uses the relative formula and doing this many patterns may be found. For instance, when you have

a full column with totals, the formulas of all those totals will be the same. This is easier to understand by looking to the Fig. 6.1, in the figure we can see the different values for each

Figure 6.1: *Formula Relative Explanation.*

case, the computed value that is the generated by the formula, the regular formula that is the formula read by excel, and the relative formula that is used by me to map the cells. We can see that the relative formula is the same for the three cases in spite being different for the others.

6.1 Future work

In the future the research of new smells is an important step, for instance, the detection of constants in the middle of formulas, because the use of constants instead of a reference to a cell may be problematic for future updates. With this smell we could detect some cells forgotten during past updates.

Another thing we should do in the future is to create some sort of layout smell, a smell to detect related cells in the spreadsheet. For instance if we have a column where the values are grades from one exercise, the values from all that column would be one detected area. This smell would be useful to identify cells out of boundaries and to combine with other smells in order to improve their accurateness.

Other possible smell would be the detection of sums written by hand, like we have mentioned in Section 5.4.7. There are some sheets where the result of the sum of some cell are written by hand, and this can also lead to problems.

On the tool level, the improvement of the implementation of the existent smells also needs to be done. For instance, in the functional dependencies the validation of results that are implied by only one column should be deeper investigated because at the first glance it seemed that almost all the detections made like this were wrong, but like we said this is something that should be subject to a better analysis. Another improvement to be

made is the on the string distance smell: a better parser should be created in order to avoid date comparison, or values with the symbol of currencies, or even avoid results where the Levenstein Distance is applied in numerical strings either full or partial *e.g.* "page 1" vs "page 2" or "0123" vs "0113".

On the quality of the data, a new and more reliable set should be found/created, what should be done is to clean and increment the existent one, we could either select by hand (painful) each spreadsheet that we find good or we could use, for instance, a software like the one that Software Improvement Group already has to analyze this corpus, select the spreadsheets with best results and filter them to a new set of spreadsheets.

About the ochiai smell that was introduced but no results were shown, we think that, since the output given by it being a percentage of correctness, in the future we could use it to validate the other results.

Finally, one other thing that would be nice to do is to combine our spreadsheet smells with the ones defined/adapted by Hermans.

Appendix A

Metric Tables

Metric Name	Brief Description	Functionality	Reliability	Maintainability	Efficiency	Usability
#Fan-In	References to other cells	x	x	x		x
#Fan-Out	References from other cells	x	x	x		x
#References	Total number of references	x	x	x		x
#Refs. to Empty Cells	Number of refs. to a Empty cell	x	x	x	x	x
#Constants in Formula	Number of constants in a formula	x	x	x		x

Table A.1: Cell Level Metrics.

Metric Name	Brief Description	Functionality	Reliability	Maintainability	Efficiency	Usability
#Fan-In	References to other sheets	x	x	x		x
#Fan-Out	References from other sheets	x	x	x		x
#References	Total number of references	x	x	x		x
#Paths	Number cells with chain references	x	x	x		x
#Blank Cells	Number of blank cells		x			
#Blank Rows	Number of blank rows		x	x		x
#Blank Columns	Number of blank columns		x	x		x
#Cells	Total number of cells			x		x
#Rows	Total rows			x		x
#Columns	Total columns			x		x
#Labels	Number of labels		x	x		
#Numerical	Number of numbers		x	x		
#Formulas	Number of formulas		x	x		
#Complex Formulas	Number of complex formulas		x	x	x	x
#Unique Formulas	Number of unique formulas		x	x	x	x

Table A.2: Sheet Level Metrics.

Metric Name	Brief Description	Functionality	Reliability	Maintainability	Efficiency	Usability
#Fan-In	References to other spreadsheets	x	x	x		x
#Fan-Out	References from other spreadsheets	x	x	x		x
#Paths	Number cells with chain references	x	x	x		x
#Circular References	References from one spreadsheet to other, then to the same.	x	x	x	x	x
#Blank Sheets	Number of blank sheets		x			

Table A.3: Spreadsheet Level Metrics.

Bibliography

[1] R. Abreu, P. Zoeteweij, and A. J. C. v. Gemund. Spectrum-based multiple fault localization. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 88–99, Washington, DC, USA, 2009. IEEE Computer Society.

