Detecting and Predicting Evolution in Spreadsheets
A Case Study in an Energy Network Company

Jansen, Bas; Hermans, Felienne; Tazelaar, Edwin

DOI
10.1109/ICSME.2018.00074

Publication date
2018

Document Version
Accepted author manuscript

Published in
Software Maintenance and Evolution (ICSME), 2018 IEEE International Conference on

Citation (APA)
Jansen, B., Hermans, F., & Tazelaar, E. (2018). Detecting and Predicting Evolution in Spreadsheets: A Case Study in an Energy Network Company. In *Software Maintenance and Evolution (ICSME), 2018 IEEE International Conference on* https://doi.org/10.1109/ICSME.2018.00074

Detecting and Predicting Evolution in Spreadsheets

A Case Study in an Energy Network Company

Bas Jansen
Delft University of Technology
Email: b.jansen@tudelft.nl

Felienne Hermans
Delft University of Technology
Email: f.f.j.hermans@tudelft.nl

Edwin Tazelaar
Alliander
Email: edwin.tazelaar@hymatters.com

***Abstract*—The use of spreadsheets in industry is widespread and the information that they provide is often used for decisions. Research has shown that spreadsheets are error-prone, leading to the risk that decisions are made on incorrect information.**

Software Evolution is a well-researched topic and the results have proven to support developers in creating better software. Could this also be applied to spreadsheets? Unfortunately, the research on spreadsheet evolution is still limited. Therefore, the aim of this paper is to obtain a better understanding of how spreadsheets evolve over time and if the results of such a study provide similar benefits for spreadsheets as it does for source code.

In this study, we cooperated with Alliander, a large energy network company in the Netherlands. We conducted two case studies on two different set of spreadsheets that both were already maintained for a period of three years. To have a better understanding of the spreadsheets itself and the context in which they evolved, we also interviewed the creators of the spreadsheets.

We focus on the changes that are made over time in the formulas. Changes in these formulas change the behavior of the spreadsheet and could possibly introduce errors. To effectively analyze these changes we developed an algorithm that is able to detect and visualize these changes.

Results indicate that studying the evolution of a spreadsheet helps to identify areas in the spreadsheet that are error-prone, likely to change or that could benefit from refactoring. Furthermore, by analyzing the frequency in which formulas are changed from version to version, it is possible to predict which formulas need to be changed when a new version of the spreadsheet is created.

I. Introduction

The use of spreadsheets is widespread in industry. Panko [1] estimates that 95% of U.S. firms use spreadsheets in some form of financial reporting and Winston [2] estimates that 90% of all analysts in industry use spreadsheets for their calculations. Hermans *et al.* make compelling arguments that spreadsheets are code [3] and with an estimate of 500 million active users worldwide, spreadsheets are a successful end-user programming language. Notwithstanding their common use, research has also proven that spreadsheets are error-prone [4] which can lead to incorrect decisions and loss of money[1].

Until now, much of the spreadsheet research was focused on improving spreadsheets by applying software engineering methods to them. The concept of testing was brought to spreadsheets by Rothermel *et al.* [5] and recently Roy [6] investigated how users test spreadsheets. Hermans [7] and Cunha *et al.* [8] covered the topic of reverse engineering and proposed methods for extracting class diagrams from spreadsheets. Several studies were published about the existence of code smells in spreadsheets [9] [10] [11]. From code smells it is a small step to refactoring. Both Hermans and Dig [12] and Badame and Dig [13] proposed tools that support several refactorings for spreadsheet formulas.

Contrary to the general belief, most spreadsheets are not one-time models that only exists for a short time. The average lifetime of a spreadsheet is 5 years and during its lifetime they are used on average by 12 persons [14]. Because of their relatively long lifetime, spreadsheets, like software, evolve over time.

Software evolution is a well-researched topic within the domain of software engineering [15], [16], [17]. Research showed that the understanding of source code evolution helps to identify errors within the software and highlights areas that are change-prone and could possibly be improved. Could a better understanding of spreadsheet evolution, bring similar benefits to the domain of spreadsheets? Unfortunately, research on spreadsheet evolution is rather limited and a better understanding of how spreadsheets evolve is needed. We need answers on questions such as:

- How do spreadsheets evolve over time?
- What is the change frequency?
- What kind of changes are made?
- What are the motivations behind these changes?
- Will the study of spreadsheet evolution contribute to better change comprehension and prediction of spreadsheets?

Analyzing the evolution of spreadsheets leads to several challenges. First, we need to be able to correctly identify the changes that were made between the different versions of the spreadsheet. For this study, we are mainly interested in the changes that were made to the formulas. Changes in these formulas change the behavior of the spreadsheet and could possibly introduce errors into the spreadsheet. In this study, we developed FormulaMatch, an algorithm that identifies and visualizes these changes.

The second challenge related to the research of spreadsheet evolution is finding a set of different versions of the

[1] http://www.eusprig.org/horror-stories.htm

2576-3148/18/$31.00

same spreadsheet that were developed and maintained over a significant period of time. Dou *et al.* proposed a semi-automated approach to identify evolution groups within a larger spreadsheet group [18]. They applied this approach to the Enron spreadsheet corpus [**?**] which resulted in a versioned spreadsheet corpus called VEnron that consists of 7,294 spreadsheets spread over 360 evolution groups. The drawback of this dataset is that we have no access to the creators of the spreadsheet.

In our study, we want to understand the changes and the motivation behind these changes that occur during the evolution of a spreadsheet. Access to the creators is therefore crucial. For this reason, we decided not to use the VEnron corpus, but to cooperate with Alliander. Alliander is a Dutch energy network company. They are responsible for the distribution of electricity and natural gas in a significant part of the Netherlands. They have 5.7 million customer connections, maintain a 90,000 km electricity network and a 42,000 km gas grid and have a yearly revenue of 1.7 billion. Alliander provided us with two sets of spreadsheets that we could use for our analysis.

The remainder of this paper is organized in the following way. In the next section, we provide background information on spreadsheet evolution and discuss related work. In Section III we discuss the algorithm that we used to detect changes between different versions of a spreadsheet. The two case studies and our findings are presented in Section IV. Some topics that affect the applicability and suitability of our findings are discussed in Section V and we end the paper with concluding remarks in Section VI.

