BumbleBee: A Transformation 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 which have lost money because of spreadsheet errors. We assert that a contributing factor to these problems is the difficulty of consistent editing of spreadsheet formulas.

In this paper, we observe the occurrence of *copy-equivalent regions* in spreadsheets, non-connected regions with similar formulas within one spreadsheet. These regions occur frequently in practice. Therefore, we design a strategy to consistently transform them, by presenting a grammar with which formula transformations can be described. We implemented these transformations in our tool BumbleBee, which is an Excel add-in that consistently applies transformations to spreadsheet formulas.

To evaluate the usefulness of our approach, we perform an evaluation that shows that 1) our transformation tool is necessary, because a vast majority of spreadsheets with formulas (over 70%) contain similar formulas in non-connected regions, 2) the BumbleBee grammar is expressive, as it can be use to express all refactorings in previous work on spreadsheet formula refactoring, as well as all migrations to update formulas to Excel 2010, and 3) that spreadsheet users perform changes to spreadsheet formulas more efficient using BumbleBee.

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 [1] and furthermore it has been reported that 90% of all computers have Excel installed [2]. 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 [3] 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 [4] studied seven different field audits into spreadsheet errors and showed that 86% of spreadsheets contain at least one error. In previous work [5], we ourselves have found that one of the problems with spreadsheets is that they are frequently transferred from one employee to another, and that the receiver of the spreadsheet often struggles with fully understanding its calculations.

One of the solutions we have proposed in [5] is to support users in understanding spreadsheets by means of dataflow diagrams. In this paper, we aim to take the next step in supporting users who work with large and complex spreadsheets, by providing them with better support to update their spreadsheets in an easier and less error-prone way.

In our work with spreadsheets, we have seen that many spreadsheets contain similar formulas in different places. Consider, for instance, a formula to calculate corporate tax. In the Netherlands, this tax is 20% below 200,000 Eur and 25% above that. So a spreadsheet formula to calculate this tax could be IF(B7<200, B7*1.2, B7*1.25), where the data is entered in thousands. This formula would likely occur multiple times in the spreadsheet, for every year and for every legal entity, for example.

Now suppose tax law changes, resulting in the addition of a second threshold: 20% will only be calculated below 100,000 Eur. Between 100,000 and 200,000 Eur, the corporate tax is 22% and above 200,000 Eur still 25% is paid. This can be calculated with the formula IF(B7<200,IF(B7<100,B7 1.2, B7*1.22),B7*1.25). Now, the tax formula has to be edited everywhere in the spreadsheet, and this is a not an edit that would be simple to perform with find-and-replace. Excel does support the use of wild cards in the 'find' field, so we could search for IF(*<200,*~*1.2,*~*1.25), where ~ indicates a literal value. However, the wild cards are not necessarily matched to the same values, so this search operation would also match with IF(B7<200, B8*1.20, B9*1.25), for example. Furthermore, wild cards are not supported in the 'replace' field. Therefore, the find-and-replace operation

does not suffice to update this spreadsheet.

This is exactly the problem we address in this paper. We describe a method for the consistent transformation of formulas, by means of transformation rules that describe how to transform a given formula.

An example of a rule in our transformation grammar—fully described in Section 3—for the corporate tax example is

IF({i,j}<200, {i,j}*1.20, {i,j}*1.25) ↔
IF({i,j}<200, IF({i,j}<100, {i,j}*1.2, {i,j}*1.22), {i,j}*1.25)

The transformation rules are implemented in a tool called BumbleBee, which lets spreadsheet users apply the rules on a selected range of cells, or within an entire worksheet or spreadsheet. This enables consistent updating of a spreadsheet, as no occurrences of the formula will be forgotten by mistake.

In addition to changing business rules, as in the tax example, our method can also be used to refactor spreadsheet formulas, and as such it can be seen as a generalization of our earlier work on spreadsheet formula refactorings [6] and [7].

To evaluate the usefulness of our approach, we perform an evaluation that shows that 1) our transformation tool is necessary, by analyzing a large set of spreadsheets and showing that the vast majority of spreadsheets with formulas (72%) contain similar formulas in different places in the spreadsheet and 2) that the BumbleBee grammar is expressive, by showing it can express all refactorings in both [6] and [7], as well as all migrations to update formulas to Excel 2010, and 3) that spreadsheet users perform changes to spreadsheet formulas more efficient using our transformation tool.

The contributions of this paper are as follows: A grammar to describe spreadsheet formula transformations, an implementation of this grammar into a tool, two application examples and finally an evaluation of the necessity, expressiveness and impact of BumbleBee.

2. BACKGROUND AND MOTIVATING EXAMPLE

	A	B	C	D	E	F	G	H	I
1		Course A				Course B			
2	StudentId	Homework	Classwork	Exam	Total	Homework	Classwork	Exam	Total
3	4150	96	56	73	=B3+C3+D3	80	84	52	=F3+G3+H3
4	5838	95	88	84	=B4+C4+D4	67	51	67	=F4+G4+H4
5	8043	80	79	62	=B5+C5+D5	68	77	90	=F5+G5+H5
6	2115	86	98	96	=B6+C6+D6	63	53	89	=F6+G6+H6
7	8382	64	97	81	=B7+C7+D7	94	90	51	=F7+G7+H7
8									

Figure 1: A spreadsheet with non-connected equivalent regions, depicted here with 'Show Formulas' option enabled.

