SheetDiff: A Tool for Identifying Changes in Spreadsheets*

Chris Chambers
Oregon State University
chambech@eecs.oregonstate.edu

Martin Erwig
Oregon State University
erwig@eecs.oregonstate.edu

Markus Luckey
Universität Paderborn
markus.luckey@upb.de

Abstract

Most spreadsheets, like other software, change over time. A frequently occurring scenario is the repeated reuse and adaptation of spreadsheets from one project to another. If several versions of one spreadsheet for grading/budgeting/etc. have accumulated, it is often not obvious which one to choose for the next project. In situations like these, an understanding of how two versions of a spreadsheet differ is crucial to make an informed choice. Other scenarios are the reconciliation of two spreadsheets created by different users, generalizing different spreadsheets into a common template, or simply understanding and documenting the evolution of a spreadsheet over time.

In this paper we present a method for identifying the changes between two spreadsheets with the explicit goal of presenting them to users in a concise form. We have implemented a prototype system, called SheetDiff, and tested the approach on several different spreadsheet pairs. As our evaluations will show, this system works reliably in practice. Moreover, we have compared SheetDiff to similar systems that are commercially available. An important difference is that while all these other tools distribute the change representation over two spreadsheets, our system displays all changes in the context of one spreadsheet, which results in a more compact representation.

1. Introduction

Studies have shown that each year hundreds of millions of spreadsheets are created, making spreadsheets one of the most widely used end-user programming environments [10]. As the end-user programmers creating these spreadsheets may not have a sound background in software design, development, or maintenance, many of these spreadsheets end up containing errors. In fact some studies report that up to 90% of real world spreadsheets contain errors [9]. This has motivated research efforts in the area of software engineering and end-user programming with the goal to support end user in their work to create reliable spreadsheets. Much of this research has focused on adapting successful programming and software engineering methods to the area of spreadsheets, such as type checking, design, program generation, testing, and debugging (for an overview and a list of references, see [5]).

However, one important area of software engineering that has not yet received much attention in the research community is how to support the *reuse* of spreadsheets. An essential component of the software reuse process is comparing artifacts. The importance of change inference for spreadsheet reuse and beyond is illustrated by the following examples.

Imagine that there is a spreadsheet that is shared amongst three different co-workers. Whenever a change is made there is no easy way for other users to see what has happened. There are ways to document these changes, such as change logs or Excel's track changes functionality, but these methods suffer from two fundamental limitations.

First, change tracking has to be actively enabled by a user, otherwise no changes will be recorded. In other words, for change tracking to be an effective tool its use must be carefully planned in advance, and also must be employed consistently by all users of a spreadsheet. Second, in the case of multiple spreadsheet versions that evolve in parallel, change tracking is only of limited use and can not show differences between arbitrary versions. For example, consider a spreadsheet S with change tracking enabled, which is then changed by three users to three different versions T, U, and V, respectively. While change tracking can show the differences for T, U, and V with respect to S, it cannot show the differences between T and U, T and V, or U and V. In contrast, a tool for inferring spreadsheet changes allows a user to always compare any two spreadsheets without prior arrangements, and so, for example, reveals any changes in T, U, and V compared to S and compared to each other.

In a completely different scenario, such a change inference tool would allow teachers to create an oracle spreadsheet, with the correct formulas and calculations in place.

*This work is partially supported by the National Science Foundation under the grant CCF-0917092 and by the Air Force Office of Scientific Research under the grant FA9550-09-1-0229.

This oracle could then be compared a students result to determine similarity. Any cells containing the incorrect formula or result would be shown, and the teacher could use this to mark the ones that are incorrect.

Finally, in addition to comparing just two versions of a spreadsheet, change inference can also reveal trends and patterns about the evolution of a spreadsheet over time. If several versions of the same spreadsheet exist, these can be compared to see the incremental change of the spreadsheet in each version, and these changes taken together provide a greater understanding of the spreadsheet and its evolution.

Each of these examples demonstrate how spreadsheet change inference allows users to put the changes in context and better understand what the spreadsheet is doing and how it is evolving. This better understanding supports users in modifying spreadsheets correctly.

In this paper we describe SheetDiff, a tool that we have developed for identifying changes between two spreadsheets and presenting them in a succinct form. SheetDiff shows the changes that are needed to get from one spreadsheet to the other. The changes are shown "in place", which supports not only the easy identification of what has changed, but also where the changes have occurred.

The rest of this paper is structured as follows. Related work is discussed in Section 2. In Section 3 we illustrate the issues involved in identifying and presenting spreadsheet changes with a small example. The algorithm for inferring spreadsheet changes is then described in Section 4. In Section 5 we present an evaluation of the SheetDiff tool on several examples. We discuss one particularly promising area of future research in Section 6, and finish with conclusions in Section 7.

2. Related Work

Much of the previous research on spreadsheets has focused on ways to detect and correct errors [5]. However, very little research has been done on identifying and reporting the differences between two related spreadsheets.