II. Background & Related Work

A. Software Evolution

The study of software evolution has provided us with insights that support developers in creating better software. The first studies date back to the late 1960s [19] although the term software evolution itself was not used until 1974 [20]. Lehman and Ramil published a comprehensive summary of 30 years of research on software evolution in 2003 [15].

Research from Novais *et. al.* provides reasons why the study of software evolution matters [16]. They conducted a systematic mapping study of the goals and purposes of software evolution visualization. The 5 most frequently mentioned purposes were: change comprehension, change prediction, contribution analysis, reverse engineering, identification of anomalies, and development communication.

In one of the early studies on software evolution, Gall *et. al.* studied the evolution of the software of a telecommunication switching system over a period of 21 months [17]. The system consisted of about 1500 to 2300 programs. They based their analysis on system properties like size, changing rate, and growing rate. They defined changing rate as the percentage of programs of the system which changed from one release to the next. In a similar manner, growing rate was defined as the percentage of programs of the system which have been added (or deleted) from one release to the next. The method Gall *et. al.* used to study the evolution of the telecommunication switching system could also be used to study the evolution of spreadsheets. Gall *et. al.* analyzed the changes in the system on the level of the individual programs. For spreadsheets, this could be adapted to the level of the individual formulas.

B. Spreadsheet Evolution

Research on spreadsheet evolution is still limited. In previous work, we compared 54 pairs of spreadsheets [21]. These pairs consisted of the original spreadsheet created by a customer and a version that was rebuilt by professional modelers from the company F1F9. The study provided insight into the effect the rebuilding had on the occurrence of code smells in formulas. However, each spreadsheet was analyzed for only two versions and therefore the obtained insights about the evolution of the spreadsheet were limited.

Most related to our research is the work of Dou *et. al.* [18]. They propose a semi-automated approach to identify evolved spreadsheets and recover the embedded version information. They applied this approach to the Enron corpus [**?**] [22] and created VEnron, a spreadsheet corpus with version information that consists of 360 evolution groups with a total of 7,294 spreadsheets. An evolution group is a set of spreadsheets that are all originated, either directly or via intermediate versions, from the same spreadsheet. Of these 360 evolution groups, 251 groups, consisting of a total of 4,149 spreadsheets, contained spreadsheets in which formulas were used. The study was mainly focused on identifying evolution groups within the Enron corpus, but the authors also compared the different spreadsheets within an evolution group with Microsoft's Spreadsheet Compare tool and made a technical classification of the type of changes they encountered. They studied 4 types of changes:

- **Structural**: Changes to the structure of the spreadsheet like adding or deleting rows.
- **Entered values**: Changed, added or deleted values (instead of formulas).
- **Formulas**: Changed, added or deleted formulas.
- **Calculated values**: Not a change in the formula, but in the calculated result of the formula, caused by changes in the input values for the formula.

In this paper, we focus on changes in formulas because a change in a formula means a change in the functionality of the spreadsheet. Furthermore, we want to investigate the motivations behind the change. Is it, for example, because of a new feature request, to correct an error or to optimize the performance of the spreadsheet. Dou *et. al.* were not able to answer these questions because they did not have access to the creators of the spreadsheet.

Xu *et. al.* proposed SpreadCluster, a different approach for recovering spreadsheet version information [23]. Instead of clustering spreadsheets based on their filenames, they use features of the spreadsheet, like table headers and worksheet names. Their study shows that SpreadCluster can cluster spreadsheets with a higher precision (78.5% vs. 59.8%) and recall rate (70.7% vs 48.7%) than the filename approach

that was used in VEnron [18]. Applying SpreadCluster to the Enron corpus resulted in a new versioned spreadsheet corpus: VEnron2. This study only focused on the clustering of spreadsheets and did not analyze the changes within an identified evolution group of spreadsheets.

C. Comparing Spreadsheets

Chambers *et. al.* present SheetDiff, a method for identifying the changes between two spreadsheets [24]. They determine all individual cell changes between two versions of the spreadsheet. Next, they optimize the cell changes into higher level changes like adding or deleting a column. As a result, this reduces the number of changes that are presented to the user. In their approach, they detect changes in all cells and not only the formulas.

Inspired by SheetDiff, Harutyunyan *et. al.* developed RowColAlign, an algorithm that can identify differences between two spreadsheets that is based on an algorithm for solving the one dimensional longest common subsequence problem [25]. RowColAlign is very successful in identifying changes caused by row insertion, row deletion, and cell level edits. However, it does not take into account operations such as copy-paste or cell-fill. Furthermore, the algorithm has not been applied to spreadsheets with cells containing formulas.

D. Spreadsheet Evolution Challenges

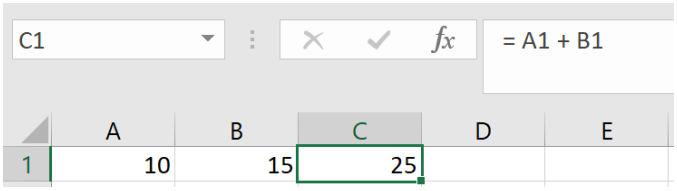
Studying spreadsheet evolution comes with its own challenges. It starts with the lack of version control systems (VCS). Spreadsheet users do not use Github or SVN because VCS are in general not suited to store the version history of spreadsheets. The lack of version control systems, also means that there are no commits and no commit messages. As a result, it is challenging to find a group of spreadsheets that belong to the same overall project. If such a group is found the next challenge is to determine the exact order in which they were created.

The next step in analyzing the evolution of spreadsheets is comparing the different versions of a spreadsheet in an evolution group. Finding changes between two versions of a program is relatively straightforward. Most programming languages are text-based and programs, classes and methods are organized in text files. Modern version control systems provide 'Diff' tools to highlight the differences between two versions on the level of code lines or even words.

