Get Your Spreadsheets Under (Version) Control

José Nuno Macedo[13], Ricardo Moreira[24], Jácome Cunha[14], and João Saraiva[13]

[1] Universidade do Minho, Portugal
[2] Universidade NOVA de Lisboa, Portugal
[3] HASLab/INESC TEC, Portugal
[4] NOVA LINCS, Portugal
a72424@alunos.uminho.pt rm.moreira@campus.fct.unl.pt
{jacome,saraiva}@di.uminho.pt

Abstract. Spreadsheets play a pivotal role in many organizations. They serve to store and manipulate data or forecasting, and they are often used to help in the decision process, thus directly impacting the success, or not, of organizations.
As the research community already realized, spreadsheets tend to have the same problems "professional" software contain. Thus, in the past decade many software engineering techniques have been successfully proposed to aid spreadsheet developers and users. However, one of the most used mechanisms to manage software projects is still lacking in spreadsheets: a version control system. A version control system allows for collaborative development, while also allowing individual developers to explore different alternatives without compromising the main project.
In this paper we present a version control system, named SheetGit, oriented for end-user programmers. It allows to graphically visualize the history of versions (including branches), to switch between different versions just by pointing and clicking, and to visualize the differences between any two versions in an animated way. To validate our approach/tool we performed an empirical evaluation which shows evidence that SheetGit can aid users when compared to other tools.

Keywords: Spreadsheets · Version Control System

1 Introduction

Spreadsheet systems are a success example in the history of software systems. They have achieved an astonishing usage both in terms of the number of users (55 million of end-users programmers in 2012 just in the USA alone [13]) and the variety of domains in which they are nowadays used. Moreover, they are used both by professional programmers at large worldwide organizations, and by non-professional programmers in small family-run businesses. In fact, spreadsheet systems are installed in 90% of all computers globally [3] .

Spreadsheet systems offer end users a high level of flexibility, making it easier to get started working with them. Although spreadsheets start as a single user, simple software artifact, as any other software system, they tend to quickly evolve

into a large and complex system! Moreover, in the age of cloud-based software development, spreadsheets tend to be maintained in a collaborative environment.

In such a context, software is nowadays developed by relying on version control systems that help collaboratively building complex systems. A version control mechanism helps teams to manage changes to the same source code over time. These systems usually track and provide control over changes in software files and implicitly allow developers to return to any earlier stage of the software.

Although the spreadsheet research community has done considerable work, for instance, in defining abstractions for complex spreadsheets [7, 6, 4] or developing testing frameworks [1, 2], unfortunately, little work has been done in supporting version control based spreadsheet development. As a consequence, version control in spreadsheet development is rare, still! There are several reasons for this situation. First, spreadsheet users, usually non-professional programmers, are not familiar with systems for version control software. Second, spreadsheets are easy to share - usually a single file artifact which has to be under version control, thus making it difficult to control and maintain their integrity. Third, version control systems, such as Git[5], work very well for text-based languages, but they cannot be directly applied to the visual-representation of spreadsheets.

Although, version control systems are rarely used in spreadsheet development, research shows that version control does help end users [11]. Because the traditional version control systems cannot be directly applied to (visual) spreadsheets, end users perform their own versioning manually by adding a suffix/prefix to a file name with the file's version number! This can be seen even in large companies such as Enron [9], where they often sent updated spreadsheets through email. However, manual versioning holds numerous risks, as it is possible for a user to receive an email with a wrong or already outdated version, which would cause a problem whose root can be difficult to trace.

The goal of our work is three-fold: First, to bring version control to spreadsheet end-users developers in an intuitive manner. Second, to incorporate yet another modern software development methodology into the spreadsheet realm. Third, to help end users to create, manage, and comprehend complex spreadsheets. For this purpose, we created an add-in for Microsoft Excel, named SheetGit, which automatically commits changes, creates branches when needed, allows the user to change between versions, and see the spreadsheet differences between versions. Because, the control version is to be used by end users, our user interface has been carefully designed: an history slider is presented in our plug-in allowing end-users to return/see any earlier version of their spreadsheet. A new version of a spreadsheet is added to the control version by just pressing a button (or by configuring such a commit to occur when a given number of cells have been updated). This will be detailed in Section 3.2.