The spreadsheet depicted in Figure 1 is an example of a spreadsheet on which our approach is aimed, as both column E and column I calculate the sum of the three cells to the left. We call this *copy-equivalent formulas in non-connected regions*. Copy-equivalence between two formulas F and F_1 is defined in [8] as F and F_1 having the same *relative R1C1 notation*. In the R1C1 notation, cells are denoted by the letter R followed by the row number concatenated with the letter C followed by the column number. The cell A2 in regular notation is denoted with R2C1—the cell in the second row of the first column—in the R1C1 notation. By putting the row or column numbers between square brackets, a row or column relative to the current cells in indicated. In this *relative* R1C1 notation, the cells in Figure 1 in E3:E7 and I3:I7 are all written as (RC[-3]+RC[-2]+RC[-1])/3. Therefore, this spreadsheet contains two copy-equivalent regions.

If a user wanted to make a change to those formulas, for instance, applying the 'replace awkward formula' refactoring from [7] to transform B1+B2+B3 and F1+F2+F3 into SUM(B1:B3) and SUM(F1:F3) respectively, he would have to do this in both regions separately, resulting in double work. Furthermore, a user applying this refactoring to column E might not be aware that he can or should apply it to column I too, possibly resulting in inconsistencies.

This is a scenario where BumbleBee assists the spreadsheet user, as it not only lets the user perform transformations, but also supports him applying that transformation consistently throughout the spreadsheet. These transformations do not necessarily have to be behavior preserving. For instance, if the weighing of the exam changes and column E has to be changed into B1+B2+(2*B3), BumbleBee can subsequently apply this change everywhere.

3. TRANSFORMATION GRAMMAR

In this section we describe the BumbleBee grammar, the language that we use to describe transformations to be applied on spreadsheet formulas. This grammar is an extension of the grammar for Excel formulas [9], which we adapt slightly by modifying and adding production rules.

```
(skip) ::= [ \t\r\n]+  # Define whitespace
<Formula> ::= <Expression> | "{" <Expression> "}"
<Expression> ::=  Operator* <ExpressionPrimitive> <
    AnotherExpression>?
<ExpressionPrimitive> ::= <Primitive> | <Function> | <
    CellReference> | <RangeReference> | <Error> | "(" <Expression
    > ")"
<AnotherExpression> ::= Operator <Expression>
#Primitive Data types
<Primitive> ::= token:Number | token:Boolean | token:String
Number ::= [0-9]+ ("." [0-9]+)? "%"?
Boolean ::= "true" | "false" | "TRUE" | "FALSE"
String ::= "\"" ([^\"]|"\"\"")* "\""
Operator ::= "<" | ">" | ">=" | "<=" | ">=" | "<=" | "<>" | "="
    | "+" | "-" | "/" | "*" | "^" | "&"
<Error> ::= "#REF!"
#Referencing cells
<CellReference> ::= <ReferencePrefix> <Cell>
<ReferencePrefix> ::= <Error>? workbook:Workbook? sheet:Sheet?
<RangeReference> ::= start:<CellReference> ":" end:<
    CellReference>
Workbook ::= "'"? "[" ^"]" "'"?
Sheet ::= "'" [^'!]* "'" "!" | [A-Za-z][.A-Za-z0-9]+ "!"
<Cell> ::= isColumnFixed(bool):Dollar? Column isRowFixed(bool):
    Dollar? row:Number?
Dollar ::= "$"
Column ::= [0-9A-Za-z_#!]* [A-Za-z_#!]
#Functions
<Function> ::= FunctionStart args:<Expression>* (separated-by)
    Comma ")"
FunctionStart ::= [A-Za-z][A-Za-z0-9_]* [ \t]* "("
Rparen ::= ")"
Comma ::= ","
```

Figure 2: Ludwig Grammar for Excel formulas as described by Badame [9]

TO the existing grammar for Excel formulas, we firstly add a production rule

<Transformation> ::= <Formula> "↔" <Formula>

This rule allows for two formulas to be combined with the symbol ↔, indicating that those two formulas may be transformed into each other. Hence, the rule F ↔ F_1 is valid for each valid Excel formula F and F_1. It expresses that formula F can be transformed into F_1 and vice versa. An example of this is

A1+A2+A3 ↔ SUM(A1:A3).

Furthermore, our language permits the use of a selected set of variables to indicate a transformation is limited to a certain formula construct. Therefore, we change the definition of expressions

<ExpressionPrimitive> ::= (as above) | F | R | C | P

F represents a <Formula>, R represents a <RangeReference>, C represents a <CellReference> and P represents a <Primitive>. These transformations can be applied if the formula it is applied to has exactly that non-terminal on the place of the variable. For example

SUM(R) ↔ SUM(R) + 5

This rule, which is an example of a transformation that is not behavior preserving, indicates transforming a SUM over any range into a SUM over that range plus 5. An example of an application of this is transforming SUM(A1:A3) into SUM(A1:A3)+5.

Furthermore, the BumbleBee grammar permits parametrized references to cells, by adapting the production rules for cells

<Cell> ::= (as above) | "{" <VarCell>"," <VarCell> "}"

and adding

<char> ::= [a-z]
<int> := [1-9]
<VarCell> ::= <char> (+ <int>)?

This allows for variables in places where the original Excel grammar only contains cells. As described in the production rule, 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, by adding '...' as an expression:

<Expression> ::= (as above) | "..."

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).

4. APPLICATION EXAMPLE: REFACTORINGS

Now that we have defined the transformation grammar, we use it to describe a number of refactorings. These refactorings are generalizations of refactorings in our previous work, which showed that spreadsheet formula smells are very common [6] and that refactorings for them are widely applicable and that refactoring them with a tool is both quicker and less error-prone [7].

