SpreadsheetDoc: An Excel Add-in for Documenting Spreadsheets

Diogo Canteiro and Jácome Cunha

Universidade Nova de Lisboa, Portugal
d.canteiro@campus.fct.unl.pt
jacome@fct.unl.pt

Abstract. Documentation is an important artefact of any software product. This is also the case for spreadsheets were, even considering an industrial setting, only 30% have some kind of documentation. This makes their usage and maintenance very difficult.
In this paper we describe a tool, SpreadsheetDoc, that allows users to document spreadsheets in a structured way, allowing them to describe different parts of spreadsheets. For instance, for (future) spreadsheet users, it is possible to describe input and output cells, and for (future) developers, it is possible to describe computation, that is, formulas: their arguments, their internal computations, and their outputs.

Keywords: Spreadsheet, Documentation, Tool, Excel, Add-in

1 Introduction

Nowadays, there are a huge number of people using spreadsheets. In fact, spreadsheet systems are the most used programming system [13], specially by non-professional programmers, the so-called end users. As in other programming languages/environments, it is quite common to find spreadsheets with errors. In fact, the error rate within spreadsheets can be up to 90% [15]. The European Spreadsheet Risk Interest Group (EuSpRIG)[1] regularly updates their web site with new stories reporting the loses (economical, brand recognition, etc.) caused by errors in spreadsheets to companies and other entities.

Many reasons exist for this scenario: the lack of abstraction, of a testing methodology, or a (very) weak type system. Some errors can also be explained by the lack or poor documentation [18]. Indeed, in many of the cases reported by EuSpRIG, the lack of or bad documentation is mentioned. Moreover, software tends to lose some of its efficiency when no proper documentation is available [7]. Without the proper documentation users and developers have more difficulties in understanding, using, and updating the software. The same happens for spreadsheets. In a recent study in a financial institution, researchers found that 70% of users that receive spreadsheets from colleagues have difficulties understanding

[1] http://www.eusprig.org/

them [11]. This transferring scenario is quite common as 85% of the study participants reported doing so. The same authors report that spreadsheet users browse them for hours trying to understand them since only 1/3 are documented [10].

Unfortunately, spreadsheet systems do not have a proper form to document their programs. In modern spreadsheet systems it is possible to add general notes to a cell, but that is a very unstructured way of doing documentation, when compared to what tools like JavaDoc allow. This can be compared as to write ad-hoc comments in a textual language. This makes it quite hard for spreadsheet developers to actually document their spreadsheets. For instance, in [9] the authors analysed more than 15.000 spreadsheets available in the Enron Email Archive [12], containing the emails from the Enron corporation. They found that some spreadsheet documentation was in the emails themselves, instead of being in the spreadsheets. This shows that there is the need to document, but not the proper means.

Users tend to workaround this situation documenting their spreadsheets as possible. Some write the documentation on a separate worksheet and reference to it informing that such worksheet is the documentation of the spreadsheet. In this case, it is not possible to see the documentation and the corresponding document artefacts at the same time, as one can do using for instance JavaDoc, making it difficult to relate the documentation with the actual spreadsheet content. Others write the documentation on the worksheet with the content, close by the cells they want to describe. However, in these cases users are inserting extra cells in the spreadsheet, which are not part of the program, making it more complex.

These kinds of documentation make users question its use. Although it is important to document software, it cannot be done in any way. It is important to write and organize it in such a way that the target readers will get what they want. A good documentation will increase the users' efficiency and effectiveness, and thus, their productivity [7, 16]. Indeed, JavaDoc, for the Java programming language, is a good example of a successful way of documenting software.

In this paper we present a tool, SpreadsheetDoc, described in Section 3, to guide spreadsheet developers to write proper documentation. In this case, two kinds of documentation should be written, as suggested in [20]: i) documentation for end users, and ii) documentation for developers. Notice that both these types of documentation should be written by the spreadsheet developers, but part is intended to be read by end users and part by developers.

Spreadsheet end users are the ones interested in executing the spreadsheet to compute the results they want. Thus, they are mostly interested in understanding which cells they should fill in to feed the program, that is, the input, and were they can find the results, that is, the output. So, spreadsheet developers should mark and document all the input and output cells, and write documentation for users, and not for developers, that is, simple and straightforward documentation. The purpose of the spreadsheet file, and of each worksheet also fits in this category so users can find the spreadsheet and corresponding worksheet they need. We describe in detail how to do this in Section 4.

On the other hand, maintainers and future developers of the spreadsheet must have more technical information about the computations performed. Thus, cells containing complex formulas should also be documented. In this case, the documentation should be technical so others can later correct or evolve the spreadsheet. We discuss this is Section 5

With our tool we also allow to document a particular column, row, or range of cells (for instance, a table in a spreadsheet, that is, a range separated of the remaining cells by empty columns and rows).

Users can then read the documentation within the spreadsheet itself, in the context of the part of the spreadsheet they are using, or read the complete documentation in a web page, which is generated by our tool in a similar way as JavaDoc. To generate the web page we first create an XML file where the complete documentation is saved. Although we decided to present to users a web page, this intermediate format makes it possible to present documentation in a different way. Moreover, it also allows to import documentation produced by other systems, as long as the correct XML file is available. This allows to publish inside a corporation all the documentation of all spreadsheets, making it easier for collaborators to find one that already does what they need. Moreover, since spreadsheets can reference other spreadsheets, the navigation to the corresponding documentation is straightforward, as links connect them.

