Towards a Catalog of Spreadsheet Smells*

Jácome Cunha[1], João P. Fernandes[12], Hugo Ribeiro[1], and João Saraiva[1]

[1] HASLab / INESC TEC, Universidade do Minho, Portugal
{jacome,jpaulo,jas}@di.uminho.pt, pg15970@alunos.uminho.pt
[2] Universidade do Porto, Portugal

Abstract. Spreadsheets are considered to be the most widely used programming language in the world, and reports have shown that 90% of real-world spreadsheets contain errors.
In this work, we try to identify spreadsheet smells, a concept adapted from software, which consists of a surface indication that usually corresponds to a deeper problem. Our smells have been integrated in a tool, and were computed for a large spreadsheet repository. Finally, the analysis of the results we obtained led to the refinement of our initial catalog.

Keywords: Spreadsheets, Code Smells, EUSES Corpus

1 Introduction

Spreadsheets are widely used, specially by non-professional programmers, the also called "end users" [21]. A typical end user is teacher, an engineer, a student, or anyone that is not a professional programmer. The number of end-user programmers vastly outnumbers the amount of professional programmers. In fact, studies suggest that in the U.S. alone there exist 11 million end users against 2.75 million of professional programmers [25]. In the same study, it is projected for 2012 a total number of 90 million end users, 55 million of which will be working on spreadsheets or databases.

The number of spreadsheet users provides enough evidence that millions of spreadsheets are created every year. The fact is that, since end users are not professional programmers, they usually do not follow the principles of good programming. Instead, they care about getting a concrete task done. This approach, together with the lack of support for abstraction, testing, encapsulation, or structured programming in spreadsheets, leads to reports showing that up to 90% of real-world spreadsheets contain errors [24], with concrete impacts on companies' profits.

In this paper we present a catalog and a methodology to identify *smells* in spreadsheets. The concept of *code smell* (or simply bad smell) was introduced by Martin Fowler [11] as a concrete evidence that a piece of software may have a

* This work has been supported by Fundação para a Ciência e a Tecnologia, under grants SFRH/BPD/73358/2010, SFRH/BPD/46987/2008, PTDC/EIA-CCO/108613/2008 and PTDC/EIA-CCO/108995/2008.

problem. Usually a smell is not an error in the program, but a characteristic that may cause problems understanding the software, for example, a long class in an object-oriented program, and updating and evolving the software. We present a methodology to define such smells in spreadsheets. We start by defining a set of possible smells as our initial catalog. Then we use a large repository to evaluate those smells. Finally, and as a result of this evaluation, we refine our spreadsheet smells in order to have a more robust catalog. To perform the detection of smells automatically we have developed a tool: SmellSheet Detective.

This paper is structured as follows. In Section 2 we present the methodology used to create, validate, evaluate and refine a catalog of bad smells for spreadsheets. Section 3 presents our initial catalog of bad smells. Then, in Section 5, we present the evaluation of our initial catalog in the EUSES corpus. In Section 4, we validate the results from the previous section. In Section 6 we adjust the initial catalog according to the results obtained. Section 7 introduces the developed tool to detect smells. In Section 8 we present related work and finally, Section 9 concludes the paper.

2 A Methodology to Identify Spreadsheet Smells

The detection of errors in software systems is an important software engineering technique. Software errors cause programs not to behave as expected and are responsible for several accidents. Even if not necessarily errors, the presence of bad smells in software code can make programs harder to understand, maintain, and evolve, for example. Martin Fowler popularized this notation of program smells in the context of object-oriented programming and this is now an important area of research. The detection of bad smells allows programmers to improve their programs by eliminating them.

In this section we present a methodology to define a catalog of bad smells for spreadsheets. This methodology is based on four steps: *catalog definition*, *catalog validation*, *catalog evaluation* and *catalog refinement*.

Step 1: *Catalog Definition* - based on our personal experiences, we propose an initial catalog of spreadsheet bad smells. We consider a bad smell in spreadsheets a reference in a formula to an empty cell, an empty cell in a table, etc. The full catalog is presented in Section 3.