The grammar 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*, those refactorings that result in changes to multiple cells. An example of this is the extract column refactoring in [7], with which part of a formula is placed in a new cell. These are left for future work. Two in-cell refactorings are described in [7]: Replace Awkward Formula and Guard Call.

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 [7] 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. More specifically, we support the ten most commonly used function in the EUSES corpus [10]. This corpus contains real-life spreadsheets from 11 different domains, ranging from educational to financial, and from inventory to biology. It was created in 2005 and has since then been used by several researchers to evaluate spreadsheet algorithms, among which [11] and [12]. Barowy *et al.* [13] 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 can define refactorings with our transformation grammar. 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}

However, the other four functions (VLOOKUP, INDEX, MATCH and OFFSET) are more complex, as they are used not for calculation, but to reference cells or cell values by location or content. The following gives a short explanation of these four functions, for a more detailed insight we refer to the official Office documentation.[1].

MATCH The MATCH function searches for a specified

[1]http://office.microsoft.com/en-us/excel-help/

item in a range of cells, and then returns the relative position of that item in the range. For example, if the range A1:A3 contains the values 5, 25, and 38, then the formula =MATCH(25,A1:A3,0) returns the number 2, because 25 is the second item in the range.[2]

OFFSET The OFFSET function returns a reference to a range that is a specified number of rows and columns from a cell or range of cells. The reference that is returned can be a single cell or a range of cells. You can specify the number of rows and the number of columns to be returned.[3] For example, the formula OFFSET(A2,5,0,2,1) return a reference to the range A7:A8, as it starts at A7 (=A2 plus 5 row and 0 columns) and takes a range sized 2 rows and 1 column from there.

VLOOKUP The VLOOKUP function searches for a value in the first column of a range of cells, and then returns a value from any cell on the same row of the range. For example, suppose that you have a list of employees contained in the range A2:B10. The employees' ID numbers are stored in the first column of the range and the names are in the second. Now you can use the VLOOKUP function to return the name of that employee. To obtain the name of employee number 38, you can use the formula =VLOOKUP(38, A2:C10, 3, FALSE). This formula searches for the value 38 in the first column of the range A2:C10, and then returns the value that is contained in the second column of the range and on the same row as the lookup value. The final boolean argument indicates whether an exact match (FALSE) or an approximate match (TRUE) needs to be found.[4]

INDEX The INDEX function returns the reference of the cell at the intersection of a particular row and column. For example INDEX(A1:B7,5,2) returns a reference to B5, as that is the cell at the fifth row and second column of range A1:B7. If the first argument is made up of nonadjacent selections, you can use a fourth argument to select which of the regions to look into.[5]

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 [7] for the most popular 10 Excel functions using the BumbleBee grammar .

4.2 Guard Call

Badame and Dig [7] 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.* [6] 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. APPLICATION EXAMPLE: MIGRATION

In addition to using the BumbleBee grammar for refactorings, we are also able to express all migrations needed to migrate formulas written in previous Excel versions to Excel 2010. In the following, we describe how the BumbleBee grammar is capable of describing the needed transformations. The full list of all 45 transformations can be downloaded.[6] For this second application example, we consider all new and changed built-in functions in Excel 2010[7]. These changes can be divided into three categories:

1. Renamed functions
2. Parameters added
3. Parameters removed

5.1 Renamed functions

The transformation for renaming is relatively simple: for a function F with n arguments that is replaced by a function F′, we add a rule of the form

$$F(A_1,A_2,...,A_n) \leftrightarrow F'(A_1,A_2,...,A_n)$$

An example of this is the Excel 2010 function TTEST which replaces the old function T.TEST. The name is the only attribute of this function that is changing, both function take two ranges and two integers as arguments. This can be implemented with

$$T.TEST(R_1,R_2,V_1,V_2) \leftrightarrow TTEST(R_1,R_2,V_1,V_2)$$

[2]http://office.microsoft.com/en-us/excel-help/match-function-HP010062414.aspx
[3]http://office.microsoft.com/en-us/excel-help/offset-function-HA102752910.aspx
[4]http://office.microsoft.com/en-us/excel-help/vlookup-function-HP010343011.aspx
[5]http://office.microsoft.com/en-us/excel-help/index-reference-function-HA102927777.aspx
[6]http://www.felienne.com/?p=2964
[7]http://office.microsoft.com/en-001/excel-help/what-s-new-changes-made-to-excel-functions-HA010355760.aspx

5.2 Parameters added

In case of added parameters, it less straight forward, as we must introduce the default value for the new parameter. Therefore, the transformations for this type of migration will have the form

$$F(A_1,A_2,...,A_n) \leftrightarrow F'(A_1,A_2,...,A_n,A_{n+1})$$

The BETA.DIST function is an example of this. The original function takes 3 parameters, where the new BETADIST function takes four. The newly introduced fourth parameter indicates whether the cumulative version of the distribution needs to be used, which was the default option for BETA.DIST. Therefore, we can use this transformation to migrate BETA.DIST

$$\text{BETA.DIST}(V_1,V_2,V_3) \leftrightarrow \text{BETADIST}(V_1,V_2,V_3,\mathit{TRUE})$$

By introducing a new default parameter, we can transform all Excel 2010 functions for which a new parameters is introduced.