In Section 2 we present a running example, in Section 6 we discuss related work, and in Section 7 we present our conclusions and future work.

2 Motivational Example

2.1 Definitions

Before we introduce our motivational example, we will define a few concepts from the spreadsheet realm.

Workbook/Spreadsheet A workbook is a spreadsheet file. The term spreadsheet is often used to refer to a workbook, when in fact it refers to the computer program, such as Excel. We will use these terms interchangeably.

Worksheet A worksheet, or simply sheet, is a single page of a workbook, that is, one of the tabs that can be found at the bottom of the spreadsheet (in most spreadsheet systems).

Cell A cell is a rectangular box in a worksheet, that is, the intersection point of a vertical line (column) and a horizontal line (row). Its name is the concatenation of its coordinates: a letter for the column and a number for the row. It also has content, which can be plain values (for instance, `4` or `Bid`), or formulas (for instance, `=SUM(A1:A3)`).

Row Refers to all the cells contained in a horizontal line (given by a number).

Column Refers to all the cells contained in a vertical line (given by a letter).

Range A range is a group of cells in a worksheet that form a rectangular area.

Input Cell A cell referenced by others, but not referencing any other cell.

Output Cell A cell that references other cells, but its not referenced by others.

2.2 Example

We now describe a spreadsheet which we will use as a running example. This spreadsheet, shown in Figure 1, was introduced in a book describing how to create spreadsheets [14].

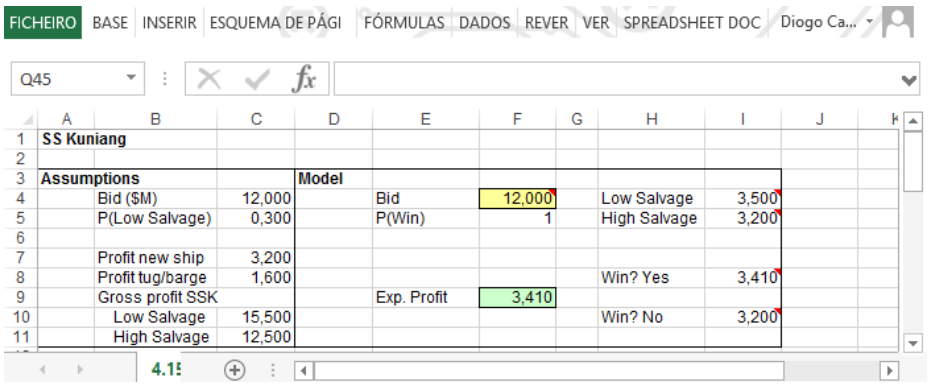

Fig. 1. A spreadsheet to calculate the winning probabilities of an auction.

This spreadsheet calculates the probability of winning an auction, according to a set of assumptions. Although this spreadsheet is well organized and rather small, it is already difficult to understand. In Figure 2 we show the same spreadsheet, but now with the formulas visible.

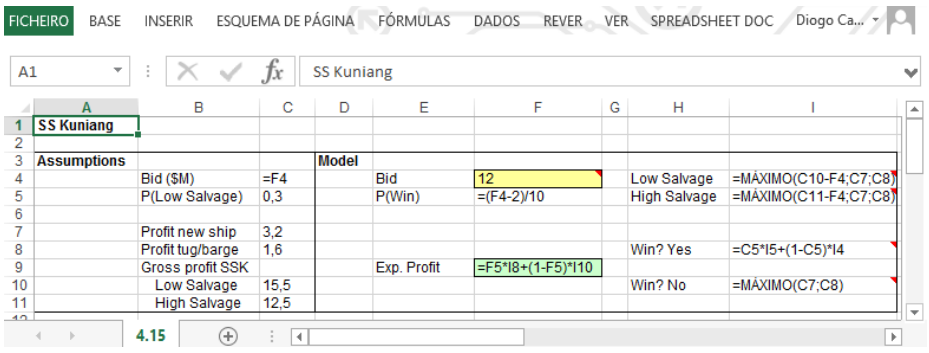

Fig. 2. Spreadsheet of Figure 1 with formulas visible.

In fact, and since this is a well designed spreadsheet, some cells even have comments on them (denoted by the small triangle in the top right corner of the corresponding cells). We list next the comments from the spreadsheet:

F4 Decision: Bid (in $million)

I4 Net value if the salvage value is low.

I5 Net value if the salvage value is high.

I8 Net value if the bid is successful.

I9 Net value if the bid is unsuccessful.

In the book, one can read some more details about this spreadsheet and corresponding computations. What we envision is a system where one can describe the different parts of the spreadsheet, but in a systematic approach, and in the corresponding context.

For instance, cell F5 calculates the probability of winning the auction. The formula present in the book is `P(Win) = (Bid - 2)/10, for 2` $\leq$ `Bid` $\leq$ `12`. This formula is more direct than the one presented in the spreadsheet, and the range of `Bid` (F4) is now clear. This should be part of that cell's documentation.