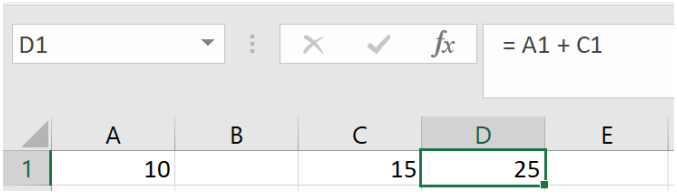
A normal 'Diff' tool is not able to compare two spreadsheets. Spreadsheets are not text-based files. A workbook consists of worksheets and worksheets consists of cells. The information in a spreadsheet is entered in a cell. To track changes in a spreadsheet you have to track changes on cell level. But what if a user inserts a row or a column or both. How should a compare tool determine which cells should be compared to each other?

Another important difference between source code and spreadsheets is the combination of data and code. In a spreadsheet data and code exist next to each other. If a user changes a numeric value in a single cell then all formulas that make reference to this cell will calculate a different outcome. If you compare the spreadsheet these changes will be detected. However, no change was made to any formula and one could argue that the spreadsheet is still unchanged.

An additional problem is the way the structure of a spreadsheet influences the formulas. The example of Figure 1 elucidates this problem.



(a) Original formula



(b) Changed formula after inserting column B

Fig. 1. The effect of changes in structure on formulas

Figure 1a shows a formula in cell C1 that adds up the values in cell A1 and B1, resulting in a calculated value of 25. Assume a user inserts a new column after column A. The result is shown in Figure 1b. Because of a structural change in the worksheet (adding a new column) the formula has changed. However, the function of this formula is still unchanged, as is its calculated value.

Now imagine that this spreadsheet contained 1,000 rows and that the formula in cell C1 of Figure 1a was copied down to all 1,000 rows. In that case, a single insertion of a new column would result in 1,000 formula changes. If these were all taken into account while comparing the two versions of the spreadsheet, the number of reported changes would be overwhelming and analysis of the differences between the two versions would be challenging.

III. Detecting Changes

As stated in the Section II, analyzing changes between two spreadsheets is difficult. Simple structural changes to a worksheet, like inserting a row or a column, can lead to a myriad of changes (see Figure 1). The goal of our study is to understand how spreadsheets evolve. We are especially interested in how formulas change over time. The formulas in a spreadsheet can be compared with the source code of a program. It determines its functionality, and it is in the formulas where most errors emerge. Furthermore, one would not argue that a program has been changed when it is run for a second time with different data. The same holds true for spreadsheets. If just the data has been changed the spreadsheet should be considered as unchanged.

The number of detected changes reduces if only changes in the individual formulas are taken into account and all changes

in structure, data and formatting are ignored. However, it still can result in thousands of changes. More complex spreadsheets contain a lot of formulas. For example, one of the models that we used in our case studies contained about 100,000 formulas. In such a spreadsheet, as described in the previous Section, a single change like moving a cell, can lead to a change in thousand related formulas. Presenting all these changes to the user, will not help them to understand the risks induced by the applied changes.

A. Unique Formulas

For this reason, the evaluation is reduced to only the unique formulas. We make use of the R1C1 notation of a formula to detect the unique formulas in a spreadsheet [26]. In Figure 2 we illustrate this with an example. In Figure 2a we see a small spreadsheet with 8 formulas. None of these formulas are the same. However, if we switch in Figure 2b from A1 notation (meaning the cell on the first row and the first column is referred to as A1) to the R1C1 notation (meaning that same cell is now referred to as R1C1) most of the formulas are the same. It turns out, as highlighted in Figure 2c, that there are actually only two unique formulas.

	Population x 1,000	Land area x 1,000 km^2	Pop. Density people/km^2
Germany	80594	357.021	=B2 /C2
France	67106	643.548	=B3 /C3
United Kingdom	65932	244.82	=B4 /C4
Italy	62138	301.32	=B5 /C5
Spain	48958	504.782	=B6 /C6
Total	=SUM(B2:B6)	=SUM(C2:C6)	=B7 /C7

(a) Formulas in A1 notation

	Population x 1,000	Land area x 1,000 km^2	Pop. Density people/km^2
Germany	80594	357.021	=RC[-2] /RC[-1]
France	67106	643.548	=RC[-2] /RC[-1]
United Kingdom	65932	244.82	=RC[-2] /RC[-1]
Italy	62138	301.32	=RC[-2] /RC[-1]
Spain	48958	504.782	=RC[-2] /RC[-1]
Total	=SUM(R[-5]C:R[-1]C)	=SUM(R[-5]C:R[-1]C)	=RC[-2] /RC[-1]

(b) Formulas in R1C1 notation

	Population x 1,000	Land area x 1,000 km2	Pop. Density people/km2
Germany	80,594	357	F1
France	67,106	644	^
United Kingdom	65,932	245	^
Italy	62,138	301	^
Spain	48,958	505	^
Total	F2	<	^

(c) Visualization of unique formulas

Fig. 2. Detecting Unique Formulas

B. Similarity score

When comparing two spreadsheets we start with detecting the unique formulas. For every formula in the first version V0, we have to find the equivalent in the second version V1. It is not possible to use the cell address, because it could have been changed if rows or columns were added or deleted. Furthermore, we have to take into account that the formula itself could have been changed.

To find a matching formula we analyze the similarity of the formulas in their R1C1 notation. As a measure of similarity we use the Levenshtein Distance [27], [28]. It denotes the minimum number of operations needed to transform one string into the other. A larger distance means that the strings are less similar.

With this similarity measure, we can match the formulas in V0 with the formulas in V1 that have the highest similarity. Still, this does not guarantee a perfect match, as different formulas can have the same similarity score. To overcome this the formulas are also compared to the following additional properties:

- Calculated result: the outcome of the formula.
- Parse tree depth: This is a measure of how nested a formula is.
- Path depth: a formula in a spreadsheet can make reference to another formula which again can make a reference to another formula. The path depth is the length of the complete calculation chain.
- Direct dependents: the number of cells that make a reference to the outcome of the formula.
- Direct precedents: the number of cells that are used by the formula as input.

C. Matching Algorithm

Trying to match the formulas in V0 with the formulas in V1, based on the maximum similarity score while taking into account the additional properties gives good results when the number of formulas in both versions are the same and no formulas have been added or deleted. To correctly handle the addition or deletion of formulas a more sophisticated matching algorithm is needed.