5.3 Parameters removed

In some cases, parameters are removed. In this case, the transformation again is straight forward

$$F(A_1,A_2,...,A_n) \leftrightarrow F'(A_1,A_2,...,A_{n-1})$$

This change applies to, for example, the CEILING function, in pre-Excel 2010 version of this function, it is required to add the significance as second parameter, CEILING(7.4,1), for example, results in 8, whereas CEILING(7.4,3) results in 9, as it is rounded to the next multiple of 3. In Excel 2010, CEILING is renamed to CEILING.PRECISE, and furthermore, the significance parameter is optional and 1 is used as default. This means we may omit it in case it is equal to 1, resulting in this transformation

$$\text{CEILING}(V_1,1) \leftrightarrow \text{CEILING.PRECISE}(V_1)$$

6. 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 [5, 14] as a basis for reading, parsing and analyzing the formulas. BumbleBee is implemented in C# and F# using Visual Studio 2010.

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.

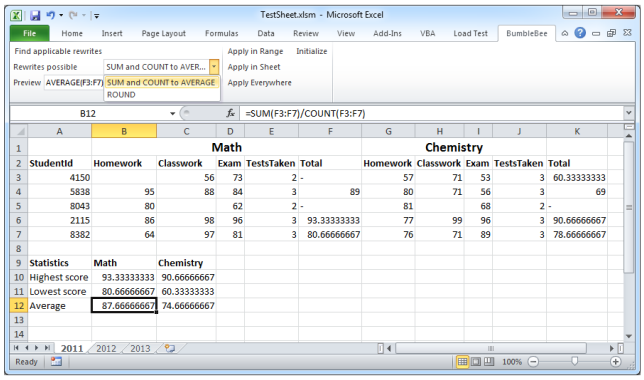

Figure 3: Screenshot of the current BumbleBee implementation in Excel 2010. The options for transformation of cell B12 are shown in the dropdown box, and the preview field shows the selected transformation applied to B12.

7. EVALUATION

To evaluate our approach, we will answer the following three research questions:

1. Is the BumbleBee grammar needed?
2. Is the BumbleBee grammar expressive?
3. Does it improve end-user productivity?

7.1 R1: Necessity

To demonstrate the necessity of our approach, we have again analyzed the EUSES corpus [10] and have determined how many spreadsheets it contains with non-connected regions of copy-equivalent formulas. These spreadsheets would benefit from a tool to update those formulas in a consistent way. Full details of our analysis are available on the GitHub page of this paper.[8]

In this analysis, we have only considered the spreadsheets in the 'processed' subfolder of the corpus, as advised by the creators of the corpus in their readme file. This adds up to a total of 4,489 files. For this analysis, we have converted these files to xlsx file format. There were 48 files that we were unable to convert. This leaves 4441 files for analysis, of which 1839 contained formulas.

For those 1839 files, we have firstly determined whether they contained non-connected regions with copy-equivalent formulas. We found that 1323 of those 1839 spreadsheets (72%) contained at least two non-connected regions with copy-equivalent formulas.

In Figure 5, the number of formulas is shown, which appear in non-connected regions, for the 1323 spreadsheets in which this occurs. As this picture shows, many of the spreadsheets (314) only have one such formula. However, there is a long tail, with ten spreadsheets even containing more than 100 formulas spread over non-connected regions.

In Figure 6 we show the size, measured in the number of regions, for the largest region, per spreadsheet. While the largest group of regions for most spreadsheets (947) consists of fewer than 10 regions, there are spreadsheet with as many

[8]https://github.com/Felienne/icse2014/tree/master/Euses_Run

	A	B	C	D	E
1	Equity1.XLS				
2	Risk & return with three assets				
3					
4	Asset Data		Exp Ret	Std Dev	
5		TBills	0.006	0.043	
6		Bonds	0.021	0.101	
7		Shares	0.09	0.208	
8					
9	Correlation Matrix		TBills	Bonds	Shares
10		TBills	1	0.63	0.09
11		Bonds	=D10	1	0.23
12		Shares	=E10	=E11	1
13					
14	VCV matrix		TBills	Bonds	Shares
15		TBills	=C10*VLOOKUP(C$14,$B$5:$D$7,3,FALSE)*VLOOKUP($B15,$B$5:$D$7,3,FALSE)	0.00273609	0.00080496
16		Bonds	=C11*VLOOKUP(C$14,$B$5:$D$7,3,FALSE)*VLOOKUP($B16,$B$5:$D$7,3,FALSE)	0.010201	0.00483184
17		Shares	=C12*VLOOKUP(C$14,$B$5:$D$7,3,FALSE)*VLOOKUP($B17,$B$5:$D$7,3,FALSE)	0.00483184	0.043264

Figure 4: Screenshot of spreadsheet equity-1.xls from the EUSES corpus, where cells C15:C17 are copy-equivalent.

as 1436 regions for the same formula. Editing them manually in case of a change seems quite tedious.

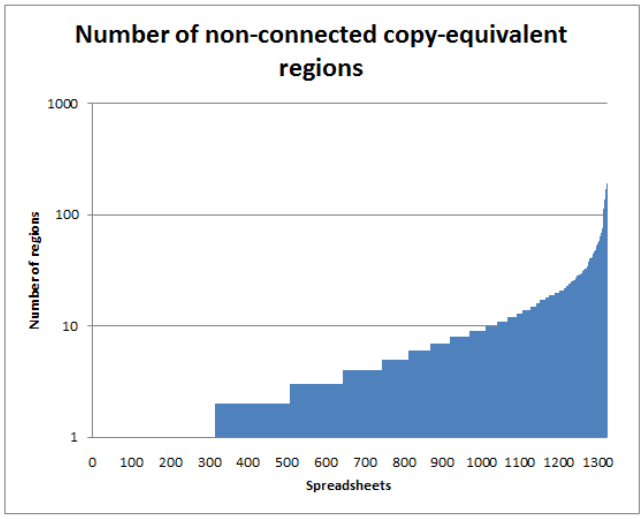

Figure 5: Number of spreadsheets with at least one formula that occurs in multiple non-connected regions, showing the number of regions for the formula with the most occurrences.