To evaluate SheetGit we performed an empirical evaluation with 18 participants, doing two different kinds of tasks in two different spreadsheets. Evidence show SheetGit helped end users being faster and more correct in most cases. Section 4.5 presents this study in detail.

[5] https://git-scm.com

2 Related work

In this paper we present a version control system, named SheetGit. The initial ideas that motivated the development of this tool were presented in a preliminary workshop paper [12]. We now present in great detail SheetGit, we extend the visual interface of the tool, and we present a validation of our approach and tool.

Microsoft Excel's official approach, which they simply call *History*, is available solely on the Windows platform. However, it is limited to spreadsheets hosted on Microsoft Sharepoint, so one cannot make use of it without being online, but it does allow collaboration between users. Versions are saved automatically when the user saves the document; Excel will either create a completely new version or merge the changes with the last one, with unknown criteria.

One of the important features in version control is the ability to see the differences between versions of a file, but Excel does not have such a feature.

It is worth noting that Microsoft has actually developed an official tool to detect spreadsheet differences called *Spreadsheet Compare*[6] that comes alongside some versions of Excel 2013 and 2016 in Windows. It works as an external tool and can function without having Excel itself open, having no integration with Excel's version control system. In *Spreadsheet Compare*, spreadsheets are displayed side-by-side without their formatting, and depending on the type of change, have specific colors highlight the altered cells. Unfortunately, there is no easy way to detect changes to entire rows and columns, as sometimes the program's algorithm merely sees them as regular changed cell values. While that is indeed correct, it is not very useful for users as it does not represent what happened in reality. When it does detect row/column changes, it adds them to the list of all changes, but does not specifically show that in the spreadsheets.

Coopy[7] is a spreadsheet version control tool which supports showing differences, patching, merging and conflict resolution on spreadsheets and database tables. It is separated from spreadsheet programs, and focuses on keeping data in synchronization with various people across multiple spreadsheet technologies by converting the worksheets to an intermediate format called CSVS. This CSVS format is based on the well known CSV format but with support for multiple worksheets per file, unambiguous header rows, and a clear representation of NULL. While Coopy does have a large feature-set, its interface is aimed more towards professional programmers with its use of concepts, information and syntax probably difficult for end users to know.

The revision history system of Google Sheets[8] is very powerful. It begins functioning immediately upon creating a new spreadsheet and automatically commits a new version whenever a new change is made. If various changes are done in a short period of time, Google Sheets may aggregate the changes into a single commit. In the Revision History page, one is able to see a list of all

[6] https://support.office.com/en-us/article/overview-of-spreadsheet-compare-13fafa61-62aa-451b-8674-242ce5f2c986

[7] http://share.find.coop

[8] https://www.google.com/sheets/about/

versions right next to the actual spreadsheet, each having the list of people who edited the spreadsheet in that particular version and their unique color. Upon clicking one of the versions, the spreadsheet will change to show that particular point in history but in a gray-scale color scheme; the cells that were edited in some fashion will be tinted with the unique color of their author.

XLTools[9] is a suite of various utilities for Microsoft Excel all in one VSTO add-in, one of them being called Version Control. This tool can create a local Git repository for the active workbook, where committing can be done either manually or automatically when saving; commit messages can also be added. The user also has access to the list of revisions, where they can choose to compare or save individual worksheets. However, there is no option to directly restore to a previous version. One has to save the file somewhere and then overwrite the original when Excel is closed, as the original cannot be overwritten while Excel and the add-on are open. Despite using Git, the system is fully local and has no collaboration features. Spreadsheet comparing can only be applied to one worksheet at a time. This will open a new Excel window with both old and new spreadsheets, having the new cell values tinted in red and its text in bold. The tool does not detect formatting changes and merely detects the actual cell value, and since formatting when comparing sheets is not changed, if one is unfortunate enough to have the same red cell formatting as the add-on uses, edited cells in the document may be confused with cells that were never touched.

Xltrail[10] is a recently created program that provides version control in spreadsheets. It is not directly integrated with Excel, it instead functions by creating a folder in the operating system that will be monitored by the program. Any spreadsheets placed in it become versioned, and when they are edited, the program will detect the changes and create a new version appropriately. It can also work through the cloud by detecting changes on Dropbox or Sharepoint accounts. In this case one would not need to download their personal client. Versions are kept in a linear format, much like Microsoft Excel. The service provides diffing, in a similar way to Coopy's highlighter format. Xltrail does not work without an internet connection nor does it support branching and merging.