If a spreadsheet with an error is reused or shared, the error is propagated amongst the different users. One field study [3] notes that errors in spreadsheets can lead to bad decision making amongst business managers and major monetary losses. This would indicate that before spreadsheets are widely reused and shared it is important to see what has been changed, which will give users a better understanding of how the spreadsheet has been changed and also indicate where possible errors may be.

One systematic approach to manage reuse is through templates [11]. There has been some work in the development of an automatic spreadsheet template tools [6, 4], which allow users to create a spreadsheet template that can be shared and reused. This template provides a structure that must be followed for the spreadsheet to be correct. However, these tools can be rather rigid and as they are not built into Excel many users may have trouble adapting them for widespread use.

There are three related systems that attempt to identify changes in spreadsheets. The first system is Excel's built in functionality, Track Changes, the other two, DiffEngineX and Synkronizer, are commercial products that are used to compare two spreadsheets and report the changes to users. These three systems are presented and discussed below.

2.1 Excel's Track Changes

Microsoft Excel has built-in functionality that allows users to highlight the changes that they are currently making to a spreadsheet, such as deleted or added rows and columns as well as edited cells. This functionality can be particularly useful when one wishes to change and then return a spreadsheet to another user. However, it does have a few problems.

First, change tracking lacks the ability to compare two spreadsheets; it can only track changes that are currently being made. If it is not turned on, none of the changes will be marked making it appear as if nothing had been changed.

Second, the visualization for changes is rather poor. Figure 1 shows the output of track changes. While this shows all the changes, it is misleading as the bold line between rows three and four represent the removal of two rows. However, this line would be the same if any number of rows had been removed.

	A	B	C	D	E	F	G
1					Year		
2	**Type**	**Detail**	**Value**		**2007**	**2008**	**2009**
3	Capacity	Tower	50,000	bbl.	50,000	89,791	23,490
4	Yield	Low-End fr. Crude	0.60		0.60	0.65	0.55
5	Contract	Crude	36,000	bbl.	36,000	34,000	6

Figure 1. Sample Track Changes Output

2.2 DiffEngineX and Synkronizer

DiffEngineX [8] is a commercial product that can be used to compare and highlight the differences between two spreadsheets. The output of this system is displayed on both of the spreadsheets, with the colors representing the changes based on the other one. An example output is shown in Figure 2.

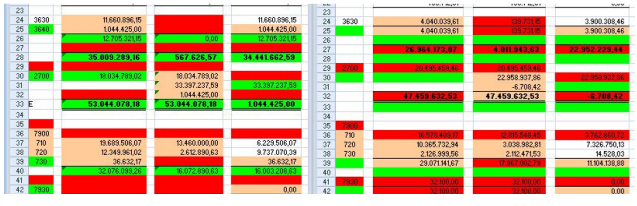

Figure 2. Sample Output from DiffEngineX

There are several options that DiffEngineX uses that assists in spreadsheet comparison. Among these is the ability to compare formula cells by either their value or the formula it contains and the ability to align rows. With these options DiffEngineX is a powerful tool for comparing spreadsheets.

	A	B	C	D	E	F	G
1					Year		
2	**Type**	**Detail**	**Value**		**2007**	**2008**	**2009**
3	Capacity	Tower	50,000	bbl.	50,000	89,791	23,490
4	Capacity	Cracker	20,000	bbl.	20,000	1,324	123,434
5	Yield	Distillate fr. Crude	0.40		0.40	0.43	0.44
6	Yield	Low-End fr. Crude	0.60		0.60	0.65	0.55
7	Contract	Crude	36,000	bbl.	36,000	36,000	6
8	Demand	Regular gas	5,000	bbl.	5,000	5,600	5,800
9	Demand	Premium gas	10,000	bbl.	10,000	10,987	981
10	Quality	Min Cat in Reg	0.50		0.50	0.50	0.50
11	Quality	Min Cat in Prem	0.65		0.65	0.65	0.65
12	Cost	Crude	$25.00	per bbl.	$28.00	$35.74	$35.74
13	Price	Low-End	$28.00	per bbl.	$25.00	$35.06	$34.87
14	Price	Regular gas	$40.00	per bbl.	$40.00	$58.77	$58.77
15							
16	Profit	Annual net income	$31,988		$31,751		
17	Efficiency	Tower utilization	5%		100%		
18	Efficiency	Cracker utilization	100%		53%		
19	Sales	Low-End	10,610		21,600		
20	Sales	Regular	21,600		580		
21	Sales	Premium	580		5,601		
22	Sales	Hi-End	10,000		4,244		