Table 7.1 shows the distribution of spreadsheets containing non-connected copy-equivalent regions per folder of the EUSES corpus. As can be seen from this table, they occur frequent (> 50%) in all domains, except for Filby, which are spreadsheets concerning biology. This is caused by the fact that these spreadsheets are of a different type, they are used to create calculation models, rather than using the tabular layout as is usual in finance. Still, even there we see that 30% of spreadsheets contain non-connected copy-equivalent regions.

In conclusion, this analysis underlines the necessity of our approach, as those 72% of the corpus would benefit from a tool with which these regions could be modified as one.

7.2 R2: Expressiveness

As described in Section 4, our new transformation language is able to express all intra-formula refactorings in previous work on refactoring. This shows that the BumbleBee grammar is as least as powerful as earlier refactoring work.

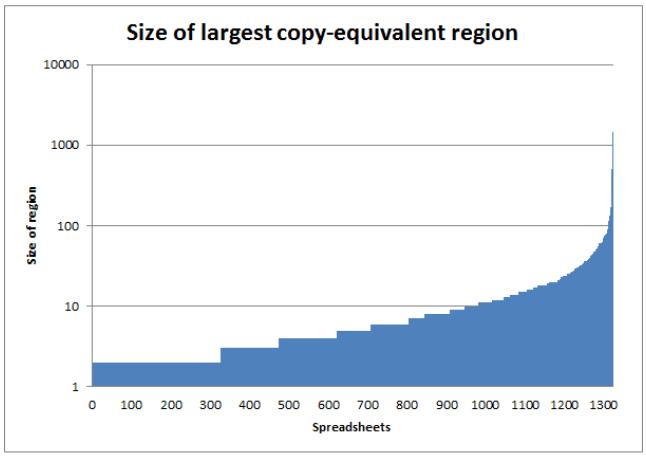

Figure 6: Number of formulas that occur in multiple non-connected regions.

Table 1: Number of spreadsheets with non-connected copy-equivalent regions per domain

Folder	Containings non-connected regions	Containing formulas	Percentage
cs101	4	8	50%
database	153	210	73%
filby	11	32	34%
financial	302	377	80%
forms3	16	22	73%
grades	257	362	71%
homework	217	326	67%
inventory	210	287	73%
jackson	12	12	100%
modeling	137	199	69%
personal	4	4	100%
Total	1323	1839	72%

In addition to that, in Section 5 we demonstrated another application, namely the migration formulas to Excel 2010. Finally, the BumbleBee grammar is also capable of expressing non-behavior preserving transformations, like changing business rules. A category of transformations we do not yet support are transformations which require the updating of multiple cells.

7.3 R3: End-user productivity

7.3.1 Setup

To test whether BumbleBee increases the productivity of spreadsheet users, we conduct a controlled experiment in which we compare two random groups of spreadsheet users. Performing the same assignments, a *test group* will use BumbleBee to do this, while the *control group* will not. Subject of this evaluation a spreadsheet to calculate student scores for two different courses. The spreadsheet is available on-

Table 2: The transformation rules loaded into BumbleBee in the experiment

Transformation	Name
F ↔ ROUND(F,0)	ROUND
SUM(R_1)/COUNT(R_1) ↔ AVERAGE(R_1)	SUM and COUNT to AVERAGE
3 ↔ 2	Change 3 to 2
SUM({i,j}:{i+2,j})/2 ↔ SUM({i,j}:{i+2,j})/{i+3,j}	Divide by previous column

line[9]. On this spreadsheet, of which a screenshot is shown in Figure 8, we ask both groups to perform the following tasks:

- The calculation of the averages in cell B12 and C12 is a bit cumbersome. Simplify them.
- Starting in 2012, the requirement to make all three tests is loosened, a student can also take two out of three. The score is then calculated by averaging the two parts that were taken.
- We would prefer to have scores calculated in integers only. This holds for students scores as well as course grades like average score. Make this change.

In addition to performing these tasks, we also ask participants to comment on each task after they have performed it. We ask them specifically whether they found the tasks hard to perform.

We have put this experiment on SurveyMonkey[10] and have recruited users both through spreading this link via Twitter and through putting this survey on Amazon's Mechanical Turk. In order to make a fair comparison, we assert that it is necessary for this study that participants have some knowledge of Excel. To guarantee this, we start with a pre-test, in which we both ask the participant to judge their Excel level and furthermore ask to evaluate three of Excel formulas by hand. For the full details we refer to the SurveyMonkey survey. Figure 7 shows one of the three questions in the pre-test. We have configured the experiment such that it can only be filled out from one IP address once, to prevent participants from attempting the pre-test several times. Only after these pre-test questions are correctly answered, subjects can enter the actual experiment. They are than randomly assigned to one of the two groups. For the test group, BumbleBee is loaded with the transformation rules shown in Table 7.3.1, in addition to the refactoring rules described in Section 4.

The spreadsheet that we provide users with, contains an embedded piece of VBA code, which logs actions that a user performs. The logging macro includes data entry, modifying formulas and hitting undo.