Using our approach, to document such formula, the user would click on the button to describe cell ("Cell"), under the group "Content Documentation", and the wizard shown in Figure 3 would appear:

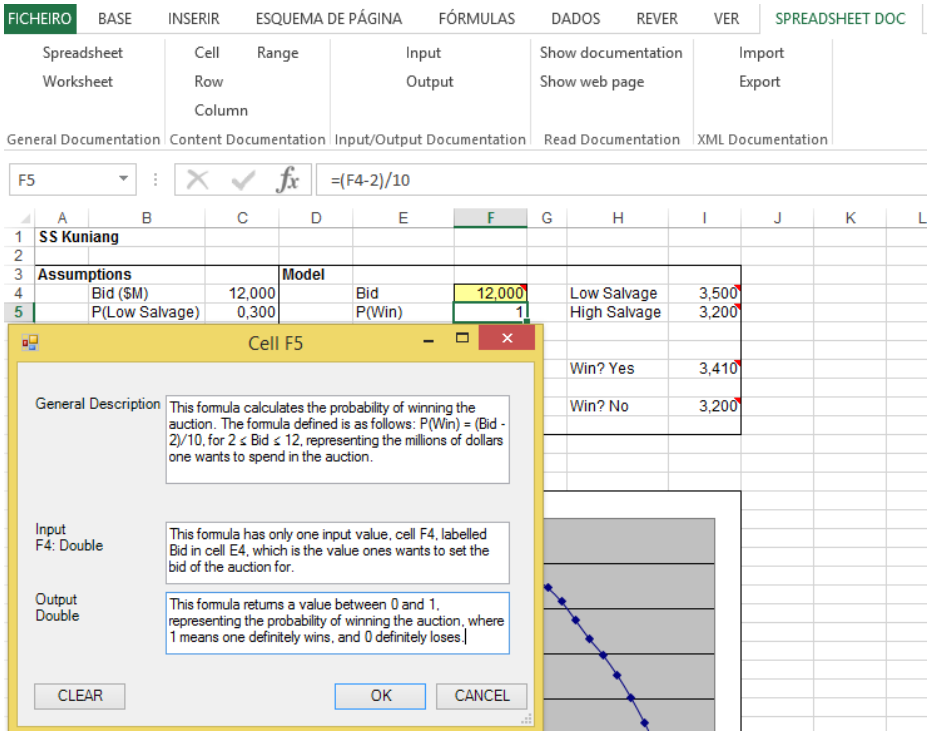

Fig. 3. Dialogue to document a formula cell.

Since we are documenting a cell, as in any other programming language, the developer should describe the computation, the input, and the output. The first text box allows the user to write a general description of the formula. This is similar to what a Java programmer starts to write when documenting a method. Next, the user can describe the argument of such formula. In this case, the input is cell F4, which is the label of the text box. Moreover, the tool also shows the type of the inputs, in this case, double. This may also help the user to detect incorrect usage of cells. Finally, the user can describe the output of the formula, again annotated with the corresponding type.

In the following sections we will describe in detail all the features of our approach, including how to describe input, output, and computations.

3 The SpreadsheetDoc Add-in

We have developed our tool, SpreadsheetDoc, as an add-in for Excel 2013. To develop our framework we used the programming language C#, version 4.0, and Visual Studio Ultimate 2013. To use our SpreadsheetDoc it is necessary to install it as an add-in within Excel. It will appear as a new ribbon with its name and when the user clicks on it he/she will have available all its functionalities. Each of these functionalities is implemented as a method. Thus, the code structure is simple allowing to easily add new methods/functionalities. Since each functionality can be used to add new documentation or to update existent one, each method must verify if some documentation already exists for the selected spreadsheet part (worksheet, cell, etc.). If no documentation exists, then a new form must be created. Otherwise, the form is loaded with the existing documentation.

Our framework is structured in three parts. The first part is where the user writes, reads, and updates all the documentation. This is done using the corresponding buttons listed in the ribbon and described in Sections 4 and 5.

The second part is the possibility of importing and exporting XML files with the documentation. Such file can be used in different ways. For once, it is used by the tool itself to create a web page where the user can read the spreadsheet documentation, possibly with links to documentation of other spreadsheets, if they are referenced.

When exported, this XML file can also be used by other tools as they wish. For instance, it can be used by other Excel add-ins to show the documentation in a different way, or by add-ins for other spreadsheet system such as LibreOffice or OpenOffice so they can open Excel spreadsheets, but also their documentation.

It is also possible to read an XML file to import documentation written in other tools. This makes it easier to exchange spreadsheet documentation. For instance, it allows the user to import a new version of the documentation the developer may have written. It may also be used to import documentation written for that spreadsheet, but using another spreadsheet software, like OpenOffice.

The third part is where the user reads the documentation on a web page. This web page is generated based on the XML file already created. Such web page can potentially be consulted by other people. For instance, inside an organization

there can be a server with all the web pages of all spreadsheets available, and users can search for some spreadsheet implementing a functionality they need.