Table 1 presents an overview of the features implemented by all tools.

Table 1. Table comparing SheetGit features with existing tools

Tool	Commit	Diff	Merge	Branch	Collaboration	Offline
SheetGit	yes	yes	yes	yes	yes	yes
Microsoft Excel	yes	no	no	no	yes	no
Spreadsheet Compare	no	yes	no	no	no	no
Coopy	no	yes	yes	no	no	yes
Google Sheet	yes	yes	no	no	yes	no
XLTools	yes	yes	no	no	no	yes
xltrail	yes	yes	no	no	yes	no

[9] https://xltools.net
[10] http://xltrail.com

3 The SheetGit Tool

In this section we describe several features for version control for spreadsheets and introduce SheetGit. We have created our solution as an Excel add-in to keep it as closely knit to Excel as possible, as this potentially increases its acceptance among spreadsheet developers as opposed to have to be run as an external tool. This will also enable us to present information directly in the active spreadsheet, and grant us access to Excel's proprietary file formats.

3.1 Technological Choices

There are two main types of add-ins for Excel: *i)* the *Visual Studio Tools for Office* (VSTO), add-ins made with C# or Visual Basic, and *ii)* the new type Microsoft calls *Office add-ins*. The latter are simply web pages that can interact with the documents using an API in Javascript. They are sandboxed and less closely integrated with Office, making them more restrictive than VSTO add-ins, but in exchange, they would also work in Excel for browsers.

We chose to create an VSTO add-in instead of using the new Javascript API, because we find the API not powerful enough for what we intend to create. We ultimately want the add-in to start immediately when Excel is ran so there is no risk of the users forgetting to enable the add-in, possibly losing data as a result, something that is not possible in the Javascript add-ins due to their sandboxed nature. As a result, our solution will only support Microsoft Excel in the Windows platforms.

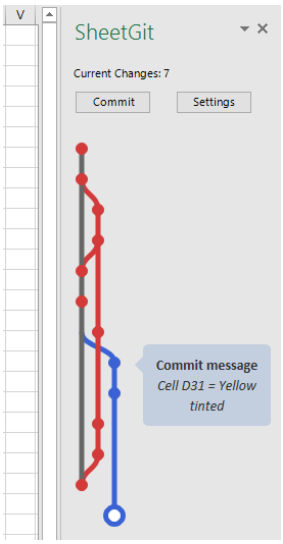


Fig. 1. SheetGit's task pane, displaying various commits

The add-in consists of a self-contained task pane, as seen in Figure 1, that includes an embedded browser to allow us to make use of Javascript, while retaining the advantages of VSTO add-ins. Our version tree is created using Gitgraph.js[11], a Javascript library. We use Bitbucket[12] in the background to actually manage the different versions of the spreadsheet, that is, each commit is actually stored in a private Bitbucket repository as a new file.

3.2 SheetGit Capabilities

We implemented a set of fundamental features: committing versions, switching between commits/versions, the option to see the actual differences between two versions, merge two different commits, and sharing these between various people.

[11] http://gitgraphjs.com
[12] https://bitbucket.org

Committing One of the most important features of a version control system is to organize the history of the commits done by the different users. In our particular case a *commit* is a set of changes in a workbook that occurred since the last commit until the next one. These commits can be done automatically (each time the user saves the file), or manually (when the user decides to commit the changes). A commit can also be called a *version* of the spreadsheet.

The list of commits, or versions, is displayed in a tree format, as shown in Figure 1. We use a tree because of its simplicity and adaptability, allowing for an intuitive graphical representation of many version control functions such as branches and merges. Very popular version control services such as Github[13] or Bitbucket also make use of a tree structure to display their version lists. The commits are placed chronologically, with the topmost being the oldest and the bottom most the youngest.

The large white commit is the currently active version, meaning it is the spreadsheet the user is seeing. A user can click in a desired commit in the tree, and the workbook will change so it is in the state of the particular version selected. In this case, the user restored his spreadsheet to a previous version, so it is not the most recent version of the spreadsheet.

