BumbleBee: A Refactoring Environment for Spreadsheet Formulas

Felienne Hermans
Delft University of Technology
Mekelweg 4
Delft, the Netherlands
f.f.j.hermans@tudelft.nl

Danny Dig
Oregon State University
2500 NW Monroe Ave
Corvallis, OR, US
digd@eecs.oregonstate.edu

ABSTRACT

Spreadsheets are widely used in industry. It is estimated that end-user programmers outnumber regular programmers by a factor of 5. However, spreadsheets are error-prone: several reports exist of companies that have lost big sums of money due to spreadsheet errors. In previous work, *spreadsheet smells* have proven to be the cause of some of these errors.

To that end, we have developed a tool that can apply refactorings to spreadsheet formulas, implementing our previous work on spreadsheet refactoring, which showed that spreadsheet formula smells are very common and that refactorings for them are widely applicable and that refactoring them with a tool is both quicker and less error-prone.

Our new tool Bumblebee is able to execute refactorings originating from both these papers, by means of an extensible syntax, and can furthermore apply refactorings on entire groups of formulas, thus improving upon the existing tool RefBook. Finally, BumbleBee can also execute transformations other than refactorings.

Categories and Subject Descriptors

H.4.1 [**Information Systems Applications**]: Office Automation—*Spreadsheets*

General Terms

Experimentation, Languages

Keywords

spreadsheets, transformation, end-user programming

1. INTRODUCTION

Spreadsheets are very important to today's society: it is estimated that end-users outnumber professional developers[3]. Their use is diverse, ranging from inventory administration to educational applications and from scientific modeling to financial systems, a domain in which their use is particularly prevailing. Panko [4] estimates that 95% of U.S. firms, and 80% in Europe, use spreadsheets in some form for financial reporting.

However, the use of spreadsheets is not without problems: Panko [5] studied seven different field audits into spreadsheet errors and showed that 86% of spreadsheets contain at least one error.

In previous work [13], we have seen that spreadsheet users are quite similar to software developers. They struggle with similar problems, such as under-documented, long living artifacts that switch owner frequently, diminishing understandability. One of those problems is the occurrence of 'spreadsheet smells', very similar to the smells described by Martin Fowler [14].

In order to relieve code smells, the idea of refactoring was introduced. A refactoring is a change to source code that improves the quality, but does not change its behavior. In this paper we describe BumbleBee, a tool that is able to execute refactorings and other transformations on spreadsheet formulas.

In addition to performing the transformations, we also provide a method to locate formulas in which transformations could be applied, thus helping the user to find formulas that could be refactored. BumbleBee is available for download[1].

2. BACKGROUND

In previous work, we have seen that end-users understand smells in spreadsheets [1, 13] and that they prefer refactored versions of formulas [2]. This lays ground for a more research on spreadsheet formula refactoring.

In our previous spreadsheet refactoring implementation Refbook [2], the refactoring rules are programmatically defined. This means they are not easy to adapt or extend, not by the tool's designers, let alone by the tool's users.

With our new tool BumbleBee, we take a more general approach, in which the refactorings can be described in a language, which is based on spreadsheet formula syntax. This enables us to describe a large st of refactorings and also allows spreadsheet users to modify and write their own version of refctorings.

3. TRANSFORMATION LANGUAGE

In this section we describe the BumbleBee grammar, the language that we use to describe transformations to be ap-

[1]http://www.felienne.com/BumbleBee

FSE'14 , November 16–22, 2014, Hong Kong, China

plied on spreadsheet formulas. This language builds upon the grammar for Excel formulas [8], which we adapt slightly by modifying and adding a few production rules.

Firstly, a transformation rule in the BumbleBee grammar consists of two Excel formulas combined with a "↔", indicating that those two formulas may be transformed into each other. An example of this is

A1+A2+A3 ↔ SUM(A1:A3).

Furthermore, our language adds a selected set of variables to indicate a transformation is limited to a certain formula construct. F represents a Formula, R represents a Range, C represents a Cell and P represents a constant. These transformations can be applied if the formula it is applied to has exactly that construct on the place of the variable. For example

SUM(R)/COUNT(R) ↔ AVERAGE(R)

This rule indicates transforming a SUM over any range divided by a count over the same range into the average of that range.

Furthermore, the BumbleBee grammar permits parametrized references to cells. This allows for variables in places where the original Excel grammar only contains cells. These variable cell references have the form {i,j} An example of this is

{i,j} + {i,j+1} ↔ SUM({i,j}:{i,j+1})

This indicates that all formulas that add two cells, whose columns are the same and rows differ by one, can be rewritten into a SUM and vice versa.