Figure 4 illustrates the potential interactions users can have with the environment SpreadsheetDoc creates.

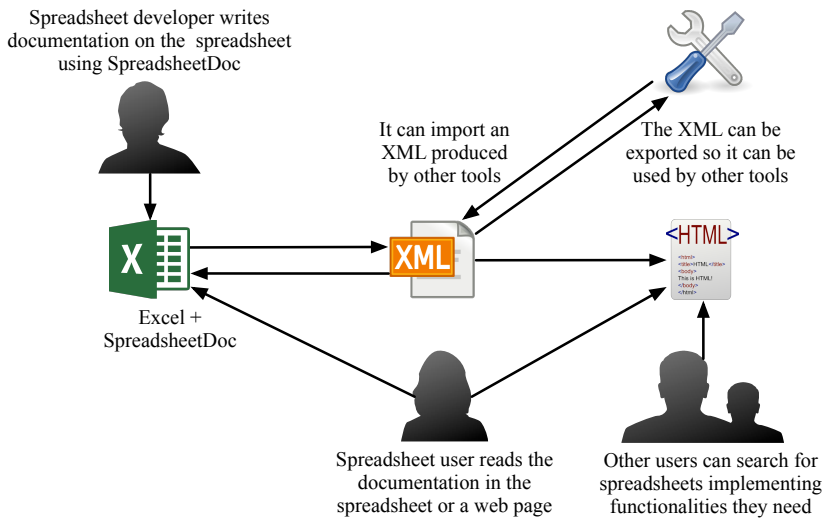

Fig. 4. The possible interactions with the SpreadsheetDoc environment.

The tool can be downloaded in the following web page:
https://bitbucket.org/spreadsheetdoc/spreadsheetdoc/.

4 Documentation for Spreadsheet Users

In this section we will describe the SpreadsheetDoc features that allow to write documentation more directed for spreadsheet end users. An end user is the person who uses the spreadsheet after it has been fully developed. Indeed, they probably do not even completely understand how it computes the results. Thus the documentation should be easy to understand. Spreadsheet end users only intend to input values and read the computed output. They can be seen as any other software end users. So, spreadsheet developers should write documentation using the next features focusing on end users. Notice that all the documentation should be written by developers. However, in some cases it is intended to be read by end user and in other cases by developers. Indeed, each of the features we present can be used to write and to read documentation, but should be used by developers to write and by users to read.

SpreadsheetDoc is composed of five different groups of functionalities: General Documentation, Content Documentation, Input/Output Documentation,

Read Documentation, and XML documentation. In the following we describe each functionality of each group.

4.1 Documenting a Spreadsheet Program

The first functionality we introduce, part of the General Documentation group, is the one to document an entire spreadsheet document.

In general an organization makes use of many spreadsheets. For instance, for the oil company Enron there were more than 15.000 spreadsheets exchanged in their emails [9]. Thus, it is important to document each spreadsheet file, so users can know each one and possibly reuse them.

So, the first documentation the user should write is about the spreadsheet itself. The "Spreadsheet" button in our add-in opens a dialogue box with a text box inside. The user can then write the spreadsheet's general purpose. In this dialogue three buttons are shown: `clear`, `ok`, and `cancel`. The `clear` button, as the name suggests, clears the text box. The `ok` button saves the dialogue box state. Finally, `cancel` drops all changes inside the dialogue box. Figure 5 shows the wizard for our running example, and the description we added.

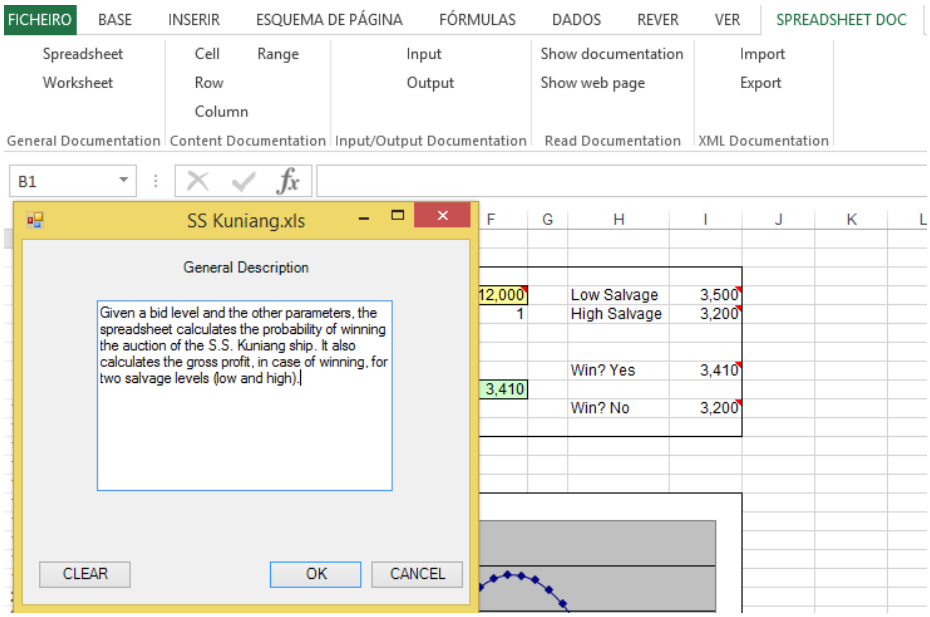

Fig. 5. The "Spreadsheet" button wizard to document a spreadsheet file.