In their paper "College Admissions and the Stability of Marriage" [29], Gale and Shapley describe an easy and elegant algorithm that solves the problems caused by the adding and deleting of formulas. They describe an algorithm that solves the so-called College Admission problem. In this problem, colleges are considering a set of n applicants. They can only accept a quota of q applicants. Of course, applicants want to be accepted at the college of their preference. Gale and Shapley designed an algorithm that will lead to an optimal stable assignment, meaning that: *"every applicant is at least as well off under it as under any other stable assignment"* [29].

To accomplish this, every college should make an ordered list of their preferred applicants and every applicant should make an ordered list of their preferred colleges. Then the algorithm can be applied as follows:

1) All applicants apply to the college of their first choice.
2) A college with a quota of q places q highest ranking applicants on its waiting list and rejects the rest.
3) Rejected applicants then apply for their second choice.

4) Each college selects the top q from the new applicants and those on its waiting list, put these on its new waiting list and rejects the rest.
5) This is repeated until all applicants are either on a waiting list or have been rejected by all of their preferred colleges.

We applied this idea to develop FormulaMatch, an algorithm that we can use for our matching problem. For each formula in V0 (*'the colleges'*) we created an ordered list of preferred formulas in V1 (*'the applicants'*) using the similarity score. Also for all formulas in V1, we created an ordered list for their preferred formulas in V0. Every formula in V0 can only be matched to exactly one formula in V1, so we set the quota of the V0 formulas to 1. Then we applied the steps of the algorithm.

The result of FormulaMatch provides us for each formula in V0 with a matching formula in V1. Based on the similarity scores we know if the formula was changed or not. It is possible that no matching formula was found, meaning that this formula was deleted in V1. It is also possible that for a formula in V1 no matching formula in V0 was found. In terms of the algorithm, it was rejected. Rejected formulas are not existing in V0 and therefore, were added in V1.

Figure 3 shows a visualization of FormulaMatch[2]. All unique formulas are indicated with an identifier (FXX). If they were changed, they are highlighted in orange and in a comment the old and new version of the formula in R1C1 notation is displayed. Deleted formulas are highlighted in red and newly created formulas in green.

IV. Spreadsheet Evolution in two Industrial Case Studies

A. Setup

The goal of this paper is to obtain a better understanding of spreadsheet evolution. With the results of the case studies we will answer the following research questions:

RQ1 How do spreadsheets evolve over time?
RQ2 How common are changes in formulas during the lifespan of a spreadsheet?
RQ3 What are the reasons behind the changes?
RQ4 To what extent can the results of a spreadsheet evolution study, support end-users in creating spreadsheets that are easier to maintain and contain fewer errors?

For the case studies, we cooperated with Alliander. Alliander is one of the large energy network companies in the Netherlands and we had the opportunity to work with employees from the Analytics group of the Asset Management department. They provide data-based insights from maintenance and failure reports to support the development and maintenance of the energy network. We asked them to provide us with examples of spreadsheets that they created and maintained for several years. Based on this request we received two spreadsheet evolution groups that together contained a total of 73 spreadsheets. Before we started the case studies we provided the owners of the spreadsheets with information about the setup of the study and they gave us a short explanation about the purpose and context of their spreadsheets.

In the design of our study, we followed a mixed methods approach [30]. We started with a quantitative phase that consisted of a detailed analysis of the evolution of the received spreadsheets and followed up with a qualitative phase that consisted of interviews with the creators of the spreadsheets.

During the two case studies, the procedure was as follows: First, we asked the creators of the spreadsheet to put the spreadsheets in the evolution group in the correct update order. Subsequently, we analyzed the spreadsheets with our Spreadsheet Scantool and ran for each pair of consecutive versions of the spreadsheets our FormulaMatch algorithm. Based on the outcome of this analysis we summarized the evolution of the spreadsheet with the below-mentioned metrics. These metrics are the same as used by Gall *et. al.*[17], but we have adjusted them in the following way to make them suitable for the use with spreadsheets.

- Size: As stated in Section II, data and code are combined in a spreadsheet. The code in a spreadsheet is formed by the formulas and the number of unique formulas (see Section III-A) is used in this study, as metric for size. Still, formulas are not the only component of a spreadsheet that is responsible for its growth. A spreadsheet can also grow in size by just adding data to it. Therefore, we also use the number of non-empty cells as a metric for size.
- Changing rate: is defined as the percentage of unique formulas in the spreadsheet that changed from one version to the next.
- Growing rate: is defined as the percentage of unique formulas that have been added (or deleted) from one version to the next.

Subsequently, we used the results from the FormulaMatch algorithm to manually inspect all changes that occurred in the unique formulas. After that, we interviewed the creators of the spreadsheets and asked them the following questions:

1) How would you describe the development process of the spreadsheet?
2) Which parts in the spreadsheet do you consider complex?
3) Which important changes that you made in your spreadsheets do you remember?
4) What were the main reasons behind the changes you made in the spreadsheets?

The answer to these questions, combined with the data collected from the analysis of the spreadsheets, provides more background and context of the evolution process and will enable us to answer the research questions.

After these questions we discussed the findings of our analysis with the creators, to get a better understanding of the motivations behind some of the changes. We also asked them if and how the results of the study could help them to create and maintain better spreadsheets.

[2]To protect the sensitive data in the spreadsheet, all column and row headers in the example have been replaced with the term 'label'