By default, commits are generated automatically, reducing the burden on the user. By using metrics such as time since last edit and amount of data entered into the workbook, we combine various commits into blocks such that the end result may be indifferent to the user. In fact, Google Sheets has a similar behavior. An option to enable manual commits has also been added, for more experienced users who know better when to actually commit changes.

Each commit is associated with an optional commit message, auto-generated version number, and an automatically generated summary of the changes, which is shown upon hovering a commit dot in the tree as illustrated in the text box displayed in Figure 1.

Even though these messages are automatically generated, the user has always the possibility of editing such message by clicking and directly editing it. Next we list some of the messages shown for particular changes (the message should be self explanatory, c being a cell address, v a value, *col* a column letter, *row* a row number, and *color* a color): Cell $c = v$; Deleted cell c content; Deleted column *col*; Inserted row *row*; Cell c = *color* tinted.

Note that the formatting changes are also recorded and displayed. This is necessary in spreadsheets as they are naturally visual tools. This is a feature none of the professional version control systems hold.

Showing Differences Between Commits To make an easy to understand comparison in complex spreadsheets, SheetGit presents a slider between two consecutive commits. Dragging the slider from one commit to the other displays the individual differences (e.g. cell changes) between them, one at a time. This allows the user to see the differences step-by-step rather than being overwhelmed

[13] https://github.com

with all of them at the same time. This scales well with spreadsheet size and complexity as long as the user controls the speed of traversing the timeline. An option to skip to certain cells of the comparison also helps in this regard. The differences that are shown as the user moves through the timeline are also appropriately highlighted in the spreadsheet as one needs to be able to tell it apart from the rest of the spreadsheet. This can be done for example by changing the affected areas color or by drawing a circle around it.

In SheetGit, the Diff Pane, as shown in Figure 2, contains all functionality related to viewing differences between two versions.

The slider moves through each cell that has differences between the two versions. The two directional buttons move the slider up and down, one step at a time, for when more precision is required and because some users may favor clicking buttons rather than dragging the slider.

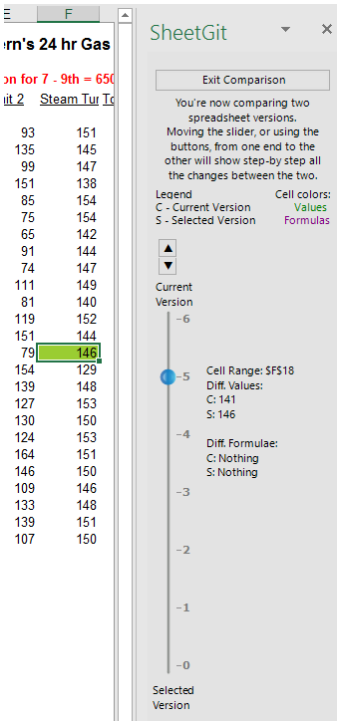


Fig. 2. Comparison between two versions in SheetGit

Collaborative Development Version control systems are a great incentive for collaborative work. Thus, the visualization of the commits must reflect the changes introduced by the different users. To do so, our tree has a different color for each user working in a particular spreadsheet.

Moreover, in order to organize the collaborative development, each user gets his own branch so such particular development can be done without interfering with the main development. Branches other than the main one are thus colored with the user's color, signaling the respective author's ownership. Thus, the commits are shown as colored dots in the tree, each color uniquely representing the commit's author. This can be seen in Figure 1, where two different colors signal that two different authors collaborated on the same spreadsheet.

Each user can have more than one branch. For instance, if the user begins to edit the spreadsheet from an older version (recall the last version is the bottom dot in the line), it will be created a new branch for the forthcoming commits.

Merging Since each user has its own branch, at some point she/he may desire to integrate her/his changes in the main branch. When done, this will be displayed in the tree structure in an analogous way to the new branch, as illustrated in Figure 1, where the red author performed merges twice over the development cycle. These merges are signaled by the red line joining the main branch.