7.3.2 Results

In total, 53 subjects participated in the study: 26 in the control group and 27 in the test group. 21% of the participants were female and 79% were male. Most participants (60%) were aged between 25 and 34. 11% was between 35 and 44, 8% between 45 and 54 and 21% was over 55. Their occupations varied between 'Business and Financial'

[9]http://www.felienne.com/?p=2964

[10]https://www.surveymonkey.com/s/GDSW5Y8

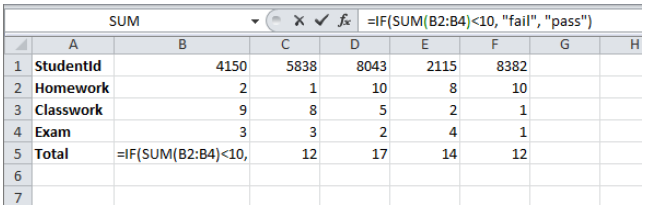

Figure 7: In the pre-test, subjects are asked to evaluate the results of Excel formulas, including as this one.

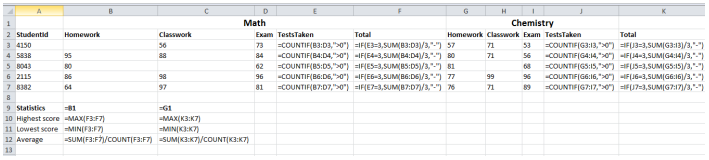

Figure 8: Test subjects are asked to perform tasks on this spreadsheet.

(21%) Computer and Mathematical(53%), Life and Social Sciences(9%) and Education(17%). Table 7.3.2 presents an overview of the characteristics of the participants. Figure 9 shows the times needed for the three tasks, the test group is shown on the left and the control group on the right. For tasks 1 and 2, the results are normally distributed. For task 1, the Shapiro-Wilk test results in W = 0.94 and 0.95 for the test and control group respectively and for task 2 in 0.94 and 0.96. The following describes the results of the study.

Task 1: Simplify SUM and COUNT.

The description for this task was: *The calculation of the averages in cell B12 and C12 is a bit cumbersome. Simplify them.* In the control group, different strategies were used to make this change. Most participants first made the change on the first worksheet and then copy-paste the formulas to the other worksheets. Five subjects made the change entirely by hand. Two participants of the control group made a mistake, by changing B12 and dragging the formula handle (the little 'plus' in the lower right corner rightwards). One participant in the test group made a mistake by selecting 'apply in sheet' rather than applying it in the entire spreadsheet.

Even with the use of copy-paste by the control group, the test group was quicker than the control group. The left-most box plot in Figure 9 shows the times for this task. We have excluded the three wrong answers (one from the test group and two from the control group) from the analysis. The test group (n=25) needed an average 73.2 ± 14.5 seconds for the first task, which is significantly less that the control group (n=25) who used 83.8 ± 18.3 (one-sided t-test, $p<0.05$*, effect size 0.64).

From the low time needed for this task, but also from the answers to the open question on the difficulty of this task, we learned that participants found it easy to perform. Some even doubted the question a bit, as one participant in the control group stated "I immediately understood it, unless there's an easier method than AVERAGE()." Wrong answers were due to performing operations in the wrong cells,

Table 3: Overview the demographic information of the experiment subjects

	Test group	Subjects	Control Group	Subjects	Total	Percentage
Gender	Male	23	Male	19	42	9%
	Female	3	Female	8	11	21%
Age	25-34	17	25-34	15	32	60%
	35-44	4	35-44	2	6	11%
	45-54	3	45-54	1	4	8%
	55-64	2	55-64	9	11	21%
Occupation	Business and Financial	3	Business and Financial	8	11	21%
	Computer and Mathematical	14	Computer and Mathematical	14	28	53%
	Life and Social Sciences	3	Life and Social Sciences	2	5	9%
	Education	6	Education	3	9	17%

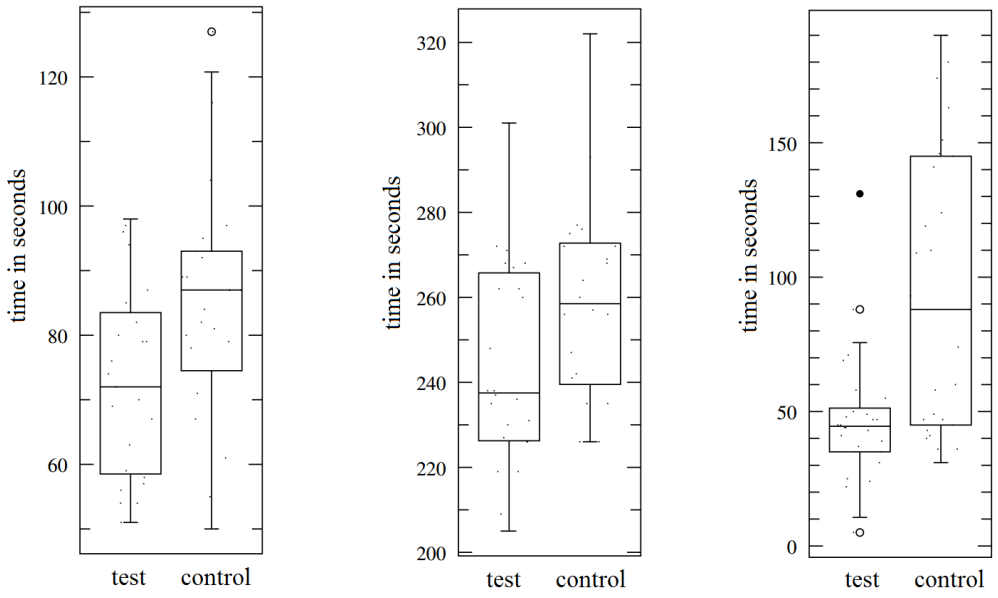

Figure 9: Results of the user experiment shown in a box plot with data swarm, for the different three tasks. In all three plots, the test group is shown on the left and the control group on the right.

and not due to not knowing how to improve the formula. This shows the fact that the test group did not have to come up with the formula to use—as they could simply select the transformation from the BumbleBee dropdownlist— did not give the test group participants an unfair advantage over the control group.