After writing and saving the documentation, if the user clicks again in the button, the same wizard will be shown, but this time showing the recorded text. This behaviour is the same for all the functionalities.

4.2 Documenting Each Worksheet

A spreadsheet can have dozens, even hundreds of worksheets. For the EUSES spreadsheet corpus the biggest spreadsheet has 106 worksheets [8] and for the Enron 175 were found in a single spreadsheet file [9]. Thus, it is quite important to document each of these worksheets, otherwise it becomes impossible to know what each one is doing.

The "Worksheet" button, from the General Documentation group, has a structure similar to the previous button, but in this case users should document the behaviour of the worksheet and not of the spreadsheet. Inside each worksheet the user should click this button and write the corresponding documentation. It will be associated with the worksheet the user is in.

So, the documentation generated by the General Documentation group has the goal of giving users a better understanding of the spreadsheet behaviour, helping them understanding what was it developed for (Spreadsheet button) and how each of its pieces work (Worksheet button). The wizard shown is similar to the one in Figure 5, and thus we do not show it.

4.3 Documenting a Cell

The finest grain in a spreadsheet structure is a cell. Although in most cases it is not necessary to document each cell individually, some of them must be documented so one can understand how the spreadsheet works. Since for now we are focused on user documentation, the description about documentation of cells with formula is left for Section 5.

The "Cell" functionality, from the group Content Documentation, can be used to document each and every cell. If the cell is a plain value, then the wizard shown is similar to the one in Figure 5 (thus we do not show it). The documentation writer can then describe the cell content. If the content is a formula, then the description must be more technical, so it can be updated by other developers. We discuss this in Section 5.

4.4 Documenting a Column

Usually, spreadsheet developers tend to organize the data by rows or columns. For now we are going to focus on columns. Depending on the spreadsheet structure, commenting a column can be useful. Indeed, a user may document an entire column describing its behaviour (for instance, by saying the column computes the average of the columns before it). This functionality can also be used when each column of the worksheet has a particular meaning. For instance, in Figure 6 one may wander what each column represents.

In fact, this spreadsheet has another worksheet containing its documentation. For instance, it clarifies that column `N (OK)` has the value `1,00` in all its cells.

The use of this functionality makes sense only if the worksheet contains a single table. If it has more than one, commenting an entire row can be confusing.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	COM_CODE	UPC	DESCRIP	SIZE	CASE	NITEM	STORE	WEEK	MOVE	QTY	PRICE	SALE	PROFIT	OK
2	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	383	16	1	2,19		33,47	1,00
3	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	384	7	1	2,19		33,47	1,00
4	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	385	15	1	2,19		33,47	1,00
5	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	386	14	1	2,19		38,49	1,00
6	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	387	14	1	2,19		43,51	1,00
7	653,00	1111140009	DOVE DISH LIQUID	42 OZ	9,00	2851281	100	388	14	1	2,19		43,51	1,00

Fig. 6. A spreadsheet representing dishwasher detergents taken from [14].

The "Row" button opens a dialogue box with a text box inside, similar to the one presented in Figure 5 (thus we omit its illustration). This button is also part of the Contend Documentation group.

4.5 Documenting a Row

In the previous sub-section we described the documentation of a column. The dual applies for rows. In some cases, spreadsheets are developed column oriented, but in some other cases, row oriented. Indeed the example shown in Figure 6 could be organized by rows instead of columns. Thus, the documentation writer must choose the adequate functionality so the user can get the most out of the documentation.

4.6 Documenting a Range

Spreadsheets are a development framework where users have freedom to do what they want. So, it is possible (and actually quite common) to have more than one table on the same worksheet. Then, tables have different objectives and it is important to understand what is the purpose of each one. So, we created a functionality where it is possible to document a selected range of cells, that is, a rectangular selection of cells.

The "Range" button opens a dialogue box with a text box inside so the user can document the selected range. Again, this is similar to what is shown in Figure 5. This button is the last of the Content Documentation group. The documentation generated by this group focus on the understanding by users of cells, rows, columns, and ranges content.

4.7 Documenting an Input Cell

The third group, Input/Output Documentation, has two functionalities: documenting input cells, and documenting output cells.

The "Input" button opens a dialogue with two text boxes inside. The first text box is in read-only mode and shows a list of all input cells and corresponding type, as given by Excel (for instance, Double). As we said, it is read-only and users cannot modify it. After the user adds a new input cell, the list is updated. The second text box allows the user to freely document the cell. A cell is added to the list after the user clicks `ok`. In such a dialogue four more buttons are sown: `clear`, `ok`, `cancel`, and `remove`. The three first buttons act as described before.

The `remove` button allows the user to remove the current input cell from the list. This is only possible if the cell is in the list already.

Note that it is possible to document a cell using the "Cell" and the "Input" functionality simultaneously. The fact that the cell is an input point is important for end users, and thus it should be documented as such. However, the spreadsheet developer may feel the need to add more technical details to such cell, which may not be of interest for end users, but only for a future developer.