Finally, we allow for referencing cells in connected groups. This expression represents all cells between the arguments before and after it. An example of that is

{i,j} + ... + {m,n} ↔ SUM({i,j}:{m,n})

With this rule, we could, for instance, transform A1+A2+ B1+B2 into SUM(A1:B2). Note that, while we list transformations that are refactorings here, our BumbleBee grammar can just as well be used to describe transformations which are not behavior preserving.

4. DESCRIBING REFACTORINGS

Now that we have defined the transformation language, we use it to describe refactorings from our previous work, showing that BumbleBee is able to express them [1, 2]. Note that the language as we have currently defined it only supports *intra-formula* refactorings. These are refactorings which take place within one cell, such as A1+A2 ↔ SUM(A1:A2).

The counterpart of intra-formula refactorings are *inter-formula refactorings*: refactorings that result in changes to multiple cells. An example of this is the 'extract column' refactoring in [2], with which part of a formula is placed in a new cell. The implementation of intra-formula refactorings is left for future work.

The remainder of this section describes how refactorings from our earlier work can be expressed in BumbleBee.

4.1 Replace Awkward Formula

The 'replace awkward formula' refactoring aims to replace a complex formula with a built-in function in order to simplify it. In [2] two such 'awkward formulas' transformations were described: refactoring plus into SUM and times into PRODUCT. With the BumbleBee grammar, we can expand the set of transformations to other commonly used Excel functions.

To demonstrate the expressiveness of the BumbleBee grammar, we will use it to describe the ten most commonly used function in the EUSES corpus [9]. This corpus contains real-life spreadsheets from 11 different domains and has been used by several researchers to evaluate spreadsheet algorithms, among which [10] and [11]. Barowy *et al.* [12] performed an analysis on the EUSES corpus to find the 10 most common functions. They are, in order of frequency: SUM, MIN, AVERAGE, MAX, PRODUCT, MATCH, OFFSET, VLOOKUP, INDEX, and CONCATENATE.

For these top ten functions, we define refactorings with our language. Some of the functions in the top ten are well known calculation functions, for which it is easy to define a corresponding refactoring:

- {i,j} + ... + {i,k} ↔ SUM({i,j}:{i,k}
- IF(F_1>F_2,F_1,F_2) ↔ MIN(F_1,F_2)
- IF(F_1>F_2,F_1,F_2) ↔ MAX(F_1,F_2)
- SUM(R_1)/COUNT(R_1) ↔ AVERAGE(R_1)
- {i,j} * ... * {m,n} ↔ PRODUCT({i,j}:{m,n}
- {i,j} & ... & {m,n} ↔ CONCATENATE({i,j}:{m,n}

Some of these functions or combinations of them are equivalent: VLOOKUP can be rewritten with a combination of INDEX and MATCH, while OFFSET can be replaced by INDEX. This leads to the following refactorings.

- INDEX(C_1:C_2, V_1, V_2) ↔ OFFSET(C_1,V_1-1,V_2-1)
- VLOOKUP(F, {i,j}:{m,n}, V) ↔ INDEX({i+V-1,j}:{i+V-1,n},MATCH(F, {i,j}:{i,n})

With this, we have expressed the 'replace awkward formula' refactoring from [2] for the most popular 10 Excel functions using the BumbleBee grammar.

4.2 Guard Call

Badame and Dig [2] furthermore describe the 'guard call' refactoring, which adds a guard to a formula to prevent it from resulting in an error. Badame and Dig only provide a refactoring to guard divisions by zero, written here in our new syntax:

- F_1/F_2 ↔ IF(F_2<>0,F_1/F_2, "Value unknown")

With the BumbleBee grammar, we can easily describe additional guard refactorings. For instance, the LOOKUP functions can result in an error when the value that was searched for has not been found. The INDEX function too can result in an error, if the values to look for are out of the bounds of the range. Therefore, adding an IFERROR around these formulas increases the robustness of the spreadsheet.

- VLOOKUP(F, C_1:C_2, V) ↔ IFERROR(VLOOKUP(F, C_1:C_2, V), "Value not found")
- INDEX(C_1:C_2, V_1, V_2) ↔ IFERROR(INDEX(C_1:C_2, V_1, V_2), "Out of bounds")

4.3 Group References

Previous work by Hermans *et al.* [1] too described intra-formula refactorings. Firstly, there is the 'group references' refactoring, which can be expressed in the BumbleBee grammar as follows:

- SUM({i,j},...,{m,n}) ↔ SUM({i,j}:{m,n})

4.4 Merge Branches

Secondly, there is the 'merge branches' refactoring that can be used to simplify conditional formulas. This transformation too is expressible in BumbleBee grammar.