	A	B	C	D	E	F	G	H
1					Year			Projected
2	**Type**	**Detail**	**Value**		**2007**	**2008**	**2009**	**2010**
3	Capacity	Tower	50,000	bbl.	50,000	89,791	23,490	40,090
4	Capacity	Cracker	20,000	bbl.	20,000	1,542	23,800	900
5	Yield	Distillate fr. Crude	0.40		0.40	0.43	0.44	0.41
6	Yield	Low-End fr. Crude	0.60		0.60	0.65	0.55	0.62
7	Contract	Crude	36,000	bbl.	36,000	36,000	1,500	45,000
8	Demand	Regular gas	5,000	bbl.	5,000	5,600	5,800	6,500
9	Quality	Min Cat in Reg	0.50		0.50	0.50	0.50	0.65
10	Quality	Min Cat in Prem	0.65		0.65	0.65	0.65	40.00
11	Cost	Non-Crude	$24.00		$27.00	$28.00	$33.00	$40.00
12	Cost	Crude	$25.00	per bbl.	$28.00	$35.74	$35.74	$38.00
13	Price	Low-End	$28.00	per bbl.	$25.00	$35.06	$34.87	$33.50
14	Price	Regular gas	$40.00	per bbl.	$40.00	$58.77	$75.00	$100.00
15								
16	Profit	Annual net income	$31,988		$31,751			
17	Efficiency	Tower utilization	5%		100%			
18	Efficiency	Cracker utilization	100%		53%			
19	Sales	Low-End	10,610		21,600			
20	Sales	Regular	21,600		580			
21	Sales	Premium	580		5,601			
22	Sales	Hi-End	10,000		4,244			

Figure 4. Original and Updated Spreadsheets

One of the major problems with this system is that it represents the changes by coloring two different spreadsheets. This requires a user to continuously switch between the two sheets and rely on their memory to judge the differences. When the changes are shown in context on one sheet the comparison can be done visually.

Synkronizer [1] is another spreadsheet difference product, and while also showing changes in two separate spreadsheets, its output tends to be much easier to understand since it can group row and column changes. An example of Synkronizer is shown in Figure 3.

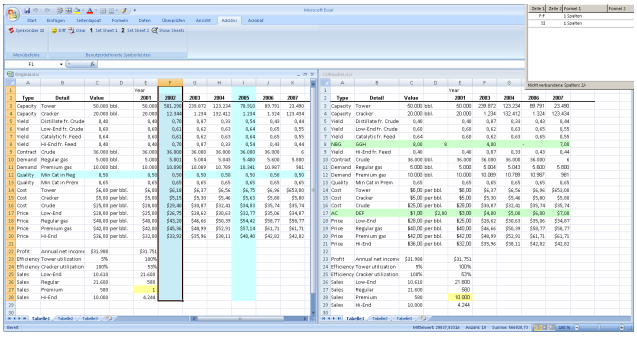

Figure 3. Sample Output from Synkronizer

3. Finding and Presenting Differences Between Spreadsheets

Consider the spreadsheets shown in Figure 4 that shows two different versions of a budget sheet. The original spreadsheet, shown on the left, has been updated to include projections and new products. The updated spreadsheet, shown on the right, is very similar to the original, and at first glance it is not easy to spot all the changes that have been made. Upon closer inspection it could be noticed that a column has been added, and some changes have occurred with two rows. However, visual comparisons can be time consuming and are not guaranteed to catch every change.

While the spreadsheets in Figure 4 are small enough that a visual comparison might be feasible, as the size of a spreadsheet grows, such a comparison becomes almost impossible, and a tool for the automatic inference of spreadsheet differences becomes a necessity.

An important design question for such a tool is: How should the changes between two spreadsheets be presented to the user? There are at least three different approaches conceivable: First, one could list the addresses of cells, rows, and columns in which changes have occurred. Second, one could present both spreadsheets side-by-side and mark cells in each version that are different in the other. As discussed in Section 2, this approach is taken by several commercially available tools. Third, one could present one of the spreadsheets and annotate it in place so that it is shown what parts differ from the other spreadsheet.

The third option is the approach we have pursued in the design of SheetDiff since it (a) provides the change information in the context of the rest of the spreadsheet, that is, in the context of the unchanged parts, and (b) it does not require a constant back-and-forth focus switching between two spreadsheets as in the second approach. With this design decision in mind, change inference can be performed in two distinct steps.

The first step finds all the cell differences, by comparing the values of the cells. For example, the value in cell G4 is 123,434 in the original spreadsheet and 23,800 in the updated spreadsheet. This would cause the cell to be marked as changed. The second step looks at all the cell changes that were found and groups them into an easier-to-understand set of changes. For example, since all the cells in the column H have been added, it makes more sense to mark this as an added column rather than simply marking every cell in the column as individually changed.