Step 2: *Catalog Validation* - in order to validate the catalog we consider a large repository of spreadsheets, the EUSES corpus [15], that contains more than 5000 spreadsheets, and we detect smells in a representative sub-set of the repository, as described in Section 4.

Step 3: *Catalog Evaluation* - in order to evaluate the results of our empirical experiment performed in the previous step, we manually inspect all bad smells detected by our catalog. We classify the detected smells in four categories: *not a smell*, *low smell*, *medium smell* and *high smell*. We present this evaluation in detail in Section 5.

Step 4: *Catalog Refinement* - based on the evaluation performed in Step 3, we have adjusted our catalog by identifying wrong smells, by refining previously

defined smells and by adding new smells that showed up when manually inspecting EUSES spreadsheets. The result of this step is our catalog of spreadsheet bad smells, which is shown in Section 6.

In order to validate our catalog in a large corpus we need a tool to automatically detect smells in spreadsheets. Thus, the full catalog defined in Step 1 was implemented as a software tool, SmellSheet Detective, that we present in Section 7.

3 Spreadsheet Smells: Catalog Definition

The notion of *bad smell* emerged from the need to identify the cases for which the internal structure of a piece of software could be improved. A bad smell is typically something that is easy to spot, and an indicator of a possible concrete issue. However, a smell is not something that can necessarily be considered an error. An example of a smell proposed by Fowler and that applies to a software project is the *long method smell*, that implements the notion that defining too long methods (i.e, methods larger than around a dozen lines of code) may lead to understandability and maintainability problems in the future.

The smells introduced by Fowler consist of a single flat list, but Mantyla has created a taxonomy for all those smells [20]. Similarly to what Mantyla has done we also grouped our smells in different categories: *Statistical Smells*, *Type Smells*, *Content Smells* and *Functional Dependencies Based Smells*.

In Figure 1 we present a spreadsheet, slightly adapted from a spreadsheet in the EUSES repository, where we can observe at least one smell belonging to each of the categories that we have defined. In the next sections, we present in detail the smells that we have fitted in each of these categories.

3.1 Statistical Smells

This category groups smells that are calculated through statistical analysis, namely the *Standard Deviation* smell.

– **Standard Deviation**

This smell detects, for a group of cells holding numerical values, the ones that do not follow their normal distribution.

Detection Most spreadsheets with numeric values are organized either by rows or columns, and it is often the case that wrong values are introduced without the user ever noticing. The *Standard Deviation* smell is detected by analyzing the rows (or columns) of a spreadsheet 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.

	A	B	C	D	E	F	G	H	I	J
1	code	upc	description	size	store_nr	week	qty	price	onSale	profit
2	653	1111140009	DOVE DISH LIQUID	42 OZ	101	385	2	1.99		3.98
3	653	1111140009	DOVE DISH LIQUID	42 OZ	101	0	5	1.99		9.95
4	653	123	DOVE DISH LIQUID	42 OZ	101	387	4	1.99	=G * H	7.96
5	653	1111140009	DOVE DISH LIQUID	42 OZ	101	388	4	1.99		7.96
6	653	1111140009		42 OZ	102	391	6	2.19		13.14
7	653	1111140009	DOVE DISH LIQUID	42 OZ	102	392	8	2.19		17.52
8	654	1111143002	SUNLIGHT DISH LIQUIDS	64 OZ	100	383	9	2.99		26.91
9	654	1111143002	SUNLIGHT DISH LIQUID	64 OZ	100	384	21	2.99		62.79
10	654	1111143002	SUNLIGHT DISH LIQUID	64 OZ	100	385	10	2.99		29.90
11	654	1111143002	SUNLIGHT DISH LIQUID	64 OZ	100	386	6	2.99		17.94
12	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	102	391	35	3.39	S	118.65
13	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	102	392	4	3.79		15.16
14	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	383	7	1.75		12.25
15	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	384	5	3.77		18.85
16	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	385		3.79		0.00
17	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	386	1	3.49		3.49
18	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	387	2	2.86		5.72
19	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	388	3	2.98		8.94
20	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	389	2	3.49		6.98
21	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	390	1	1.99		1.99
22	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	391	10	3.39	S	33.90
23	655	1111147006	SUNLIGHT POWDER AUTO	85 OZ	103	392	1	3.49		3.49

Fig. 1: A spreadsheet for a warehouse of cleaning products.