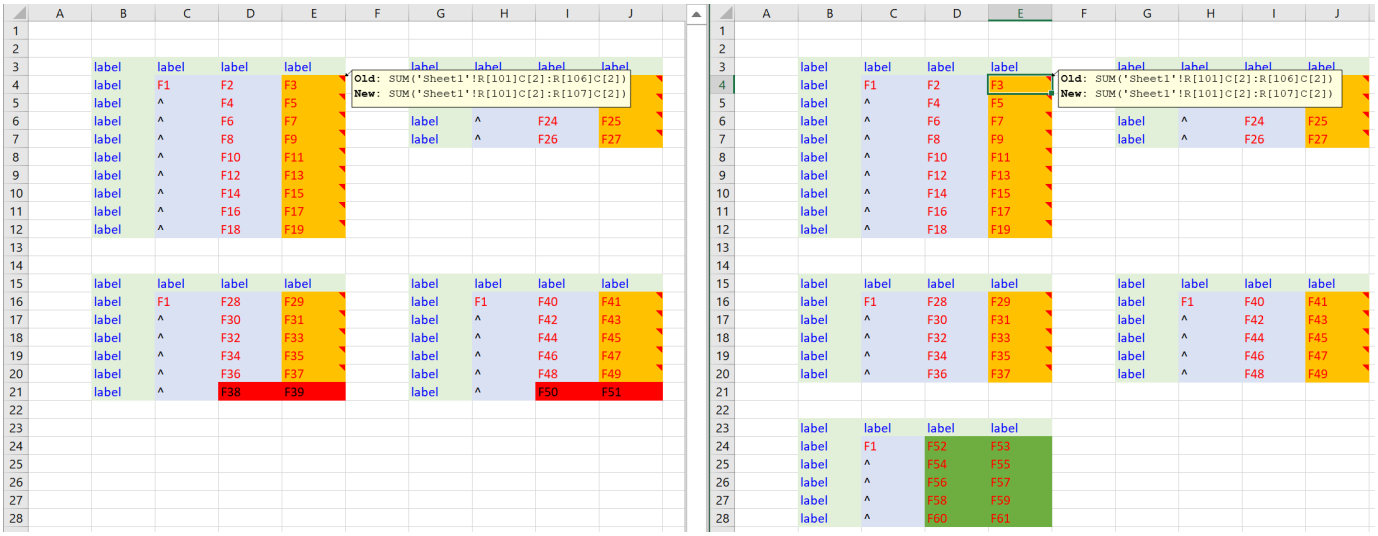


Fig. 3. Visualization of FormulaMatch

B. Case Study I: failure density in the natural gas distribution network

The first case study concerns a failure density model for the gas distribution network. It reports monthly on failure incidents, causes of failures and it provides a forecast for the coming months of the year. We received 35 spreadsheets that span a period of almost 3 years. The first spreadsheet dates back to February 2014 and the last version is from January 2017.

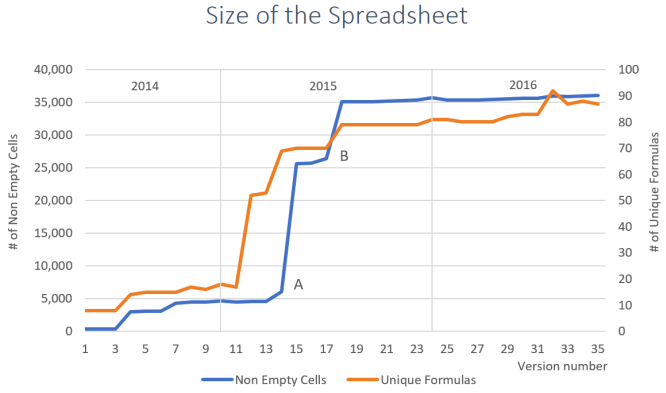


Fig. 4. Evolution of size of the spreadsheet over several versions

Figure 4 shows the evolution of size in number of cells over the different versions of the spreadsheet. As stated in Section II, spreadsheets consist of data and code. When analyzing the size of the spreadsheet, a distinction has to be made between data and formulas. Therefore, in the chart we show the number of non-empty cells as a metric for the size of the data and the number of unique formulas as metric for the formulas. The spreadsheet starts in version V01 with 383 non-empty cells and grows in V35 to 36,079 cells. The unique formulas grow from 8 in V01 to 87 in V35. These 87 unique formulas represent a total of 2,046 formulas.

Although the growth of the data and the unique formulas follow a comparable pattern, there is no relation between the two. During the interview, we discussed the growth patterns with the creator of the spreadsheet. The two steep increases (A and B in Figure 4) of non-empty cells were related to adding and extending a reference table with zip code information of the Netherlands. The increases in unique formulas that occurred at almost the same moments were related to new information requests and were not related to the zip code table.

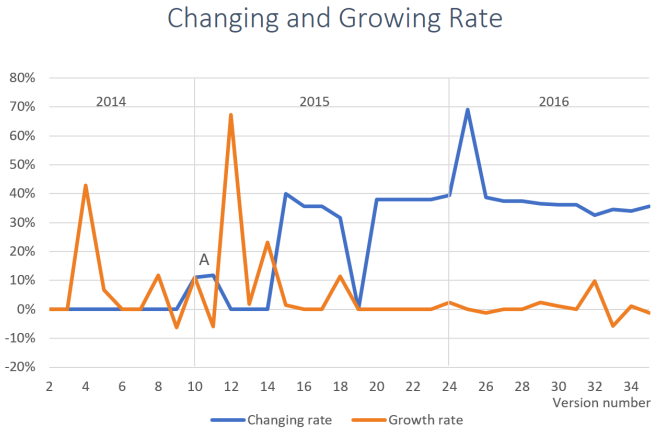


Fig. 5. Changing and growing rate of the spreadsheet over several versions

Figure 5 displays both the changing and growing rate of the spreadsheet. In the early versions, the model was extended with new functionality. We see this between V03 and V04 and around V11 to V14. This corresponds to the growth of the unique formulas shown in Figure 4. During this time frame, new functionality was added to the model based on requests from end-users for more detailed information. At the same time, there were almost no changes. Only a small spike

around V10 and V11 (See A in Figure 5) with a changing rate of 10%. These changes coincide with a year-end rollover. Something similar can be seen a year later from version 24 to 25. Some formulas needed to be changed when the model was transferred from one year to another.

Until version 14 there are, except for the year-end rollover, no changes in the existing formulas. This changed in version 15, from that moment the changing rate continuously stays at a level of about 40%. We discussed this with the creator of the spreadsheet. In V14 a set of formulas were added that needed to be adjusted every time data for a new month was added. Therefore, it stands out that in V19 the changing rate suddenly drops back to 0%. In this version, new data for the month was added, but the adjustments in the formulas were forgotten.