By employing this methodology we can locate the changes in the new spreadsheet and display them to the user. The results, shown in Figure 5, indicate seven changes to the spreadsheet. One column and one row has been added

	A	B	C	D	E	F	G	H
1					Year			Projected
2	**Type**	**Detail**	**Value**		**2007**	**2008**	**2009**	**2010**
3	Capacity	Tower	50,000	bbl.	50,000	89,791	23,490	40,090
4	Capacity	Cracker	20,000	bbl.	20,000	1,324	123,434	900
5	Yield	Distillate fr. Crude	0.40		0.40	0.43	0.44	0.41
6	Yield	Low-End fr. Crude	0.60		0.60	0.65	0.55	0.62
7	Contract	Crude	36,000	bbl.	36,000	36,000	6	45,000
8	Demand	Regular gas	5,000	bbl.	5,000	5,600	5,800	6,500
9	Demand	Premium gas	10,000	bbl.	10,000	10,987	981	
10	Quality	Min Cat in Reg	0.50		0.50	0.50	0.50	0.50
11	Quality	Min Cat in Prem	0.65		0.65	0.65	0.65	0.65
12	Cost	Non-Crude	24.00		27.00	28.00	33.00	40.00
13	Cost	Crude	$25.00	per bbl.	$28.00	$35.74	$35.74	$38.00
14	Price	Low-End	$28.00	per bbl.	$25.00	$35.06	$34.87	$33.50
15	Price	Regular gas	$40.00	per bbl.	$40.00	$58.77	$58.77	$100.00
16								
17	Profit	Annual net income	$31,988		$31,751			
18	Efficiency	Tower utilization	5%		100%			
19	Efficiency	Cracker utilization	100%		53%			
20	Sales	Low-End	10,610		21,600			
21	Sales	Regular	21,600		580			
22	Sales	Premium	580		5,601			
23	Sales	Hi-End	10,000		4,244			

Figure 5. The Results of SheetDiff

(shown with the blue highlight in column H and row 12), one row has been deleted (denoted by the red highlight in row 9), and four individual cells have been changed (denoted with orange cells). The black cell, located at the intersection of the added column and the deleted row, indicates that there was no value in this cell.

4. Change Inference

The SheetDiff change inference algorithm operates in the following two interrelated phases.

- Compare individual cells
- Optimize cell changes into higher-level changes

While the first step is fairly simple, the inference of higher-level changes is generally ambiguous and not straightforward. For example, in the case that the majority of the cells in a row are changed, does this signify that a row has been added or simply that many of the cells have been changed? Apart from the question of how to best represent such a change, a decision about how to parse and present sets of changes also has an impact on the change inference of the rest of the spreadsheets, and it generally requires that the results of individual cell comparisons must be dynamically adapted.

To understand this latter point, consider a spreadsheet S that contains in column A and row i the number i (for $1 \leq i \leq 20$). Now consider spreadsheet T in which cell A1 contains 1 and column A, row i contains the value $i+1$ (for $2 \leq i \leq 19$). A comparison of the individual cells yields changes in all cells A2, A3, ..., A20. Now we can observe that the change in cell A2 (2 changed to 3) is, in fact, a change of the whole row 2. More specifically, the deletion of row 2 in S yields the current value of A2 in spreadsheet T. But what is more important in this case is the fact that by identifying the deletion of row 2 as a change from S to T, the remaining individual cell changes disappear. This can be seen as follows. The deletion of row 2 caused all cells in T to move up one row. Therefore, the remaining rows of T (from row 2 on upwards) have to be compared with rows in S that have their row index increased by 1, that is, we have to compare A2 in T with A3 in S, A3 in T with A4 in S, and so on. All these comparisons do not yield any change.

This small example illustrates two important aspects of change inference.

- First, the information about individual cell changes is changed dynamically with the identification of higher-level changes and has to be recomputed in some cases.
- Second, the number of changes to be reported can be reduced (in some cases enormously) through the identification of higher-level changes.

It also seems that in order to avoid redundancy individual cell changes should be computed only incrementally up to the point where a higher-level change (that is, a row or column insertion or deletion) can be detected. However, this works only if higher-level changes are considered in a strict ordering (for example, top-down, left-right). In one version of the change inference algorithm, we actually consider the inference of higher-level changes based on how promising they are, irrespective of where they occur in the spreadsheet. To realize this strategy, we need access to all individual cell changes.

In the following we will describe the change inference algorithm in more detail. For two given spreadsheets, S and T, let W (H) be the maximum width (height) of any row (column) in S or T. All the individual cell changes between S and T can be determined by applying a simple traversal function $\Delta(S,T)$ that produces a set of changes that will be kept in a set δ. These changes can be grouped by row and column, and the sets $R(r)$ ($C(c)$) denote the set of changes in row r (column c). In step 4 of the algorithm, we use the notation $S \otimes x$ for $\otimes \in \{+,-\}$ and $x \in \{r,c\}$ to denote the result of applying different possible high-level changes "$\otimes x$" to spreadsheet S, namely adding or deleting a row or column.

ALGORITHM SheetDiff.
INPUT: Two spreadsheets S and T.
OUTPUT: A set of changes δ, initially set to $\varnothing$.

Step 1. Determine all individual cell changes between S and T, that is, let $\delta = \Delta(S,T)$.