Example By inspecting Figure 1 we have realized, for instance, that the standard deviation of column B values is 2.369E8. Then, the values that are acceptable by normal distribution should be within the range [5.868E8, 1.534E9]. This means that a smell is detected for cell B4, since it contains the value 123. Cell G12 is also indicated as a smell by standard deviation analysis.

3.2 Type Smells

In this category we have included the *Empty Cell* and the *Pattern Finder* smells, both analyzing the type of a cell, being it a *Label*, a *Number*, a *Formula* or an *Empty Cell*.

– **Empty Cell**

In order to detect cells that are left empty, but that occur in a context that suggests they should have been filled in, we have implemented the *Empty Cell* smell.

Detection What we do here is to select all possible windows of five cells from each row (or column) and verify in each window whether it holds or not precisely one empty cell.

Example In the spreadsheet in Figure 1, we can see that cells C6 and G16 are empty. However, they occur in a context where all their neighbor cells have been filled in; for these cells, a smell is signaled. Notice that, for example in column I, several other empty cells are not pointed out as smells: indeed, they occur in windows of 5 cells where more than just a single cell is empty.

– **Pattern Finder**

For faster development, spreadsheet users often simplify parts of formulas by introducing in them constant (numeric or label) values. This is a poor design decision that, if not corrected at some point, may lead to problems. In order to point out the cells where this situation occurs, we have implemented the *Pattern Finder* smell, that in fact works not only for formulas. Indeed, this smell is able of finding patterns in a sheet such as a row containing only numerical values except for one cell holding a label or a formula, or being empty. We flag such a cell as a smell.

Detection The detection of *Pattern Finder* smell follows an approach in all similar to the detection of *Empty Cell* smells. In *Pattern Finder*, we use a four cell window and we search for one cell with type characteristics that are different from the ones in the other window cells. In this smell, we have chosen a smaller window since the occurrence of patterns other than empty cell patterns is also smaller.

Example In Figure 1 we can see that cell F3 holds the textual value "o", being then a cell of type Label; furthermore, it is surrounded by numbers in the window constructed as described above: F3 is then a smell, and indeed it is likely that the value that instead should have been inserted is "0".

3.3 Content Smells

In this category we include smells that are found through the analysis of the content of cells: the *String Distance* smell and the *Reference to Blank Cells* smell.

– **String Distance**

Typographical errors are frequent when typing in a computer. In order to try to detect these type of errors, we have implemented the *String Distance* smell, that signals string cells that differ minimally with respect to other cells in a spreadsheet.

Detection In the detection of this smell we use the algorithm created by Levenshtein [17]. This algorithm compares two strings and finds the minimum number of edits that are needed to transform one string into another.

In our case, we apply Levenshtein's algorithm to each pair of strings in a row (or column), and signal as a smell the cases for which the result is the value 1. This means that a single change to a string in the pair is enough to obtain the other string. We have furthermore limited the comparison to strings of length greater than three: this prevents, for example, all cells in a row holding the alphabet letters to be considered smells.

Example We can see a *String Distance* smell in Figure 1: in row C, cell C8 holds the plural of cells C9 to C11; this suggests a typing error on cell C8.

– **Reference to Empty Cells**

The existence of formulas pointing to empty cells is a typical source of errors in spreadsheets, and we have therefore included in our catalog a smell for detecting all occurrences of this situation.

Detection The detection of this smell is implemented by searching for all formulas in a spreadsheet and by gathering all their references. Then, we simply check whether each of these references points to an empty cell or not.

Example Cell J16 of Figure 1 is recognized as a smell since its value is calculated using the value of cell G16, which is empty.

3.4 Functional Dependencies Based Smells

In this category, we have adapted to spreadsheets data mining techniques that were first introduced for databases. In particular, we search for *dirty values* in a spreadsheet.