Task 2: Changing business rule.

The second task, in which we asked participants to change the calculation of the formulas in columns F and K on worksheets 2012 and 2013, with the following task *Starting in 2012, the requirement to make all three tests is loosened, a student can also take two out of three. The score is then calculated by averaging the two parts that were taken.* The results of this task indicate that it was more difficult for participants of both groups. More mistakes were made: in the test group two participants did not complete the task correctly, five did not successfully complete it in the control group. Times needed for this tasks corroborate this, as it took the test group (n=24) an average of 4.1 minutes ± 23.7 seconds and the control group (n=22) 4.3 minutes ± 23.2 seconds. This is a significant difference (one-sided t-test, $p<0.05$*, effect size 0.64).

As can be seen in the middle plot in Figure 9, there is quite some spread in time needed to complete the task correctly in both groups. In the test group, there are 9 subjects that need more than 250 seconds. 8 of those are subjects who use of BumbleBee is restricted to 'apply in range', which is a lot slower than using the 'apply in sheet' or 'apply everywhere' options. The other subject who took more than 250 seconds made an opposite mistake, he applied the transformation everywhere and then realized it only had to be done in 2012 and 2013, so he spent a considerable amount of time on manually reverting worksheet 2011. In the control group, there are differences also. Most participants only changed the top most formulas and subsequently used the fill handle by dragging it down or double clicking it. However, only some realized that they could copy-paste the formulas from column F to column K. As they are copy-equivalent, Excel will correctly update the references. Even fewer participants realized that they could copy-paste these formulas to a different worksheet also, and they would still be calculated correctly.

Table 4: Subjects who completed assignment incorrectly

Assignment	Control Group	Test Group
#1	2	1
#2	5	2
#3	0	0

Task 3: Rounding.

For the final task, we asked subjects to round all values to integers, with this assignment: *We would prefer to have scores have to be calculated in integers only. This holds for students scores as well as course grades like average score. Make this change.*

This task seems to be easier than the second task: all subjects completed this task correctly, both in the control group and in the test group. As can be seen in the rightmost boxplot in Figure 9, in this third task, the times between the control and test group differ a lot. The control group needed an average of 94.1 ± 52.4 seconds for this task, while the test group averaged 47.2 ± 23.8 seconds. This difference is significant (one-sided Mann-Whitney-Wilcoxon test, $p<0.001$***, effect size 1.11). We assert that the reason that the difference is the biggest for this task is related to the number of regions in which the transformation has to be performed; changes had to be made in 8 different regions per worksheet, making a total of 24 regions for the entire worksheet. So even supported by the fill handle and copy-paste, still at least 8 changes had to be made in the control group, whereas the test group could, in principle, solve this task with just one action. The results of this third task seem to indicate that the benefits of BumbleBee increase with number of regions which have to be edited.

7.3.3 General findings and directions for improvement

Difference in errors made Because of the relatively low number of errors made, we do not see a statistically significant difference between the test and the control group. However, the test group does perform better as shown in the Table 7.3.3. Especially in the second test, which we consider to be the most difficult one, the test group only contains two erroneous solutions, where the control group has five. If we look into why these errors are made, we see that of the five errors in the control group, four were due to inconsistent changes: column F was correctly updated, but column K was missed. One subject did not understand the assignment and simply did not complete it. In the test group too, there was one participant who misunderstood the question, he did loosen the rule of at least 3 tests taken, but did not change the calculation of the results.

The other participant who did not answer correctly used 'apply in sheet' instead of 'apply everywhere', this is the same mistake this subject made in the first task. These results seem to underline the hypothesis that BumbleBee addresses the problem of inconsistent changes, by helping spreadsheet users to apply a change everywhere, but, obviously, additional studies are needed.

A better undo

One of this things we saw in the tests is that BumbleBee could use a better, more specific undo functionality. While analyzing the tests, we noticed that some participants in the test group used the 'apply everywhere' functionality, but later realized they had applied it to too many cells, for instance on all worksheets, instead of on two if the three. In the current implementation, such a rewrite operation can only be reverted as a whole. Therefore, these users undid the whole thing, and then applied the transformation to the two worksheets where it was needed. Being offered the possibility to revert it only in the place where they noticed an 'undo' was needed, might improve user experience.

7.4 Research questions revisited

In this section, we revisit the research questions.

Is the BumbleBee grammar needed? As shown by our analysis of the EUSES corpus, 72% of all spreadsheets contain non-connected copy-equivalent regions, and could hence benefit from a tool to transform them consistently.

Is the BumbleBee grammar expressive? BumbleBee is at least as expressive as existing work on spreadsheet refactoring. Moreover, it can express transformation for all formula migrations to Excel 2010.