Step 2. Group the changes in δ by rows and columns of their addresses, which yields a mappings R and C.

Step 3. Select the first row r for which $|R(r)|/W > p$ where p is a fixed threshold that is required for identifying higher-level row and column changes.

Similarly, select the first column c for which $|C(c)|/H > p$. If neither r nor c can be found, stop.

Step 4. If r and c were successfully determined in step 3, consider what effect the addition and deletion of row r and column c has in terms of simplifying the set of reportable changes δ. (If only one of r or c is available, consider only the respective case.)

Compute four new spreadsheets and change sets as follows. $S^{\otimes x} = S \otimes x$ and $\delta^{\otimes x} = \{\otimes_x\} \cup \Delta(S^{\otimes x}, T)$. Find the the smallest change set $\delta^{\otimes x}$, and let $S := S^{\otimes x}$ and $\delta := \delta^{\otimes x}$. Continue with step 1. □

We illustrate the algorithm, using the optimum p value of 70%, with the help of a small example. Consider the two spreadsheets, shown in Figure 6.

	A	B
1	Jack	Walking
2	Jim	Swimming
3	Jill	Biking

	A	B	C
1	Jack	Mon, Wed	Running
2	Jill	Tue, Sat	Biking

Figure 6. Two Spreadsheets to be Compared

In step 1 individual cell differences are computed, producing changes in δ for the cells A2, A3, B1, B2, B3, C1, and C2. The grouping in step 2 yields the mappings (for brevity we show only the addresses of the changes).

$$R = \{(1,\{B1,C1\}),(2,\{A2,B2,C2\}),(3,\{A3,B3\})\}$$
$$C = \{(A,\{A2,A3\}),(B,\{B1,B2,B3\}),(C,\{C1,C2\})\}$$

In step 3 we select the first row and column that contain over 70% changes (i.e. p=0.7). In this example row 2 and column B qualify. Step 4 considers the effect of adding or deleting row 2 or column B and generates four corresponding new versions of the first spreadsheet and the following versions of δ.

δ^{+2} = {$+_2$, B1, B2, C1, C2, D1, D2}
δ^{-2} = {$-_2$, B1, B2, C1, C2}
δ^{+B} = {$+_B$, A2, A3, C1, C2, C3}
δ^{-B} = {$-_B$, A2, A3, B1, B2, C1, C2}

The smallest change set is δ^{-2}. We therefore continue the algorithm in step 2 with $S = S^{-2}$ and $\delta = \delta^{-2}$, which produces the following mappings.

R = {(1, {B1, C1}), (2, {B2, C2})}
C = {(B, {B1, B2}), (C, {C1, C2})}

In this iterations, step 3 yields two rows (rows 1 and 2) and two columns (B and C) that have over 70% changes. Our algorithm only considers the first of those and thus leads to another four different change sets to be considered.

δ^{+1} = {$+_1, -_2$, A2, A3, B2, B3, C2}
δ^{-1} = {$-_1, -_2$, A1, A2, B1, B2, C1, C2}
δ^{+B} = {$+_B, -_2$, C1}
δ^{-B} = {$-_B, -_2$, B1, B2, C1, C2}

In this case, the smallest change set is δ^{+B}, and the algorithm proceeds again with step 2. The change set is now $\delta = \{+_B, -2, C1\}$, which leads to a termination of the algorithm in step 3. The three changes would be visually represented as shown in Figure 7. As we can see column B is highlighted in blue to represent the added column, row 2 is highlighted in red to indicate a deleted row, and cell C1 is shaded orange indicating that the cell value has been changed.

	A	B	C
1	Jack	Mon, Wed	Walking
2	Jim		Swimming
3	Jill	Tue, Sat	Biking

Figure 7. Visual Presentation of Results.

Finally, we note that this basic algorithm can be varied in at least two different ways.

First, the hypothetical consideration of higher-level changes in step 4 could be extended to multiple levels, that is, for each $S^{\otimes x}$ and $\delta^{\otimes x}$ we can compute four versions $(S^{\otimes x})^{\otimes y}$ and $(\delta^{\otimes x})^{\otimes y}$ and then select $S^{\otimes x}$ based on the smallest $(\delta^{\otimes x})^{\otimes y}$. Of course, this "lookahead" can be performed for an arbitrary number of levels. However, this would lead to an exponential running time.

Second, the selection of r and c in step 3 can be driven by the highest ratios $|R(r)|/W$ and $|C(c)|/H$ instead of top-down, left-right ordering.

We have tested several combinations of both variations, neither of which produced significantly better results than the plain version described above. In fact, some combinations performed worse. Moreover, the lookahead versions had, as expected, a higher runtime. We therefore stick to the basic version of the algorithm. Finally, we found that using $p = 0.7$ as a change-ratio threshold for identifying higher-level changes was able to find the most changes consistently (that is, across a mix of small and large spreadsheets with few and many changes).