– **Quasi-Functional Dependencies (QFD)**

In [4] it is described a technique to identify dirty values using a slightly relaxed version of *Functional Dependencies* [5], and this is the technique that we use here. Two columns A and B are *Functionally Dependent* if multiple occurrences of the same value in A always correspond to the same value in B. For instance, in the example show in Figure 1, column A functionally determines column B, as knowing one particular **code** is enough to know its associated **upc**. When equal values in a column correspond to the same value in another column, except for a *small* number of cases, this is a situation that *smells*, and that we flag.

Detection The implementation of this smell follows the approach described in [4]. It involves collecting and matching all data in a spreadsheet in order to find QFD, actually identifying dirty values and then ranking all these values.

Example Cells E12 and E13 of the spreadsheet presented in Figure 1 are pointed out as smells by the analysis of QFD. Indeed, we may see that the values $(655, 1111147006, SUNLIGHT\ POWDER\ AUTO, 85\ OZ)$ in columns A to F always determine the value 103 in column E, except precisely for cells E12 and E13.

4 Catalog Validation

In the previous section, we have described the catalog of spreadsheet smells that we propose in this paper. Now, we need to validate our catalog against real spreadsheets. For this purpose, we will use the EUSES Corpus [15], a large spreadsheet repository which has been widely used by the software engineering community [1, 10]. This repository consists of 5606 spreadsheets which have been divided into six categories: Database, Financial, Grades, Homework, Inventory and Modeling. The proportion of spreadsheets per category can be seen in Figure 2.

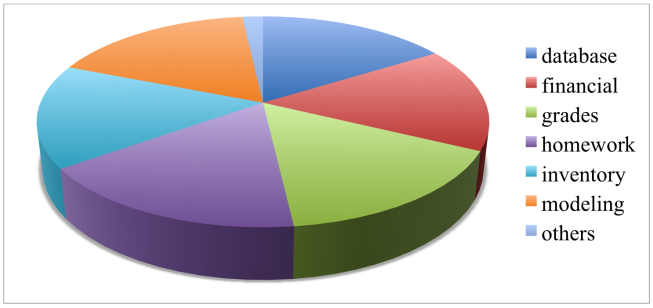

Fig. 2: EUSES spreadsheet categories.

In order to evaluate our catalog, we have implemented in a tool, SmellSheet Detective, the spreadsheet smells that the catalog includes (this tool is presented in detail in Section 7). For each smell, the results we obtain are presented in Table 1: the different smells are listed in rows and the different EUSES categories are listed in columns.

The results observed in Table 1 consider 180 EUSES spreadsheets *only*. Still, smells were identified, in those spreadsheets, 3841 times. This is a number that is large enough for validation purposes while still enabling the manual validation of the detected smells. Indeed, the next phase of our methodology was to inspect individually and in detail each spreadsheet that we considered, and validate the smells identified for them. The validation tried to establish two things: a) whether the identification of a smell was accurate or not; b) in case the identified smell was accurately pointed out, how *severe* we consider it to be, i.e., how much a smell impacts in the overall quality of the spreadsheet.

Smell/Category	Database	Financial	Grades	Homework	Inventory	Modeling	Total
Empty cells	53	27	56	42	109	106	393
Patterns	86	62	66	58	111	114	497
Std. Dev.	33	35	134	14	32	7	255
String Dist.	25	3	1357	231	158	658	2442
CFD	12	13	150	2	13	24	204
Ref2empty	0	23	27	0	0	0	50
Total:	209	163	1790	347	423	909	3841

Table 1: Number of smells collected for each EUSES category.

5 Catalog Evaluation

The purpose of this section is to establish a model to evaluate the smells that our catalog identifies as such. For this, we need to define how the smells that are identified can be measured and classified. We have inspired ourselves in the technique used by the software quality measurement company Software Improvement Group [14] and empirically classify the smells that we detect under three categories:

- **Low smells (ls):** In this category we include the cells that are identified as smells but that we can not ensure that indeed constitute smells, neither can we ensure that, on the contrary, do not constitute smells;
- **Medium smells (ms):** This category includes the cells that we we can ensure are smells with a high degree of certainty, but that belong to spreadsheets that we can not fully understand;
- **High smells (hs):** Here, we include all smells that are detected and that our inspection confirmed as such, without a doubt;
- **Not smells (ns):** Finally, we include here all smells that we consider to have been wrongly detected, for example, due to tool malfunctioning;

The creation of this classification model made it possible to uniformly classify spreadsheet smells. In the remaining of this section, we present different views of the classification we establish for each smell we identify.