[2] T. L. Alves, C. Ypma, and J. Visser. Deriving metric thresholds from benchmark data. In *International Conference on Software Maintenance*, pages 1–10, 2010.

[3] A. Bregar. Complexity metrics for spreadsheet models. volume abs/0802.3895, pages 85–93, Klagenfurt, 2004.

[4] F. Chiang and R. J. Miller. Discovering data quality rules. *The Proceedings of the VLDB Endowment.*, 1:1166–1177, August 2008.

[5] M. Corporation. Excel. `http://office.microsoft.com/pt-pt/excel/`.

[6] T. DeMarco. *Controlling Software Projects, Management Measurement & Estimation*. Prentice Hall, 1986.

[7] M. P. Felienne Hermans and A. van Deursen. Detecting and visualizing inter-worksheet smells in spreadsheets. Technical report, Delft University of Technology, 2011.

[8] N. E. Fenton and S. L. Pfleeger. *Software Metrics - A Rigorous and Practical Approach*. PWS, 1998.

[9] T. D. Foundation. Libreoffice. `http://www.libreoffice.org`.

[10] T. S. A. Foundation. Openoffice. `http://www.openoffice.org`.

[11] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA, 1999.

[12] Google. Google development toolkit (gwt). `http://code.google.com/intl/pt-PT/webtoolkit/`.

[13] Google. Google docs. `http://docs.google.com`.

[14] E. S. R. I. Group. Eusprig. `http://www.eusprig.org/horror-stories.htm`.

[15] I. Heitlager, T. Kuipers, and J. Visser. A practical model for measuring maintainability. In *Proceedings of the 6th International Conference on Quality of Information and Communications Technology*, pages 30–39, Washington, DC, USA, 2007. IEEE Computer Society.

[16] K. Hodnigg and R. T. Mittermeir. Metrics-based spreadsheet visualization: Support for focused maintenance. volume abs/0809.3009, London, 2008.

[17] M. F. Ii and G. Rothermel. The euses spreadsheet corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. In *In 1st Workshop on End-User Software Engineering*, pages 47–51, St. Louis, Missouri, USA, 2005.

[18] ISO. ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model. Technical report, International Organization for Standardization, 2001.

[19] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *In Proceedings of International Conference on Data Engineering*, Shanghai, China, 2009.

[20] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[21] M. Mäntylä, J. Vanhanen, and C. Lassenius. A taxonomy and an initial empirical study of bad smells in code. In *Proceedings of the International Conference on Software Maintenance*, ICSM '03, pages 381–384, Washington, DC, USA, 2003. IEEE Computer Society.

[22] B. A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, Cambridge, MA, USA, 1st edition, 1993.

[23] R. R. Panko and S. Aurigemma. Revising the panko-halverson taxonomy of spreadsheet errors. *Decision Support System*, 49:235–244, May 2010.

[24] R. R. Panko and R. P. Halverson Jr. Spreadsheets on trial: A survey of research on spreadsheet risks. In *Proceedings of the 29th Hawaii International Conference on System Sciences Volume 2: Decision Support and Knowledge-Based Systems*, HICSS '96, pages 326–335, Washington, DC, USA, 1996. IEEE Computer Society.

[25] C. Peixoto. Quality model for spreadsheets. Master's thesis, Informatics Department, University of Minho, 2011.

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

[27] S. G. Powell, K. R. Baker, and B. Lawson. Errors in operational spreadsheets: A review of the state of the art. In *Hawaii International Conference on System Sciences*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[28] T. A. P. Project. Apache poi. `http://poi.apache.org/`.

[29] K. Rajalingham, D. Chadwick, B. Knight, and D. Edwards. Quality control in spreadsheets: A software engineering-based approach to spreadsheet development. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, pages 1–9, Washington, DC, USA, 2000. IEEE Computer Society.

[30] K. Rajalingham, D. R. Chadwick, and B. Knight. Classification of spreadsheet errors. In *in: Symposium of the European Spreadsheet Risks Interest Group (EuSpRIG)*, Amsterdam, 2001.

[31] C. Scaffidi, M. Shaw, and B. Myers. The '55m end-user programmers' estimate revisited. Technical report, Carnegie Mellon University, Pittsburgh, 2005.