5. Evaluation

We have implemented SheetDiff as an add-in for Microsoft Excel. Using this add-in we evaluate how the method described in this paper compares spreadsheets. We will also contrast SheetDiff to two commercial products, DiffEngineX and Synkronizer, both of which are described in more detail in Section 2. The evaluation of these tools is used to answer the following research questions.

RQ1: *How effective are the systems at finding changes?* Are the tools able to identify all changes in a spreadsheet correctly? An effective system will find most or all of the changes and show very few unnecessary changes.

RQ2: *How understandable are the presented results?* How the changes are represented is important since results

that make little sense can cause users to become confused. An understandable system will show a minimal number of changes and present it in a way that makes sense. For example, presenting a deleted row as two deleted rows and an added row would not be understandable.

RQ3: *Are there any ways in which SheetDiff could be changed to improve the functionality?*
If there are spreadsheets that our system has problems with, can these be analyzed to determine ways to change the tool in the future to provide better, more compact, easier to understand changes to the user?

5.1 Experiments

To answer these research questions we have gathered two different sets of spreadsheets from the EUSES corpus [7].

The first set of selected spreadsheets consists of sheets in the EUSES repository that are versions of the same sheet. These 8 spreadsheets represent real world examples of how different versions of spreadsheets have changed and will be used to look at how the systems represents changes.

The second set consists of 10 randomly selected spreadsheets to which we applied a specific set of changes to determine whether the tools could identify these. This set will be used to supplement the knowledge gained from the real world examples as well as check the overall correctness of the representation.

To each of the randomly selected spreadsheets we applied four distinct sets of changes to allow us to test the tools in a number of different scenarios. The change sets are listed below.

1. Individual cell changes only: Change randomly selected formula and value cells.
2. Row insertions & cell changes: A small number of rows are inserted, plus random cell changes from (1).
3. Row/column insertions/deletions & cell changes. A varying number of rows and columns are inserted and deleted, plus cell changes from (1).
4. Many cell changes in one row. A high number of changes are made to a single row to test the ratio p of SheetDiff.

The modifications that were applied are based on data gathered from inspecting changes in the spreadsheet pairs of set 1. The changed cells were calculated as a percent of the total number of cells, and the rows and columns changed were set at a small constant depending on the size of the spreadsheet. The last change set was selected to test the robustness of high-level change detection in SheetDiff, that is, we wanted to know: How often does SheetDiff incorrectly identify individual changes as row insertions or deletions?

We then ran our tool and the two commercial products on each of these sets.

For the first set, we determined how many changes were reported by each tool. Since Synkronizer and SheetDiff can identify higher-level changes, we expect lower numbers for these two than for DiffEngineX.

For the second set of spreadsheets we determined the number of correctly identified changes. Each correctly identified change was counted as 1. When a tool reported an inserted or deleted row or column as a set of individual cell changes, we considered this as "somewhat correct" and awarded 0.5 points. We also counted the number of false positives and false negatives.

To answer RQ3 we will look at the cases that cause problems for the commercial systems and SheetDiff in particular. We will determine if there is any solution or simple change to our tool that would make it easier to determine changes in spreadsheets.

5.2 Results

The results of running the three tools on the first set of spreadsheet pairs are shown in Table 1. As this table shows, both SheetDiff and Synkronizer have compact representations, with DiffEngineX, reporting a high number of changes, as expected, due to cell based comparisons.

Table 1. Change Compactness

	SheetDiff		Synkronizer		DiffEngineX	
Pair	**chg**	**err**	**chg**	**err**	**chg**	**err**
1	205	0	623	0	2,190	0
2	57	0	31	5	131	0
3	4	0	12	6	3	0
4	17	0	-	-	200	0
5	17	0	214	0	1,584	0
6	586	0	646	0	2,345	0
7	6	0	6	0	6	0
8	131	0	68	0	131	0
SUM	1,023	0	1,618	11	6,590	0

Compactness is important as it represents the number of changes that will be displayed to users. If all the changes can be represented in fewer operations, this will make the display less cluttered and make it much easier to understand. Since Synkronizer crashed when run on the pair 4, no results could be reported for this case (this also makes the total number of changes look better than it is).

To supplement the numbers from Table 1, we also analyzed the three tools qualitatively and characterized its change representation approach along several categories. The results are shown in Table 2. In this table, context represents how the changes are displayed, a tool shows changes in context if the changes are shown on one sheet. Cell or Line measures how the system determines changes; Cell only detects changes to cells, while Line will detect row and column changes. Workbook comparison is the ability

so check many spreadsheets at once. We also report under "confusing results" cases where the results took effort to decode and where the displayed results looked different than the changes that were actually performed.

Table 2. Change Representation

Tool	SDiff	Synk	DEX
context	yes	no	no
cell or line	both	both	cell
workbook comparison	no	yes	yes
confusing results	2	2	6

SheetDiff is the only system that shows the changes in context, which makes the representation more compact. The other two systems offer only a split view, which requires repeated switching back and forth between two windows to interpret the change results.