In Table 2, we show the absolute results of classifying each detected smell under one category of the catalog validation model. As a concrete example of this, the value 7 in line 'Empty Cells', column 'Database', sub-column 'ls' indicates that 7 of the smells that were identified by the 'Empty cells' smell in spreadsheets of the 'Databases' EUSES category were identified as low smells.

In Table 3 we present the percentage of automatic detections, per smell, that were classified in each of the validation model categories. Taking the same cell as above, it means that 13% of empty cell smells that were identified in database spreadsheets were classified as low smells.

The information of Table 4 shows the percentage of automatic detections, per validation model category, for all the smells. Again reading the same table

Smell/Level	Database				Financial				Grades				Homework				Inventory				Modeling			
	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns
Empty cells	7	1	0	45	3	2	0	22	15	1	0	40	1	0	0	41	1	0	0	108	88	0	0	18
Patterns	7	1	0	78	6	2	0	54	16	5	0	45	1	1	0	56	1	0	0	110	93	0	0	21
Std. Dev.	10	0	0	23	2	0	0	33	2	0	0	132	5	0	0	9	1	0	0	31	1	0	0	6
String Dist.	0	0	1	24	0	0	0	13	0	0	4	1353	1	0	0	230	1	0	2	155	11	2	0	645
CFDs	4	7	1	0	0	0	3	0	55	5	1	89	1	0	0	1	1	1	0	11	3	0	1	20
Ref2empty	0	0	0	0	9	1	7	6	0	8	1	18	0	0	0	0	0	0	0	0	0	0	0	0
Total:	28	9	2	170	20	5	10	128	88	19	6	1677	9	1	0	337	5	1	2	415	196	2	1	710

Table 2: Smell classification (absolute values).

Smell/Level	Database				Financial				Grades				Homework				Inventory				Modeling			
	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns
Empty cells	13	2	0	85	11	7	0	81	27	2	0	71	2	0	0	98	1	0	0	99	83	0	0	17
Patterns	8	1	0	91	10	3	0	87	24	8	0	68	2	2	0	97	1	0	0	99	82	0	0	18
Std. Dev.	30	0	0	70	6	0	0	94	1	0	0	99	36	0	0	64	3	0	0	97	14	0	0	86
String Dist.	0	0	4	96	0	0	0	100	0	0	0	100	0	0	0	100	1	0	1	98	2	0	0	98
QFD	33	58	8	0	0	0	100	0	37	3	1	59	50	0	0	50	8	8	0	85	13	0	4	83
Ref2empty	0	0	0	0	39	4	30	26	0	30	4	67	0	0	0	0	0	0	0	0	0	0	0	0
Total:	13	4	1	81	12	3	6	79	5	1	0	94	3	0	0	97	1	0	0	98	22	0	0	78

Table 3: Smell classification (relative values, per smell).

element, 25% of the smells that we have included in the 'Low Smells' category, for database spreadsheets, were identified by the 'Empty Cell' smell.

Smell/Level	Database				Financial				Grades				Homework				Inventory				Modeling			
	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns	ls	ms	hs	ns
Empty cells	25	11	0	26	15	40	0	17	17	5	0	2	11	0	0	12	20	0	0	26	45	0	0	3
Patterns	25	11	0	46	30	40	0	42	18	26	0	3	11	100	0	17	20	0	0	27	47	0	0	3
Std. Dev.	36	0	0	14	10	0	0	26	2	0	0	8	56	0	0	3	20	0	0	7	1	0	0	1
String Dist.	0	0	50	14	0	0	0	10	0	0	67	81	11	0	0	68	20	0	100	37	6	100	0	91
QFD	14	78	50	0	0	0	30	0	63	26	17	5	11	0	0	0	20	100	0	3	2	0	100	3
Ref2empty	0	0	0	0	45	20	70	5	0	42	17	1	0	0	0	0	0	0	0	0	0	0	0	0

Table 4: Smell classification (relative values, per evaluation model category).