4.8 Documenting an Output Cell

Output cells are where the user usually sees the results produced by the other cells. Thus, these are probably the most important cells for end users.

Similar to "Input", the "Output" button opens a dialogue with two text boxes inside, one (in read-only mode) showing a list with all the output cells (name and type) of the spreadsheet (top part of Figure 7), and a text box to describe the current selected output cell (bottom part of Figure 7). For each output cell, it is also shown its type, as given by Excel. In Figure 7 we show the wizard for this case, documenting a cell of our running example.

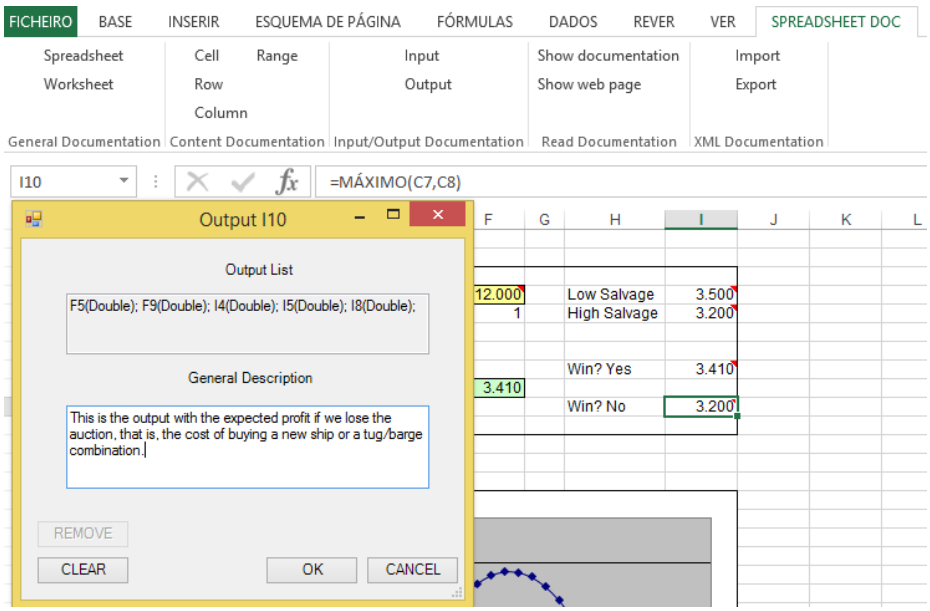

Fig. 7. Dialogue to document an output cell.

Again, notice that output cells may also be documented as cells to explain more technical details about its content (see Section 5).

4.9 Show Cell Documentation

The fourth group, Read Documentation, has two functionalities: one to read the documentation on the spreadsheet, and another to read it on a web page.

The "Show documentation" functionality opens a dialogue showing the user the selected cell documentation. This can be documentation either from the "Cell" functionality, from the "Row" functionality, or from any other content documentation. This allows the user to see the complete documentation of a cell with a single click.

This button has two possible states: enabled and disabled. If the button's state is disable it represents that there is no documentation for the selected cell. Otherwise, it is possible to click on it and read the existent documentation. Nevertheless, the user can always click on each individual functionality to see the corresponding documentation.

4.10 Show Documentation Web Page

The last user documentation functionality, is also the last one from the Read Documentation group. The "Show web page" button opens a web page showing all spreadsheet's documentation. The web page is created locally using the documentation previously written. This allows to show the documentation in a more appealing fashion. Moreover, references to cells are links that can be clicked to read the corresponding documentation. In fact, if the spreadsheet references other spreadsheets, and they have documentation, it can also be read.

5 Documentation for Developers

Spreadsheet developers are the ones designing, implementing, and documentation the spreadsheet. When they write documentation, they should know the target people that will use their spreadsheet [16]. Indeed, they should distinguish documentation for the spreadsheet users and (future) maintainers. The documentation for developers may have complex details and technical concepts. The SpreadsheetDoc feature we now describe allows developers to write documentation for cells they consider complex, for instance, cells with formulas.

Indeed a spreadsheet can have thousands of cells: for the EUSES spreadsheet corpus the biggest spreadsheet has 889.952 [8], and for the Enron 113.134 [9]. With our framework it is possible to document each cell individually (although in many cases this is not necessary). The developer must decide which ones deserve to be documented.

We have previously described the "Cell" functionality to document a particular cell, in the context of end users. Such functionality can also be used to described formula cells. The "Cell" button opens a dialogue that is contextualized with the cell content. If the cell contains a plain value (a number or a string), only a text box is presented as we presented in Section 4.3. On the other hand, if the cell content is a formula, at least three different text boxes must be filled in, as shown in Figure 3:

1) The first text box is for the user to write a small description of the selected cell. The next text boxes are used to describe the input.

2) For each input, that is, for each reference or range in the formula, a text box is presented so the user can describe such input. Each text box has a label on the left showing two possible options:

   a) If the input is a cell range, such range is shown, so the user knows which cells he/she is describing. The range type is also shown. The type of a range however must be computed by our tool as Excel does not have such information. If all the cells have the same type, then such type is presented. Otherwise, we compute the type represented in more cells and present that type, showing the remaining types and corresponding cells.

   b) If the input is a reference to a single cell, then the tool presents its name (reference) and type as given by Excel. This can be seen in Figure 3 for our running example.