The results from running the tools on the second set of spreadsheets is summarized in the following two tables that report the percentage of correctly identified changes for each tool. Table 3 shows the correctness levels for each spreadsheet summarized over all four change sets. We can observed that overall SheetDiff performed well when compared to the other systems. However, Synkronizer is more consistent, with the number being low due to crashing for one test on a spreadsheet pair. However, as this is the second test Synkronizer failed on (it also failed on pair 4 in the first experiment) this may point to Synkronizer being less robust.

As DiffEngineX only reports cell changes, it does poorly in this evaluation since no row and column changes can be identified.

Table 3. Correctness Levels by Sheets

Pair	SheetDiff	Synkronizer	DiffEngineX
1	98%	100%	90%
2	98%	100%	91%
3	93%	100%	81%
4	97%	100%	92%
5	95%	60%	89%
6	92%	100%	83%
7	100%	100%	85%
8	100%	100%	84%
9	100%	100%	85%
10	100%	100%	91%
AVG	98%	96%	88%

To break down which types of change sets gave the systems the most trouble, Table 4 gives the correctly identified changes grouped by the type of change set and aggregated over all spreadsheets. As we can see all three tools are able to detect 100% of the cell changes. The category that gives the most trouble is "Rows/Cols", which are by far the most complex. The crashed test for Synkronizer, again, provides for a lower score.

The test that was designed to trick SheetDiff into incorrectly identifying high-level changes shows that even in some of the worst cases (for our tool), it performs admirably.

Table 4. Correctness Levels by Change Type

Changes	SheetDiff	Synkronizer	DiffEngineX
Cells only	100%	100%	100%
Rows	100%	100%	86%
Rows/Cols	94%	89%	78%
Quasi Row	97%	100%	100%
AVG	98%	96%	88%

5.3 Discussion

As our results show, SheetDiff and Synkronizer are the only tools to identify high-level changes and score therefore relatively high on the correctness test. Even though SheetDiff has a slightly higher overall correctness rate, Synkronizer seems to be more consistent; but in cases when it gets confused (such as spreadsheet 5 in Table 3) it fails more dramatically. (It also crashed on example 4 in the second test.) Thus, with very good correctness results, we can answer RQ1 positively for SheetDiff and Synkronizer.

On RQ2 we see an advantage for SheetDiff since it scores better on change compactness. The context-based representation is not only easier to understand, but allows users to trace back and convert from the new spreadsheet to the original one. While we believe that the contextual representation is more user friendly, this will need to be established by a separate user study.

Finally, regarding RQ3, we have determined several problems with the tested systems. These are discussed in the following.

The first problem is table movement. Take for example the table in the lower left corner in Figure 4. Moving the block one to the right, SheetDiff will report many added and deleted rows. While technically correct, this could be improved. The other systems have even more problems with this, with Synkronizer even crashing.

The second problem is string comparison. There are several cases were the only difference between a cell is a trailing space. What is the difference in the value "Report " compared to "Report"? All three systems had this problem, and while technically correct, this can result in confusing change representations and could be improved by a more refined cell comparison method.

The final problem is that all three systems ignore row and column headers. There has been work done into how to determine headers for spreadsheets [2]. However, none of these systems leverage this. If the headers were determined

for rows and columns then changed rows and columns may be able to be determined with higher likelihood. For example, if the original spreadsheet has column headers “2001”, “2003”, and “2005” and the new spreadsheet has column headers “2001”, “2003”, “2004”, and “2005” it would be very likely that the column with the header “2004” was added.

6. Future Work

Currently, all three tools can compare only two sheets at a time. But the contextual, in-place change representation of SheetDiff leads to an interesting extension of the system that is able to compare a range of spreadsheets. As an example consider a budget sheets that is extended every year. A mock up of this functionality is shown in Figure 8.