Finally, in Table 5, we present the total number of smells, for each of smell and for each model evaluation category.

Smell \ Level	Low smells	Medium smells	High smells	Not smells	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
QFD	64	13	6	121	204
Ref2empty	9	9	8	24	50
Total:	346	37	21	3437	3841

Table 5: Global total results (absolute values).

This table shows that we are able to detect smells in the EUSES corpus. However, 90% of the smelly cells detected were not confirmed as such after the manual inspection of those 3841 cells. In the next section we conduct a refinement on the initial catalog that we have defined, based on the analysis of the results presented in this section.

6 Catalog Refinement

Having evaluated our initial spreadsheet catalog in a representative set of spreadsheets of the EUSES repository, firstly, by automatically running a spreadsheet smell detector and, secondly, by manually validating the results, we can now refine the catalog based on the results and experience obtained. We consider three different types of refinement:

- *Overlapped smells:* The pattern smell overlaps the empty cell smell. In fact, all empty cells were detected by both approaches. Since we want to distinguish an empty cell from a pattern in order to have a more precise notion of the smell itself, then the empty cells should not be considered as a pattern smell.
- *New smells detected:* when manually inspecting a large number of EUSES spreadsheets, we detected several bad smells in spreadsheets that we had not considered, namely, the use o summation cells where no formulas are used to do the calculations: constant values are used instead! This is the case, for example, in the spreadsheet file *"FIN_hospitaldataset2002 MEMORIAL"*, cell F22. Thus, a smell detecting constant values where formulas should be used needs to be added to the catalog.
- *Wrong smells:* The string distances smell produced the highest number of wrongly detected cells. Because we use the Levenshtein distance algorithm to find close 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. This would lead to an improvement in this smell of 88%.

The other smells 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.

Next we present the results of running our smell detector according to the catalog refinements. It should be noticed that no new smells have been implemented.

Smell \ Level	Low smells	Medium smells	High smells	Not smells	Total
Empty cells	115	4	0	274	393
Patterns	9	5	0	90	101
Std. Dev. cells	21	0	0	234	255
String Dist.	13	2	7	290	312
QFD	64	13	6	121	204
Ref2empty	9	9	8	24	50
Total:	231	33	21	1033	1315

Table 6: Refined catalog results.

As the results included in Table 6 show, we were able to detect 285 manually validated smells out of 1315 detected by our smell detector. Thus we have 21.7% of correct smells detected. Moreover, 54 out of 231 (18.9%) presents a medium or high level probability of the occurrence of a problem in the spreadsheet. As a result of our refinement, we have improved our smell detection from 90% to 78% of false positive smells.

Based on our analysis we present in Table 7 the EUSES categories for which we believe each smell can be effectively used.

Smell \ Category	Database	Financial	Grades	Homework	Inventory	Modeling
Empty cells						X
Patterns						X
Std. Dev.		X				
String Dist.	X	X	X	X	X	X
QFD	X	X	X			X
Ref2empty	X	X	X	X	X	X

Table 7: The use of smells in EUSES categories.

The table show that all smells are applicable in at least two categories and two of them (string distance and references to empty cells) are applicable in all categories.

7 SmellSheet Detective

To analyze the selected spreadsheet sample we implemented the tool SmellSheet Detective which detects the smells introduced in the Section 3. This implementation was made using the Java programming language [16], the Google Web Toolkit (GWT) [12], the Apache POI library [3] and the Google libraries to work with spreadsheets from the Google Docs [13]. We decided to support spreadsheets written in the Google Docs platform because it is becoming more and more used. In fact, the popular Microsoft Office suite has also its online version [18]. Indeed, the migration from desktop to online applications is becoming very common. Nevertheless, we also support spreadsheet written using desktop applications, as can be seen in Figure 3.