3) Finally, the last text box is used to describe the output generated by the cell. Its label is the type of the cell.

This documentation process can be compared to the JavaDoc tool where users document their methods. In this case, developers document formulas. With JavaDoc the user writes a general description of the method, our first box, describes each method argument, our following boxes, and finally describes the return of the method, our last box.

6 Related Work

In [19] the authors present a system to format spreadsheet documentation. This system uses an external editor to document spreadsheets and macros to format such text. This can be compared to literate programming, where documentation and code are kept together in the same file. The objective is to easily write and update documentation for spreadsheets. However, the user must "program" the documentation itself, for instance, writing the following code to create a variable to document a formula: `@MACRO(xfor(v)=[@P(@V(f_@X(v)))])`. These variables can then be assigned to parts of the spreadsheet and used in the text documentation. However, this seems difficult to learn, specially for end users. SpreadsheetDoc on the other hand shows contextualized wizards with the necessary text inputs the documentation writer must fill in, making it more convenient for end users to document their spreadsheets.

Raymon did a study showing that a good documentation improves mainly two factors: effectiveness and efficiency of users [16]. So, our framework intends to improve these two factors. Raymon describes that users usually ask to colleagues when they do not understand the system they are working with. Others spend many time understanding what should be done. Thus, users usually lose efficiency and effectiveness, increasing the losses of their corporations. Our framework aims to improve the usability of user when writing documentation. For instance, it is possible to send spreadsheets to other users without the need to explain it

because it is already documented. Thus, users will have less difficulties working on spreadsheets that were not created by them.

In [20] the author discusses for two kinds of documentation: development documentation and user documentation. The former is about the software itself, its internal form, and is created for developers, with technical knowledge about the software and its implementation details. The later is for the software users, with possibly no technical skills to understand documentation for developers. We have also separated both these kinds of documentation in SpreadsheetDoc. On one hand formulas should be documented technically, that is, with enough technical detail so they can be updated by other developers. On the other hand, input and output cells should be documented for end users so they can know where to input their values and read the results.

Abraham et al. developed a framework called UCheck [1]. This framework can compute the labels that affect each cell. Thus, when using the *headers* functionality the tool displays arrows directed from the header cells (labels) to the target cells, that is, the ones labelled by such text. The *units* functionality runs the unit checker and the system marks the cells with unit errors. For instance, a cell that adds cells with label apples and cells with label oranges is probably wrong. This work can also be seen as an automatic way of documenting spreadsheets, as it more explicitly shows information about the relationship of cells and their labels. It is interesting and relevant to our work because, by automatically inferring the headers, we could use this to show to users the correspondent labels of selected cells. Thus, we intend to integrate the inference system in our framework providing more information to users when they document spreadsheets, and specially when they read it. The more information we can provide to users, the more easily they will understand the spreadsheet they are working on.

7 Conclusions

Spreadsheets are the most used programming environments in the world. However, they lack many of the features modern programming environments offer. In particular, there is no structured way of documenting spreadsheet programs. Indeed, there is strong evidence that users wast too much time trying to understand spreadsheets, specially when they are using one that was not created by them. They search within spreadsheets trying to find some kind of documentation, ask for help to colleagues, or end up by quitting.

To alleviate this scenario, in this paper, we have presented a tool, SpreadsheetDoc, to document spreadsheets. It is built as an Excel add-in so users can easily install and use it. For each part of the spreadsheet SpreadsheetDoc offers the correct wizard with the necessary fields to be filled in by the user. The user can then read the documentation within the spreadsheet, but also in a web page generated from such documentation. It is also possible to export the documentation via an XML file so it can be reused by other tools. In fact, it is also possible to import documentation from an XML file.

7.1 Future Work

The work we present in this paper is the very first step towards easing the creating of documentation for spreadsheets. There are however many important directions for future work.

The most important one is the validation of our approach. Although we believe that it can help users to be more productive, we do not yet have the empirical evidence of such fact. In fact, there are at least two kinds of evaluations we intend to do. First, we intend to evaluate the usability of our tool, trying to learn if end users understand the purpose of each of its parts, and can actually use it in the correct way. Second, we will investigate its impact in users productivity, that is, given the same spreadsheet, one documented with our tool, and another not documented, or even documented with the current ad-doc techniques, investigate if users more easily and faster can understand the spreadsheet documented with SpreadsheetDoc. These evaluations will be performed with empirical studies, which we will design and run based on our previous experience [2, 3, 6].

Another quite interesting and promising direction is the automation or inference of documentation. As we mentioned, we intend to integrate the inference of headers and units [1], which will automatically give some documentation for users. For instance, for our running example, we would like to automatically document the input of cell F5 as being `Bid`, and not as being cell F4 (as `Bid` it the label of cell E4), since the label is much more informative than the cell reference. Moreover, we would also like to integrate some heuristics to automatically describe existing formulas, in a similar way as described in [17] for mining business rules from spreadsheets. For instance, for our running example, the system could automatically infer the following description for cell I10: *Cell I10 calculates Win? No, that is, the maximum between Profit new ship and Profit tug/barge.* This can easily be inferred from the formula, its inputs, and corresponding labels. Again, we have some experience in automatically inferring information from spreadsheet [4, 5], which will help us succeed in this work.