	A	B	C	D	E	F	G	H	I	J	K
1					Year						
2	**Type**	**Detail**	**Value**		**2005**	**2006**	**2007**	**2007**	**2008**	**2009**	**2010**
3	Capacity	Tower	50,000	bbl.	50,000	45,000	30,000	78,000	89,791	23,490	40,090
4	Capacity	Cracker	20,000	bbl.	20,000	15,000	15,000	1,000			
5	Yield	Distillate fr. Crude	0.40		0.40	0.40	0.40	0.40	0.43	0.44	0.41
6	Yield	Low-End fr. Crude	0.60		0.60	0.60	0.60	0.60	0.65	0.55	0.62
7	Contract	Crude	36,000	bbl.	36,000	36,000	36,000	36,000	36,000	1,500	45,000
8	Demand	Regular gas	5,000	bbl.	5,000	5,000	5,000	5,000	5,600	5,800	6,500
9	Demand	Premium gas	8000.00	bbl.				8000	9,000	20,000	50,000
10	Quality	Min Cat in Reg	0.50		0.50	0.5	0.5	0.5	0.5	0.5	0.5
11	Quality	Min Cat in Prem	0.65		0.65	0.65	0.65	0.65	0.65	0.65	0.65
12	Cost	Non-Crude	$24.00		$27.00	$27.00	$27.00	$27.00	$28.00	$33.00	$40.00
13	Cost	Crude	$25.00	per bbl.	$28.00	$28.00	$28.00	$28.00	$35.74	$35.74	$38.00
14	Cost	Cracker	$5.00	per bbl.	$5.00	$5.00	$5.00	$5.00			
15	Price	Low-End	$28.00	per bbl.	$25.00	$25.00	$25.00	$25.00	$35.06	$34.87	$33.50
16	Price	Regular gas	$40.00	per bbl.	$40.00	$40.00	$40.00	$40.00	$58.77	$75.00	$100.00
17	Price	Premium gas	42.00	per bbl.				$40.00	$58.77	$75.00	$100.00
18	Price	Hi-End	$200.00	per bbl.	$200.00	$200.00	$200.00	$200.00	$210.00	$210.00	$2,000.00
19											
20	Profit	Annual net income	$31,988		$31,751						
21	Efficiency	Tower utilization	5%		100%						
22	Efficiency	Cracker utilization	100%		1						
23	Sales	Low-End	10,610		21,600						
24	Sales	Regular	21,600		580						
25	Sales	Premium	580		5,601						
26	Sales	Hi-End	10,000		4,244						

Figure 8. Multiple Versions Changes

The changes that are more recent are shown in a darker shade. Since column K was added most recently, it is darker shaded than column F, which was added first. When a row is removed the cells in the columns that are added later are black, signifying that no data exists for those cells. A similar process is done for the added rows. This functionality allows the system to report on the evolution tendencies of the spreadsheet. In Figure 8 one can notice that every year a new column has been added, and in occasional versions rows have been added or deleted. This is a very simple evolution of a spreadsheet, one that tends to be very linear. However, much more complex spreadsheet evolution patterns could conceivably be recognized and displayed.

These evolutionary patterns can help us accomplish two things; namely, detecting errors and identifying possible templates. If we already have the template used for a spreadsheet then we can compare the template to the changes that were made. If any change is incongruent with the template then this could be reported as an error.

If there is no template for a spreadsheet, the evolutionary pattern information can be used, along with header information to create one. This is particularly evident in Figure 8, where a column has been added in every new version. This shows that the template would be the first four columns, with a repeating block of one column.

7. Conclusion

As our results have shown SheetDiff is a robust tool that efficiently and effectively finds the changes between two spreadsheets. The ability to identify higher-level changes and the contextual embedding of changes results in more compact representations. This allows users to not only see the changes that have been made in an easy to understand representation, but also shows the steps one could take to transform one spreadsheet to the other.

SheetDiff gives end-users the ability to see the changes made between versions with the click of a button and determine if the new version has been changed correctly or if there are any unexpected changes. This makes SheetDiff a very useful tool in a business setting and it holds the potential to help facilitate the reuse and sharing of spreadsheets.

References

[1] Synkronizer, 2010. http://www.synkronizer.com/.

[2] R. Abraham and M. Erwig. Header and Unit Inference for Spreadsheets Through Spatial Analyses. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, pages 165–172, 2004.

[3] J. P. Caulkins, E. L. Morrison, and T. Weidemann. Spreadsheet errors and decision making: Evidence from field interviews. *Journal of Organizational and End User Computing*, 19:1–23, 2007.

[4] G. Engels and M. Erwig. ClassSheets: Automatic Generation of Spreadsheet Applications from Object-Oriented Specifications. In *20th IEEE/ACM Int. Conf. on Automated Software Engineering*, pages 124–133, 2005.

[5] M. Erwig. Software Engineering for Spreadsheets. *IEEE Software*, 29(5):25–30, 2009.

[6] M. Erwig, R. Abraham, S. Kollmansberger, and I. Cooperstein. Gencel: a program generator for correct spreadsheets. *J. Funct. Program.*, 16(3):293–325, 2006.

[7] M. Fisher and G. Rothermel. The euses spreadsheet corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. In *WEUSE I: Proceedings of the first workshop on End-user software engineering*, pages 1–5, New York, NY, USA, 2005. ACM.

[8] Florencesoft. DiffEngineX — Compare Excel Worksheets, 2010. http://www.florencesoft.com/index.html.

[9] K. Rajalingham, D. Chadwick, B. Knight, and D. Edwards. Quality control in spreadsheets: A software engineering-based approach to spreadsheet development. In *33rd Hawaii Int. Conf. on System Sciences-Volume 4*, pages 1–9, 2000.

[10] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 207–214. IEEE Computer Society, 2005.

[11] D. M. Volpano and R. B. Kieburtz. The templates approach to software reuse. *Software reusability: vol. 1, concepts and models*, pages 247–255, 1989.