- IF(F_1, F_3, IF(F_2, F_3,F_4)) ↔ IF(OR(F_1, F_2), F_3,F_4)

5. TOOL DESIGN

Our current approach for transforming spreadsheet formulas is implemented as an add-in for Excel 2010. It uses our existing spreadsheet analysis framework Breviz[6, 13] as a basis for reading, parsing and analyzing the formulas. BumbleBee is implemented in C# and F# using Visual Studio 2013.

Currently, the user interface offers the following options:

- Find applicable formulas for a selected cell
- Get a dropdownbox with the possible transformations, when selecting one, the user gets a preview of the transformed formula
- Apply this transformation in the selected range, in the entire worksheet or the entire file

When a user selects a transformation and a formula, BumbleBee parses them, and subsequently applies pattern matching to determine whether a transformation rule is applicable on a formula.

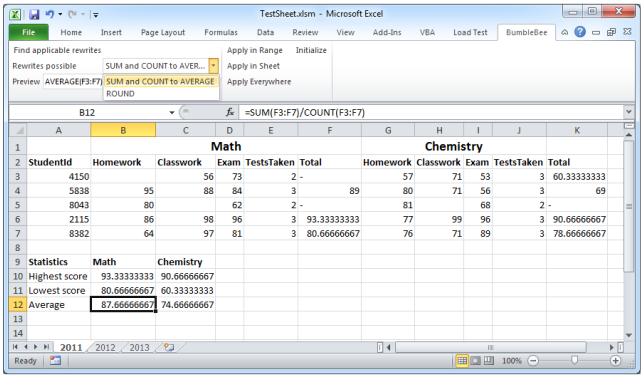

Figure 1: Screenshot of the current BumbleBee implementation in Excel 2010. The options for transformation of cell B12 are shown in the dropdown box, and the preview field shows the selected transformation applied to B12.

6. RELATED WORK

Efforts related to our research include work on source code refactoring, most prominently the work of Fowler [14].

Furthermore, there is work on the improvement of spreadsheet, such as the work on spreadsheet design guidelines. Raffensberger [15], for instance advises to merge references that occur only once. He furthermore states that unnecessary complex formulas with many operations and parenthesis should be rewritten. Rajalingham *et al.* [16] also propose guidelines to improve spreadsheet quality, which they base on principles of software engineering. Secondly, there are papers that address spreadsheets errors, like [17, 5], together with their causes. Powell *et al.* for instance [18] names conditional formulas among the top three of commonly occurring spreadsheet error categories.

There is also related work on finding anomalies on spreadsheets, for instance the work on the UCheck tool [19, 20, 21]. UCheck determines the type of cells, and locates possible anomalies based on this type system. UCheck uses a similar visualization, with colors in the spreadsheet, to indicate found anomalies. Their follow up work on debugging of spreadsheets [22] also suggests corrections for errors and is as such related to our current research, as it is focused on maintaining existing spreadsheets.

Spreadsheet testing too has been a subject of interest for some time. Most prominently, there is the "What You See Is What You Test" methodology by Rothermel *et al.*, who have created [23] and subsequently validated [24] a method to support end-users in defining and analyzing tests for spreadsheets.

This paper can be seen as overarching both [13] and [2], because it is a more general method for formula transformation: refactorings can be described with rules and are not embedded in the tool. Furthermore, the BumbleBee approach is not necessarily aimed at refactoring, but can also be applied to other transformations, such as migration or changing business rules.

7. DISCUSSION

The current implementation of BumbleBee, while still a prototype, supports spreadsheet users in updating their spreadsheet formulas in a consistent way. In this section, we discuss a variety of issues that affect the applicability and suitability of the proposed approach.

7.1 Transformations Impacting Multiple Cells

The BumbleBee grammar as it is currently defined, only supports changes within one formula. While this is certainly useful, it would be even better to extend the BumbleBee grammar to be able to also transform formulas over multiple cells. This, of course, has its challenges, as inserting cells might have consequences for the entire spreadsheet.

7.2 Definition of the Transformation Rules

In the current approach, we as language builders have also defined the transformation rules. While this is useful for generic transformations, such as refactoring or migration formulas, we cannot provide rules for all possible changing business rules. BumbleBee allows for users to describe their own rules, but since spreadsheet users are not professional trained developers, only a very small set of 'power users' will be able to do so. This diminishes the applicability of our tool, specifically for the scenario in which business rules

change. One of the plausible solutions for this is to derive transformation rules from changes a user makes to a spreadsheet. This is one of the most useful directions for future work we see.

8. CONCLUDING REMARKS

This paper describes BumbleBee: a tool that can be used to define and execute transformations on spreadsheet formulas. We have used the newly defined language to describe all refactorings in our previous work, showing it is as expressive. Furthermore, BumbleBee can be used for other types of transformations, including migrating spreadsheets to new version of Excel and updating formulas in case of changing requirements.