In this first approach we did not work on documenting the visual basic for applications (VBA) scripts that are part of some spreadsheets. Although this may seem important, in the Enron's corpus only 47 spreadsheets (out of more than 15.000) used VBA scripts [9]. Also in the EUSES corpus only 126 (out of 4.498) used VBA [8]. Nevertheless, we will also address this in future work. Since in this case we are probably addressing more advanced users, we intend to follow an approach similar to JavaDoc, where the programmer annotates the different parts of the source code, and from which it is possible to generate a comprehensive web page. This will extend the web page generated by our tool.

References

1. R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. In *Proc. of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing*, pages 165–172, Washington, DC, USA, 2004. IEEE.

2. L. Beckwith, J. Cunha, J. P. Fernandes, and J. Saraiva. An empirical study on end-users productivity using model-based spreadsheets. In S. Thorne and G. Croll, editors, *Proc. of the EuSpRIG Annual Conference*, pages 87–100, 2011.
3. L. Beckwith, J. Cunha, J. P. Fernandes, and J. Saraiva. End-users productivity in model-based spreadsheets: An empirical study. In M. Costabile, Y. Dittrich, G. Fischer, and A. Piccinno, editors, *Proceedings of the Third International Symposium on End-User Development*, pages 282–288, Heidelberg, June 2011. Springer.
4. J. Cunha, M. Erwig, J. Mendes, and J. Saraiva. Model inference for spreadsheets. *Journal of Automated Software Engineering (ASE)*, pages 1–32, 2014. in press.
5. J. Cunha, M. Erwig, and J. Saraiva. Automatically inferring ClassSheet models from spreadsheets. In *Proc. of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 93–100, Washington, DC, USA, 2010. IEEE.
6. J. Cunha, J. P. Fernandes, J. Mendes, and J. Saraiva. Embedding, evolution, and validation of spreadsheet models in spreadsheet systems. *IEEE Transactions on Software Engineering*, 43(3):241–263, 2014.
7. K. A. de Graaf, P. Liang, A. Tang, and H. van Vliet. Supporting architecture documentation: A comparison of two ontologies for knowledge retrieval. In *Proc. Int. Conf. on Evaluation and Assessment in Soft. Eng.*, pages 3:1–3:10. ACM, 2015.
8. M. Fisher and G. Rothermel. The EUSES spreadsheet corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. In *Proc. of the First Workshop on End-user Soft. Eng.*, pages 1–5. ACM, 2005.
9. F. Hermans and E. Murphy-Hill. Enron's spreadsheets and related emails: A dataset and analysis. In *Proceedings of the International Conference on Software Engineering (ICSE'15)*, Firenze, Italy, May, 2015. (to appear).
10. F. Hermans, M. Pinzger, and A. van Deursen. Breviz: Visualizing spreadsheets using dataflow diagrams. *CoRR*, abs/1111.6895, 2011.
11. F. Hermans, M. Pinzger, and A. van Deursen. Supporting professional spreadsheet users by generating leveled dataflow diagrams. In *Proc. of the 33rd International Conference on Software Engineering*, pages 451–460. ACM, 2011.
12. B. Klimt and Y. Yang. Introducing the Enron corpus. In *CEAS 2004 - First Conference on Email and Anti-Spam*, 2004.
13. J. Lawrance, R. Abraham, M. M. Burnett, and M. Erwig. Sharing reasoning about faults in spreadsheets: An empirical study. In *2006 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 35–42. IEEE, 2006.
14. S. G. Powell and K. R. Baker. *The Art of Modeling with Spreadsheets*. John Wiley & Sons, Inc., New York, NY, USA, 2003.
15. K. Rajalingham, D. R. Chadwick, and B. Knight. Classification of spreadsheet errors. *CoRR*, abs/0805.4224, 2008.
16. J. G. Raymond. Audience identification for end user documentation. In *Proceedings of the June 7-10, 1982, National Computer Conference*, AFIPS '82, pages 281–285, New York, NY, USA, 1982. ACM.
17. S. Roy. Business rule mining from spreadsheets. In F. Hermans, R. F. Paige, and P. Sestof, editors, *Proceedings of the Second Workshop on Software Engineering Methods in Spreadsheets*, volume 1355 of *SEMS '15*, pages 5–6. CEUR, 2015.
18. J. E. Scott. Technology acceptance and erp documentation usability. *Commun. ACM*, 51(11):121–124, Nov. 2008.
19. R. M. Snyder. A system for automating the update of spreadsheet documentation. *J. Comput. Sci. Coll.*, 23(2):163–169, Dec. 2007.
20. B. van Loggem. Software documentation: A standard for the 21st century. In *Proceedings of the International Conference on Information Systems and Design of Communication*, ISDOC '14, pages 149–154, New York, NY, USA, 2014. ACM.