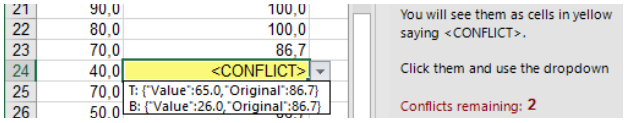


Fig. 3. Example of conflict when merging with SheetGit: When a new branch was created the original value of the conflicting cell was 86.7. Then, in the new branch (B) that cell changed to 26.0, while it was also changed in the main branch/trunk (T) to 65.0. Thus, a conflict occurs when merging the branches.

To make the development process easier for end user, we decided that it is not possible to merge changes between branches from different users. Nevertheless, it is possible to merge branches from the same user.

In some cases, conflicts may happen when merging to the main branch. In these cases two situations may occur. First, if there are concurrent changes to cells, the user must solve the conflicts by hand, that is, he/she must decide which version goes to the main branch. SheetGit will show the possible solutions, allowing the user to choose the intended one, as displayed in Figure 3. The second situation occurs when the changes are in different cells. In this case, all the cells will be integrated in the spreadsheet without the need for user intervention.

SheetGit is available at http://spreadsheetsunl.github.io/sheetgit.

4 Empirical Evaluation

The aim of our study is to evaluate the effectiveness and efficiency of users using SheetGit when compared to the use of Excel and Spreadsheet Compare.

4.1 Study Design

Hypotheses. With this study we intend to test two hypotheses:

1. In order to perform a given set of tasks, users spent less time when using SheetGit instead of using only Excel and Spreadsheet Compare.
2. Spreadsheets used with the support of SheetGit have a correctness grade higher than using only Excel and Spreadsheet Compare.

Formally, the two hypotheses being tested are: H_T for the time that is needed to perform a given set of tasks, and H_C for the correctness grade found in different types of spreadsheets. They are respectively formulated as follows:

1. **Null hypothesis**. H_{T_0}. The time to perform a given set of tasks using SheetGit is not less than that taken using Excel and Spreadsheet Compare. $H_{T_0} : \mu_d \leq 0$, where μ_d is the expected mean of the time differences.
   Alternative hypothesis. $H_{T_1} : \mu_d > 0$. The time to perform a given set of tasks using SheetGit is less than using Excel and Spreadsheet Compare.
   Measures needed. Time taken to perform the tasks.

2. **Null hypothesis**. H_{C_0}. The correctness grade in spreadsheets when using SheetGit is not smaller than using Excel and Spreadsheet Compare. H_{C_0} : $\mu_d \leq 0$, where μ_d is the mean difference of the correctness grades.
   Alternative hypothesis. $H_{C_1} : \mu_d > 0$. The correctness grade when using SheetGit is smaller than when using only Excel and Spreadsheet Compare.
   Measures needed. Correctness grade for each spreadsheet.

The independent variables are, for H_T, the time to perform the tasks (efficiency), and for H_C, the correctness grades (effectiveness).

Subjects and Objects. The study was performed with freshmen computer science students from the NOVA University of Lisbon. They were chosen because they are likely to have minimal experience with Excel and not enough programming experience to know about version control. To provide incentive for participation, we decided to raffle a voucher with the value of fifty Euro for a retail store. From the 18 subjects, 16 were male and 2 were female. All of the subjects were below 20 years old. 62% of the subjects had experience with Excel and only 25% have had experience with version control.

The objects of this study are three spreadsheets. The spreadsheet used as a tutorial, termed Southpoint, calculates the total gas usage when given input values. It was used to explain how to use SheetGit to half of the participants, and Spreadsheet Compare to the other half. The second spreadsheet, termed Grades, manages and calculates grades for students of an university course. The third spreadsheet, termed Markets, calculates the income of Enron's global markets. The spreadsheets were retrieved from two widely used spreadsheet repositories: EUSES [8] and VEnron [5].

Instrumentation. After the tutorial phase, each participant received two spreadsheets: the Grades and the Markets. For each spreadsheet, the participants received two tasks. The first task asks users to correct an error in a spreadsheet that sprouted in a recent version, but was correct some time ago. So users would have to *diff* the multiple versions to find out where the error occurred, and then correct it in a new version. The second task asks users to unite two versions of a spreadsheet that have diverged from a common ancestor. Users would then have to compare the three spreadsheets (the ancestor and the two different children) and then perform a three-way merge. During the study a pre-questionnaire a post-questionnaire were given to participants.