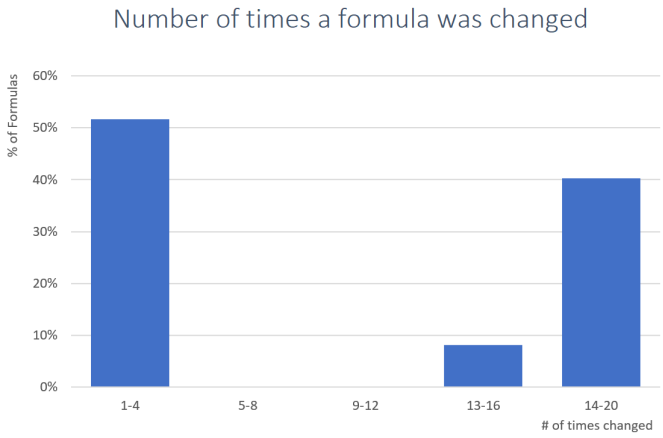


Fig. 6. Frequency distribution of formula changes

In the 35 versions of the spreadsheet, 617 changes to unique formulas were detected. These 617 changes were made to only 62 unique formulas, meaning that a lot of unique formulas were changed multiple times. Figure 6 shows the frequency distribution of these changes. About half of the formulas were changed between one and four times, the other half between thirteen and twenty times. This relates to the set of formulas that had to be changed every month. It started in V15 with 25 formulas and was later, in V20, extended to 30 formulas.

During the interview, the creator of the spreadsheet summarized the changes that were made in the 35 different versions as providing gradually more information to the end-users of the spreadsheet. The spreadsheet is part of a monthly reporting cycle and by providing more information a lot of recurring questions could already be answered based on the information in the spreadsheet. The change that was remembered the most coincides with the spike in the changing rate in V24 (see Figure 5).

According to the creator of the spreadsheet, the most complex part of the spreadsheet are the formulas that calculate a forecast for the coming months. From the 617 changes, only 3 were related to this part of the spreadsheet. In this case, the complexity of a formula was not a driver for frequent change.

The main reasons for changing formulas in the spreadsheet, mentioned during the interview, are 1) incorporating new requests from the end-users of the spreadsheet or 2) changing formulas to make future updates and maintenance easier. The creator of the spreadsheet is well aware of the fact that there are areas in the spreadsheet that could be improved. It should not be necessary to update formulas every time data is added to the spreadsheet. However, it is not always easy to find the time to implement such changes.

C. Case Study II: Failure analysis of the Medium Voltage Grid

The second case study is a set of spreadsheets providing Alliander with an analysis and forecast of failures in the Medium Voltage Electricity Grid. The set consists of 38 spreadsheets with a time span of 38 months, from January 2014 to February 2017.

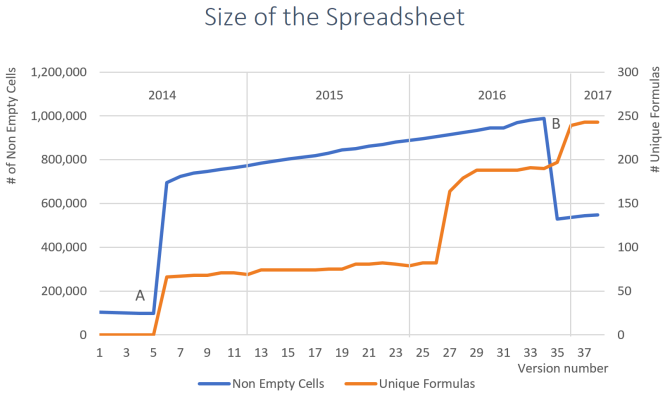


Fig. 7. Evolution of size of the spreadsheets over several versions

The evolution of size for the complete set of spreadsheets is shown in Figure 7. The data component in this spreadsheet is much larger in comparison with the spreadsheets from the first case study. The chart indicates two moments (A and B in Figure 7) where the number of non-empty cells suddenly changes. In June 2014 the number of cells changes from 99,091 cells to 695,471. In the interview, we asked the creator about the reason for this change. Until that time the analysis was based on the last twelve months of data. As from June 2014, this was extended to a time span of five years.

In November 2016 a large part of the data is deleted, reducing the size of the model to 529,853 cells. First, we assumed that old data was removed from the model. We discussed this during the interview and it turned out that this was only a partial explanation of the size reduction. Indeed one year of data was deleted but at that time the model contained almost one million non-empty cells and it started to get really slow. Closer inspection of the source data revealed that it was possible to reduce it by filtering out rows with a status that was not needed for the calculations. It was this filtering that was mainly responsible for the size reduction.

The model started with one pivot table and without formulas. in Figure 7 we can see that there were three distinctive moments that new unique formulas were added to the model:

mid-2014, early 2016 and late 2016. In all cases, the reasons behind the addition of new formulas were requests for more insights from the end-users of the spreadsheet. The ratio of unique formulas to formulas is about 1 to 1,000, which is much higher than in the first case study where it was about 1 to 25. This high ratio can be explained by the fact that the model contains a data set of about 10,000 rows and that formulas are defined for each individual row. This means one unique formula with 9,999 siblings.

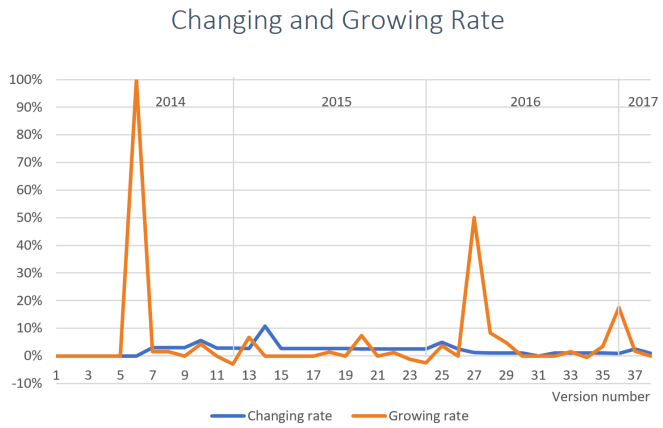


Fig. 8. Changing and growing rate of the spreadsheets over several versions

The changing and growing rate are shown in Figure 8. The three moments when formulas were added to the model are clearly recognizable. In comparison with the first case study, this model is much more stable. The changing rate stays below ten percent for most of the time. However, also in this model, there are formulas changed in almost every version. Figure 9 illustrates this.