Does it improve end-user productivity? Our user study shows a significant different in the test group on all three assignments.

8. RELATED WORK

Efforts related to our research include work on source code refactoring, most prominently the work of Fowler [15].

Furthermore, there is work on the improvement of spreadsheet, such as the work on spreadsheet design guidelines. Raffensberger [16], 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.* [17] also propose guidelines to improve spreadsheet quality, which they base on principles of software engineering. Secondly, there are papers that address spreadsheets errors, like [4,18], together with their causes. Powell *et al.* for instance [19] names conditional formulas among the top three of commonly occurring spreadsheet error categories and this is one of the refactorings BumbleBee supports.

There is also related work on finding anomalies on spreadsheets, for instance the work on the UCheck tool [20–22]. 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 [23] 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 [24] and subsequently validated [25] a method to support end-users in defining and analyzing tests for spreadsheets. In follow-up work [26], copy-equivalent regions were exploited to optimize the testing process.

This paper differs from our previous work by focusing on the improvement, rather than on the analysis of spreadsheets as we did in [14]. In that paper we focused on detecting smells between worksheets, like high coupling. That paper followed our earlier work, in which we worked on the visualization of spreadsheets by means of class diagrams [27]

and dataflow diagrams [14]. It differs from our earlier work on spreadsheet refactoring [7] because it is a more general method for formula transformation, refactorings can be described with a grammar and are not embedded in the tool. Furthermore, the current approach is not necessarily aimed at refactoring, but also at migration or changing business rules. Furthermore.

9. DISCUSSION

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

9.1 Threats to Validity

9.1.1 Construct Validity

A threat to the construct validity of this study is *hypothesis guessing*. We announced in the tweet in which we invited participants [28] that we were testing a new tool, and therefore participants in the test group might try their best to make this new tool work. We, however, did not make them aware that there was a control group also, to mitigate this effect.

9.1.2 Internal Validity

The internal validity of our study could be affected by *experiment mortality*, as some of the participants did not complete the tasks correctly and were not included in the study. Therefore we consider them dropouts with respect to the statistical analysis. As there are more dropouts in the control group, this influences the experiment slightly. Regarding other threats to internal validity, by randomizing assignment to the test and control group, we have aimed to mitigate them as much as possible.

9.1.3 External Validity

A threat to the external validity of our evaluation concerns the representativeness of the selected spreadsheet and the tasks in the experiment. The spreadsheet is relatively simple, having only a few worksheets and a handful of formulas and as such also the maintenance tasks are relatively simple. However, already on such a small and simple spreadsheet, our approach is beneficial to users. We suspect, although this obviously requires further studies, that the decrease in editing time will likely be larger in more complex spreadsheets, rather than smaller.

Another threat to external validity is the *novelty effect*: users in the test group might use BumbleBee, as it is new and therefore they want to try how it works. They might not use the tool anymore once they have grown accustomed to it, they might use it less frequent, diminishing the positive effect.

9.2 Transformations impacting multiple cells

The BumbleBee grammar as it is currently defined, only supports changes within one formula. While this paper shows this is already 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.

9.3 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.

10. CONCLUDING REMARKS

The goal of this paper is to facilitate the ease of updating groups of similar, copy-equivalent, formulas in spreadsheets. To that end we have defined a transformation language and a corresponding tool: BumbleBee. We have subsequently evaluated our approach in three ways: We have shown that it is needed by analyzing a large corpus of spreadsheets, we have shown that it is expressive with two extensive application examples and finally we have performed a quantitative evaluation, by performing an experiment with 53 subjects on an example spreadsheet.

The key contributions of this paper are as follows:

- A grammar to describe spreadsheet formula transformations (Section 3)
- Two extensive application examples (Section 4 and 5)
- An implementation of the grammar into the BumbleBee tool (Section 6)
- An evaluation of the necessity, expressiveness and impact on the efficiency of spreadsheet users BumbleBee. (Section 7)

Our evaluation shows that 72% of spreadsheets contain non-connected regions with copy-equivalent formulas and could thus benefit from the BumbleBee. Furthermore, we have shown that the BumbleBee grammar is at least as expressive for intra-formula refactorings as previous work on spreadsheet formula refactorings. Finally, an empirical evaluation revealed that using BumbleBee results in a significant speedup of editing spreadsheet formulas.

The current research gives rise to several directions for future work. Firstly, more empirical evidence of 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.

11. REFERENCES

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

[2] L. Bradley and K. McDaid, "Using bayesian statistical methods to determine the level of error in large spreadsheets," in *Proc. of ICSE '09, Companion Volume*, pp. 351–354.

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

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

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

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

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

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

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

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

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

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

[13] D. W. Barowy, D. Gochev, and E. D. Berger, "Data debugging," Tech. Rep. UM-CS-2012-033, University of Massachusetts, Amherst. `https://web.cs.umass.edu/publication/details.php?id=2283`.

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

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

[16] J. Raffensperger, "New guidelines for spreadsheets," *International Journal of Business and Economics* **2** (2009) 141–154.

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

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

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

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

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

[22] M. Erwig, "Software engineering for spreadsheets,"*IEEE Software* **26** (September, 2009) 25–30.

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

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

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

[26] 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.* **28** (June, 2002) 576–594. `http://dx.doi.org/10.1109/TSE.2002.1010060`.

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

[28] Felienne, [Twitter user `@Felienne`], "Have half an hour to spare and love spreadsheets? Please participate in a test for a new tool we are building http://ow.ly/o7Lrh" 21 Augustus, 2013.