Participants were randomly split between two groups as to have an equal amount of people with and without SheetGit. Some participants started with the Markets spreadsheet and others with the Grades spreadsheet. This is important as during the tasks performed with their first spreadsheet, participants are still learning and will gradually improve, so when they reach the second spreadsheet, they will have more experience than they had at the start. Concentration levels also begin to decrease over a period of time, which could influence the time spent on each task.

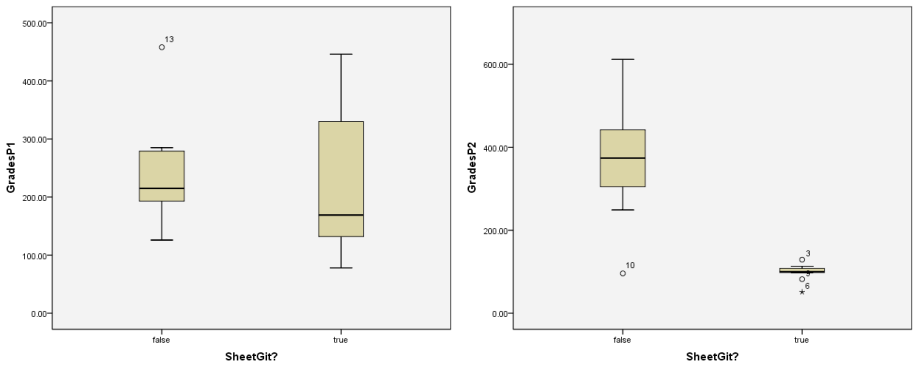


Fig. 4. Box plots for the Grades tasks time (first/second task on the left/right chart).

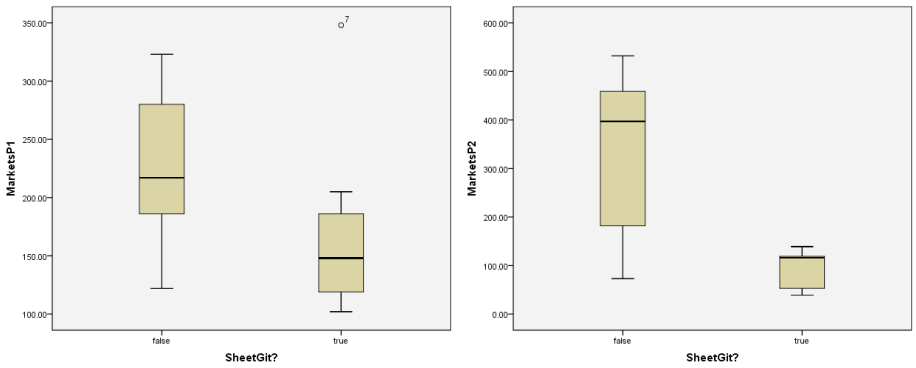


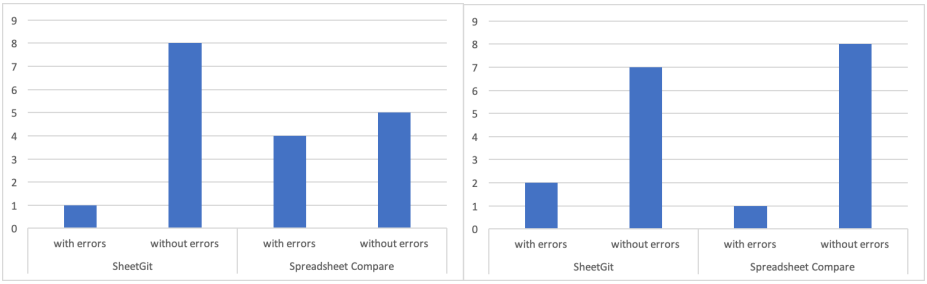
Fig. 5. Box plots for the Markets tasks time (first/second task on the left/right chart).

4.2 Results Analysis

Time spent (efficiency). There are noticeable differences in the time the participants used to perform the study. Figure 4 shows the time it took for the participants to perform Grades spreadsheet tasks. The Y axis contains the time in seconds and the X axis defines if SheetGit (box with label *true*) or Spreadsheet Compare was used (box with label *false*).

Figure 5 presents the times for the Markets spreadsheet.