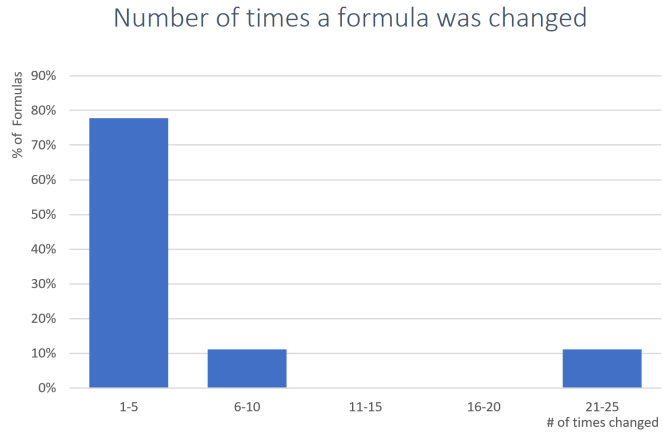


Fig. 9. Distribution of change frequency of formulas

In the 38 versions of the spreadsheet 76 changes were made to only 18 unique formulas. The majority of these formulas were only changed once but there are four formulas that were changed multiple times. We discussed this in the interview with the creator of the spreadsheet. There were two formulas that had to be changed every month. The range that was used in these formulas to calculate a forecast had to be adjusted. The other two formulas had to change only once a year. The logic for the calculation of a KPI was different in the last month of the year than in the other months. Finally, we saw that at the end of 2016 a set of four new formulas were added that had the year hardcoded in the formula. These formulas had to be changed at the beginning of 2017.

This model started with one pivot table. When we asked the creator of the spreadsheet to summarize the development of the model in the 38 consecutive versions, he explained that most changes had to do with new information requests from the end-users of the spreadsheet, but also to replace the pivot table with formulas to mitigate the risk that the spreadsheet was updated without updating the pivot table and therefore presenting information that was outdated.

According to the creator of the spreadsheet, the most complex part of the model was related to the calculation of the long-term and short-term trend of the failure analysis. Although complex, the formulas used for this calculation were stable. They were not changed in any of the 38 spreadsheets.

The changes that were remembered the most by the creator of the spreadsheet, coincide with the three spikes in the growth rate in Figure 8.

The creator of the spreadsheet was well aware of the formulas that needed maintenance every month and also possesses the knowledge of how to change them. Finding the time and priority is, in this case, the limiting factor.

D. Conclusion

Based on the information collected during the case studies, we revisit the research questions.

RQ1 How do spreadsheets evolve over time? In both cases, it is clear that the spreadsheets grow over time, both in the number of non-empty cells as unique formulas. It also became clear that, for the cases considered, there is no direct relation between the growth in data and the growth in unique formulas. Spreadsheets that exists for a longer time-span are often used for reporting purposes. The growth of the data is caused by adding more data points to the analysis. Growth in unique formulas is caused by adding new functionality to the spreadsheet.

RQ2 How common are changes in formulas during the life-span of a spreadsheet? For both cases, unique formulas were changed in almost every version. Only the number of changes differed between the two cases. The addition of new data points made it necessary to change the logic of some of the formulas. In the context of a spreadsheet this sounds reasonable but translated to the context of software evolution it would mean that source code need to be changed every time the program is run with new data.

RQ3 What are the reasons for the changes? The motivation for most changes in the formulas is new feature requests from the end-users of the spreadsheet. Another reason for change is to improve the maintainability of the spreadsheet. A third reason, that was not mentioned during the interviews,

but of which we found examples in our evolution study, is the correction of an error that was made in a previous version.

RQ4 To what extent can the results of a spreadsheet evolution study, support end-users in creating spreadsheets that are easier to maintain and contain fewer errors? When we discussed the results of the evolution study with the creators of the spreadsheet, several suggestions were made about how these results could support a spreadsheet user in making a better model:

- Summarize the changes that are made in a new version. It helps the creator of the spreadsheet to detect if all changes are intended and correctly implemented. Choosing to only show the changes in unique formulas instead of all formulas helps to present the changes in a concise way to the creator of the spreadsheet.
- Provide the creator of the spreadsheet with a list of formulas that were frequently changed in earlier versions of the spreadsheet. This could function as a checklist to make sure that all necessary changes have been made
- Suggest formulas that are candidates for refactoring. Formulas that have to be changed in every new version can often be rewritten in such a way that changes are not necessary.
- Highlight sudden drops or spikes in changing and growing rates. They sometimes indicate anomalies in the spreadsheet. For example, a sudden drop in the changing rate from 32% to 0% between V18 and V19 in case study I revealed a set of formulas that should have been updated but were not.

V. Discussion

A. Threats to validity

The cooperation with Alliander gave us a unique opportunity to study spreadsheet evolution in real-life scenarios. However, a real-life dataset comes with the price of reduced repeatability. We strongly support open data, but because the spreadsheets contain sensitive information, we are not able to share them.

A threat to the external validity of our study concerns the representativeness of the selected set of spreadsheets that we analyzed in this paper. The aim of this study is to gain a better understanding of the evolution of spreadsheets. The direct access to the creators of the spreadsheet is of decisive importance to obtain this understanding, but also sets a practical limit on the number of spreadsheets that could be included. However, if we look at properties, like size and number of formulas, of the spreadsheets then they are comparable to the spreadsheets in the Enron corpus [**?**] and with this respect, they seem to be representable.

B. VBA Code, Pivot Tables, and Charts

In this study, we have chosen to study the evolution of spreadsheets by analyzing the changes in formulas. They determine the functionality of the spreadsheet and it is in the formulas that most errors occur. However, there are other components in spreadsheets that evolve over time, such as pivot tables, charts, and VBA code. In future research, we will shift our focus to these components.

VI. Concluding Remarks

The aim of this paper is to get a better understanding of the evolution of spreadsheets. In the two case studies, we saw that spreadsheets grow over time, both in data as in the number of formulas. The main drivers for the growth of the formulas are new feature requests from the end-users of the spreadsheet. Besides new feature requests also improving the maintainability was mentioned as a motivation to implement changes and finally, we saw that some changes were related to bug fixing. In both case studies, there was a certain percentage of formulas (48% in study I and 22% in study II) that changed in almost every version. Results show that these formulas had to be adjusted when new data points were added to the spreadsheet.