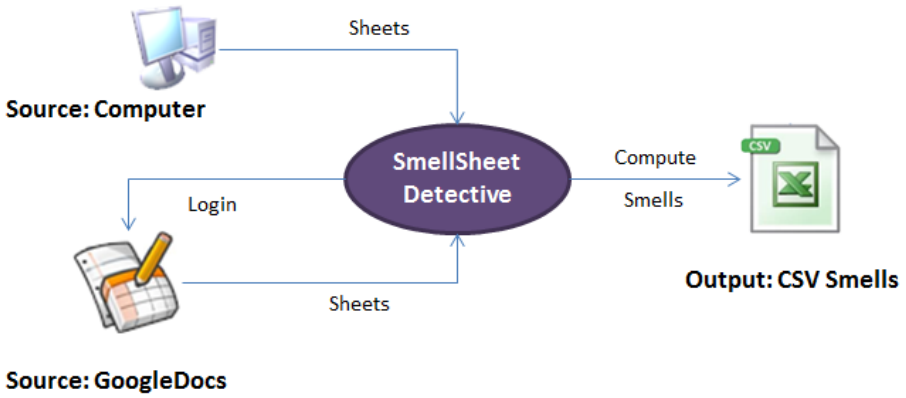

Fig. 3: SmellSheet Detective architecture.

The tool can receive spreadsheets, as we mentioned before, from Google Docs or directly from the computer where it is running. If the Google Docs source is used, a valid Google account login is required. The tool then selects all the spreadsheets in the account and shows them to the user. After selecting a particular spreadsheet, the user can select a single sheet to be analyzed, or otherwise the entire spreadsheet is used. If we use the direct upload source, the user can browse the spreadsheets in the computer and select one. The sheet selection works in a similar way as explained.

After the selection process, the SmellSheet Detective searches the sheets selected for bad smells. The detection of smells is done as explained in Section 3.

Finally, the tool generates a comma-separated value file with the outputs. Each row will contain information about a single sheet: the first value is the name of the spreadsheet/sheet and the next numbers are the ones reported by the different parts of the tool implementing the detection of the smells empty cells, standard deviation, string distance, functional dependencies, pattern finder and references to blank cells.

We have also defined a model-based spreadsheet environment that guides end users to introduce correct data [7–9]. We are now integrating the SmellSheet Detective in this environment. These tools are available at `ssaapp.di.uminho.pt`.

8 Related Work

Fowler [11] was the first to introduce the concept of smell, to create a list of 22 smells and pointing a possible solution for each one of them. In the sequence of Fowler study, Mantyla *et al.* [19] has created a taxonomy for the smells listed by Fowler so they could be easier to understand. They created five groups of smells, namely, the bloaters, the object-oriented abusers, the change preventers, the dispensables and the couplers.

Hermans *et al.* [10] adapted Fowler's work to detect inter-worksheets smells. Their work differs from the work we present here in the fundamental approach to define spreadsheet smells: while Hermans adapts Fowler's smells to the spreadsheet realm, we analyze a large corpus, and based on that, we define, validate, evaluate and refine spreadsheet specific smells. We believe that the two approaches are orthogonal and as a consequence a full catalog should include both smells.

The analysis of possible errors in spreadsheets was also studied by several researchers, namely, Panko *et al.* [22] who proposed a revised taxonomy for spreadsheet errors [23], Correia *et al.* [6] who used Goal Question Metric to measure the maintainability of spreadsheets, and Abraham *et al.* [2] who developed a tool for debugging spreadsheets.

9 Conclusion

In this paper we have presented a detailed study on spreadsheet smell detection. We have presented a catalog of smells that are spreadsheet specific and we have validated it using a large spreadsheet repository. Furthermore, we have refined the catalog by manually evaluating the results obtained. The smell detection has been implemented in the SmellSheet Detective tool, and we have presented our preliminary results that show that we are able to detect a significant amount of smells using our tool. Also, we have confirmed that more than 20% of the detected smelly cells point to a possible problem in the spreadsheet.

In the future, we intend to extend the work presented in this paper in two different ways. Firstly, we believe that the number and the type of smells that belong to a spreadsheet smell catalog can be further extended. Secondly, ever since the smells were first proposed for software repositories, they are usually associated with refactorings that can eliminate them. These are promising research directions that we are already exploring and whose results we plan to bring out in the near future.