The key contributions of this paper are as follows:

- A language to describe spreadsheet formula transformations (Section 3)
- The demonstrated use of this language to describe refactorings (Section 4)
- An implementation into the BumbleBee tool (Section 5)

The current research gives rise to several directions for future work. Firstly, empirical studies on the usefulness of our approach is needed, especially in regards to benefits for error reduction. Also, more studies are needed to test the applicability of BumbleBee on industry-sized spreadsheets. Secondly, it would be useful to expand the BumbleBee grammar to incorporate transformations which impact multiple cells and transformations that modify the structure of a spreadsheet. Finally, a method to automatically extract the transformation rules from edits by users would greatly improve usability of BumbleBee, as rules will not have to be entered manually anymore.

9. REFERENCES

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

[2] S. Badame and D. Dig, "Refactoring meets spreadsheet formulas," in *Proc. of ICSM '12*, 2012, pp. 399–409.

[3] C. Scaffidi, M. Shaw, and B. A. Myers, "Estimating the numbers of end users and end user programmers," in *Proc. of VL/HCC '05*, 2005, pp. 207–214.

[4] R. Panko, "Facing the problem of spreadsheet errors," *Decision Line*, vol. 37, no. 5, 2006.

[5] R. R. Panko, "What we know about spreadsheet errors," *Journal of End User Computing*, vol. 10, no. 2, pp. 15–21, 1998.

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

[7] J. Sajaniemi, "Modeling spreadsheet audit: A rigorous approach to automatic visualization," 2000.

[8] S. Badame, "Refactoring meets spreadsheet formulas," Master's thesis, University of Illinois at Urbana-Champaign, 21012.

[9] M. Fisher and G. Rothermel, "The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, 2005.

[10] R. Abraham and M. Erwig, "Inferring templates from spreadsheets," in *Proc. of ICSE '06*, 2006, pp. 182–191.

[11] J. Cunha, J. Saraiva, and J. Visser, "Discovery-based edit assistance for spreadsheets," in *Proc. of VL/HCC '09*, 2009, pp. 233–237.

[12] D. W. Barowy, D. Gochev, and E. D. Berger, "Data debugging," University of Massachusetts, Amherst, Tech. Rep. UM-CS-2012-033.

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

[14] M. Fowler, *Refactoring: improving the design of existing code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[15] J. Raffensperger, "New guidelines for spreadsheets," *International Journal of Business and Economics*, vol. 2, pp. 141–154, 2009.

[16] K. Rajalingham, D. Chadwick, B. Knight, and D. Edwards, "Quality control in spreadsheets: a software engineering-based approach to spreadsheet development," in *Proc. HICSS '00*, 2000, pp. 133–143.

[17] Y. Ayalew, M. Clermont, and R. T. Mittermeir, "Detecting errors in spreadsheets," in *Proc. of EuSpRIG '00*, 2000, pp. 51–62.

[18] S. Powell, K. Baker, and B. Lawson, "Errors in operational spreadsheets: A review of the state of the art," in *Proc. of HICCS '09*. IEEE Computer Society, 2009, pp. 1–8.

[19] R. Abraham and M. Erwig, "Ucheck: A spreadsheet type checker for end users," *Journal of Visual Languages and Computing*, vol. 18, pp. 71–95, 2007.

[20] C. Chambers and M. Erwig, "Automatic detection of dimension errors in spreadsheets," *Journal of Visual Languages and Computing*, vol. 20, pp. 269–283, 2009.

[21] M. Erwig, "Software engineering for spreadsheets," *IEEE Software*, vol. 26, pp. 25–30, September 2009.

[22] R. Abraham and M. Erwig, "Goaldebug: A spreadsheet debugger for end users," in *Proc. of ICSE '07*, 2007, pp. 251–260.

[23] G. Rothermel, "Testing strategies for form-based visual programs," in *Proc. of ISSRE '97*, 1997, pp. 96–107.

[24] K. J. Rothermel, C. R. Cook, M. M. Burnett, J. Schonfeld, T. R. G. Green, and G. Rothermel, "Wysiwyt testing in the spreadsheet paradigm: an empirical evaluation," in *Proc. of INCSE '00*, 2000, pp. 230–239.

[25] M. Burnett, A. Sheretov, B. Ren, and G. Rothermel, "Testing homogeneous spreadsheet grids with the "what you see is what you test" methodology," *IEEE Trans. Softw. Eng.*, vol. 28, no. 6, pp. 576–594, June 2002. [Online]. Available: http://dx.doi.org/10.1109/TSE.2002.1010060

[26] F. Hermans, M. Pinzger, and A. van Deursen, "Automatically extracting class diagrams from spreadsheets," in *Proc. of ECOOP '10*, 2010, pp. 52–75.