The contributions of this paper are as follows:

- FormulaMatch: an algorithm to match unique formulas of two different versions of the same spreadsheet (Section III).
- Two case studies in which a detailed study is made of the evolution of a spreadsheet. In both cases, the analyzed spreadsheets had a time-span of three years (Section IV).
- Insights from spreadsheet users about how the results of an evolution study can support them in creating better spreadsheets (Section IV).

This research gives rise to several directions for future work. The FormulaMatch algorithm in combination with the concept of unique formulas makes it possible to present in a concise way the changes between two versions of a spreadsheet. We will research if it is possible to further improve the results by combining the FormulaMatch algorithm with other algorithms that detect structural changes in spreadsheets. Furthermore, the case studies showed that measuring the change frequency of formulas over the life-span of a spreadsheet supports 1) the identification of formulas that are good candidates for refactoring or 2) predicting which formulas need to be changed in the version on hand. We are planning to develop a tool that uses this change frequency analysis to present this information to the user. Finally, we will study the evolution of VBA code, pivot tables, and charts within a spreadsheet.

References

[1] R. R. Panko and N. Ordway, "Sarbanes-oxley: What about all the spreadsheets?" *arXiv preprint arXiv:0804.0797*, 2008.

[2] W. Winston, "Executive education opportunities millions of analysts need training in spreadsheet modeling, optimization, monte carlo simulation and data analysis," *OR MS TODAY*, vol. 28, no. 4, pp. 36–39, 2001.

[3] F. Hermans, B. Jansen, S. Roy, E. Aivaloglou, A. Swidan, and D. Hoepelman, "Spreadsheets are code: An overview of software engineering approaches applied to spreadsheets," in *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2016.

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

[5] K. J. Rothermel, C. R. Cook, M. Burnett, J. Schonfeld, T. R. Green, and G. Rothermel, "Wysiwyt testing in the spreadsheet paradigm: An empirical evaluation," in *Software Engineering, 2000. Proceedings of the 2000 International Conference on*. IEEE, 2000, pp. 230–239.

[6] S. Roy, F. Hermans, and A. van Deursen, "Spreadsheet testing in practice," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 2017, pp. 338–348.

[7] F. Hermans, M. Pinzger, and A. Deursen, *ECOOP 2010 – Object-Oriented Programming: 24th European Conference, Maribor, Slovenia, June 21-25, 2010. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ch. Automatically Extracting Class Diagrams from Spreadsheets, pp. 52–75. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14107-2_4

[8] J. Cunha, M. Erwig, and J. Saraiva, "Automatically inferring classsheet models from spreadsheets," in *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*. IEEE, 2010, pp. 93–100.

[9] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting and refactoring code smells in spreadsheet formulas," *Empirical Software Engineering*, pp. 1–27, 2014.

[10] J. Cunha, J. P. Fernandes, H. Ribeiro, and J. Saraiva, "Towards a catalog of spreadsheet smells," in *Computational Science and Its Applications–ICCSA 2012*. Springer, 2012, pp. 202–216.

[11] D. W. Barowy, D. Gochev, and E. D. Berger, "Checkcell: Data debugging for spreadsheets," in *ACM SIGPLAN Notices*, vol. 49, no. 10. ACM, 2014, pp. 507–523.

[12] F. Hermans and D. Dig, "Bumblebee: a refactoring environment for spreadsheet formulas," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 747–750.

[13] S. Badame and D. Dig, "Refactoring meets spreadsheet formulas," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 399–409.

[14] F. Hermans, "Analyzing and visualizing spreadsheets," Ph.D. dissertation, PhD thesis, Software Engineering Research Group, Delft University of Technology, Netherlands, 2012.

[15] M. M. Lehman and J. F. Ramil, "Software evolution—background, theory, practice," *Information Processing Letters*, vol. 88, no. 1-2, pp. 33–44, 2003.

[16] R. L. Novais, A. Torres, T. S. Mendes, M. Mendonça, and N. Zazworka, "Software evolution visualization: A systematic mapping study," *Information and Software Technology*, vol. 55, no. 11, pp. 1860–1883, 2013.

[17] H. Gall, M. Jazayeri, R. R. Klosch, and G. Trausmuth, "Software evolution observations based on product release history," in *Software Maintenance, 1997. Proceedings., International Conference on*. IEEE, 1997, pp. 160–166.

[18] W. Dou, L. Xu, S.-C. Cheung, C. Gao, J. Wei, and T. Huang, "Venron: a versioned spreadsheet corpus and related evolution analysis," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 162–171.

[19] M. M. Lehman, "The programming process," *internal IBM report*, 1969.

[20] ——, "Programs, cities, students—limits to growth?" in *Programming Methodology*. Springer, 1978, pp. 42–69.

[21] B. Jansen and F. Hermans, "Code smells in spreadsheet formulas revisited on an industrial dataset," in *Proceedings of the 2015 International Conference on Software Maintenance and Evolution*. IEEE Press, 2015, pp. 372–380.

[22] B. Klimt and Y. Yang, "Introducing the Enron corpus." in *CEAS*, 2004.

[23] L. Xu, W. Dou, C. Gao, J. Wang, J. Wei, H. Zhong, and T. Huang, "Spreadcluster: recovering versioned spreadsheets through similarity-based clustering," in *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*. IEEE, 2017, pp. 158–169.

[24] C. Chambers, M. Erwig, and M. Luckey, "Sheetdiff: A tool for identifying changes in spreadsheets," in *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*. IEEE, 2010, pp. 85–92.

[25] A. Harutyunyan, G. Borradaile, C. Chambers, and C. Scaffidi, "Planted-model evaluation of algorithms for identifying differences between spreadsheets," in *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*. IEEE, 2012, pp. 7–14.

[26] J. Sajaniemi, "Modeling spreadsheet audit: A rigorous approach to automatic visualization," *Journal of Visual Languages & Computing*, vol. 11, no. 1, pp. 49–82, 2000.

[27] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[28] B. Fluri, M. Wuersch, M. PInzger, and H. Gall, "Change distilling: Tree differencing for fine-grained source code change extraction," *IEEE Transactions on software engineering*, vol. 33, no. 11, 2007.

[29] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[30] J. W. Creswell and J. D. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017.