References

1. Abraham, R., Erwig, M.: Inferring templates from spreadsheets. In: Proc. of the 28th Int. Conf. on Software Engineering. pp. 182–191. ACM, New York, NY, USA (2006)
2. Abraham, R., Erwig, M.: Goaldebug: A spreadsheet debugger for end users. In: ICSE '07: Proceedings of the 29th international conference on Software Engineering. pp. 251–260. IEEE Computer Society, Washington, DC, USA (2007)
3. Apache POI: `http://poi.apache.org`
4. Chiang, F., Miller, R.J.: Discovering data quality rules. The Proceedings of the VLDB Endowment. 1, 1166–1177 (August 2008)
5. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM 13(6), 377–387 (1970)
6. Correia, J.P., Ferreira, M.A.: Measuring maintainability of spreadsheets in the wild. In: ICSM. pp. 516–519. IEEE (2011)
7. Cunha, J., Fernandes, J.P., Mendes, J., Saraiva, J.: MDSheet: A Framework for Model-driven Spreadsheet Engineering. In: Proceedings of the 34rd International Conference on Software Engineering. ICSE'12, ACM (2012), to appear.
8. Cunha, J., Mendes, J., Fernandes, J.P., Saraiva, J.: Embedding and evolution of spreadsheet models in spreadsheet systems. In: IEEE Symp. on Visual Languages and Human-Centric Computing. pp. 179–186. IEEE CS (2011)
9. Cunha, J., Visser, J., Alves, T., Saraiva, J.: Type-safe evolution of spreadsheets. In: Int. Conf. on Fundamental Approaches to Software Engineering. pp. 186–201. FASE'11/ETAPS'11, Springer-Verlag, Berlin, Heidelberg (2011)
10. Felienne Hermans, M.P., van Deursen, A.: Detecting and visualizing inter-worksheet smells in spreadsheets. In: Proceedings of the 34rd International Conference on Software Engineering. ICSE'12, ACM (2012), to appear.
11. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston, MA, USA (1999)
12. Google Development Toolkit (GWT): `http://code.google.com/intl/pt-PT/webtoolkit/`
13. Google Docs: `http://docs.google.com`
14. Heitlager, I., Kuipers, T., Visser, J.: A practical model for measuring maintainability. In: Proceedings of the 6th International Conference on Quality of Information and Communications Technology. pp. 30–39. IEEE Computer Society, Washington, DC, USA (2007)
15. Ii, M.F., Rothermel, G.: The euses spreadsheet corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. In: In 1st Workshop on End-User Software Engineering. pp. 47–51. St. Louis, Missouri, USA (2005)
16. Java: `http://www.java.com`
17. Levenshtein, V.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady 10, 707 (1966)
18. Live, M.O.: `http://www.officelive.com`
19. Mäntylä, M., Vanhanen, J., Lassenius, C.: A taxonomy and an initial empirical study of bad smells in code. In: Proceedings of the International Conference on Software Maintenance. pp. 381–384. ICSM '03, IEEE Computer Society, Washington, DC, USA (2003)
20. Mäntylä, M.V., Lassenius, C.: Subjective evaluation of software evolvability using code smells: An empirical study. Empirical Softw. Engg. 11, 395–431 (September 2006)

21. Nardi, B.A.: A Small Matter of Programming: Perspectives on End User Computing. MIT Press, Cambridge, MA, USA, 1st edn. (1993)
22. Panko, R.R., Aurigemma, S.: Revising the panko-halverson taxonomy of spreadsheet errors. Decision Support System 49, 235–244 (May 2010)
23. Panko, R.R., Halverson Jr, R.P.: 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. pp. 326–335. HICSS '96, IEEE Computer Society, Washington, DC, USA (1996)
24. Rajalingham, K., Chadwick, D.R., Knight, B.: Classification of spreadsheet errors. In: Symposium of the European Spreadsheet Risks Interest Group (EuSpRIG). Amsterdam (2001)
25. Scaffidi, C., Shaw, M., Myers, B.: The '55m end-user programmers' estimate revisited. Tech. rep., Carnegie Mellon University, Pittsburgh (2005)