Correctness grade (effectiveness). In regards to correctness, we divided the type of errors participants could perform into two categories: having incorrect values inputted in the correct version and inputting the correct values in a wrong version. As such, the bar chart in Figure 6a shows the number of participants who committed the former error, while Figure 6b the latter. The charts show all the spreadsheets' results together.



(a) Wrong values in the right version. (b) Correct values in wrong version.

Fig. 6. Errors produced by the participants.

Hypothesis Testing. To test our hypothesis on efficiency, we ran a Welch unequal variances t-test to determine if there is any statistical significance [10].

For Grades task 1, $p = 0.682$, task 2, $p = 0.001$, and for Markets task 1, $p = 0.116$, task 2, $p = 0.004$.

We also calculated Cohen's d to determine our effect size [10].

For Grades task 1 the results if 0.196974, task 2 2.499612, and for Markets task 1 0.784395, task 2 1.819253.

Comparison of times. From the t-test results we can deduce that only task 2 has statistical significance for both spreadsheets. For the task 1, there was no statistical significance within the study.

Through Cohen's d we can see that in task 2 for both spreadsheets the difference between the two means can be classified as very large (> 0.8). So SheetGit in these tasks helped the participants in a scale of more than one standard deviation, which is very impressive.

Comparison of correctness. A couple of different tests were conducted, such as the Pearson Chi-Square and Fisher's Exact Test but no statistical relevance could be found, mostly due to the low count of errors in both cases.

4.3 Results Interpretation

The results from the analysis suggest that SheetGit does improve user performance while performing these tasks. The second task, related to merging in version control, had strong statistical relevance, being noticeably superior over not using SheetGit. This is likely because SheetGit actually introduces a new method that directly aids users in the merging process. An example would be how it pinpoints conflicting cells while the counterpart users would have to search for them manually. SheetGit also automates parts of the merging process when possible to perform decisions without user input, which helps greatly lower the time, difficulty and possibility of errors within the task.

Regarding the first task, related to diffing and correcting errors, there was no statistical significance found, though the average time required to solve the task was inferior for SheetGit users. This is likely because while SheetGit allows one to move between versions and diff without changing windows, it is not that much faster than performing a diff with Spreadsheet Compare. Even if the interface proves to be simpler, both sides of the participants received tutorials for their tools, so provided they understood the tool, it would be likely for the difference to be small. The results can also be attributed to the fact that there were few versions to compare, which can provide an edge to Spreadsheet Compare, which displays all differences between two versions instantaneously. SheetGit instead shows the changes one by one, though it can group changes from multiple versions in a single diff. In this scenario, SheetGit would likely be even faster because those without it would have to navigate through menus several times to change the versions to compare.

While no statistical relevance was obtained from analyzing the correctness of the tasks, SheetGit had less errors in terms of wrong values overall. This may be because of the unified interface, all inside Excel, which keeps the users focused and can lead to less human error. The lower average time when performing the tasks would also help in terms of focus. It is interesting to note that SheetGit did indeed have more errors when it comes to users inputting the correct values, but in wrong versions. What this means is that they corrected what error they had to find, but in an old version. So the new resulting version did not have any of the changes that occurred between that old version and the latest. All of these errors occurred in the exact version where the error had, which means the users just did not return to the latest version to correct it there.

4.4 Threats to validity

A few threats to validity deserve being analyzed.

Regarding internal validity, in order to minimize any effects on the independent variables that would reflect on the causality, several actions were taken. First, half the participants started with the Markets spreadsheet, and the other half with Grades. This would minimize any learning effects from happening throughout the session. Second, the study was intentionally short in order to prevent the participants from losing focus while performing their tasks. Third, the study was performed over two sessions, one in which half the participants used SheetGit and the other where the latter half did not. Fourth, all participants executed the exact same tasks, so no group received special treatment.

For conclusion validity, a concern is the low amount of participants, which leads to a lower statistical power for the study. When calculating the correctness grade, we grouped the tasks' errors together to increase the statistical power.

As for construct validity, participants were informed beforehand that they were not under any sort of evaluation to guarantee they would not be affected by the study itself. The tasks we asked the participants to perform are common issues that are solved by the use of version control, either with or without our tool, such as merging and diffing spreadsheets. By choosing these sort of tasks,

our study construct can evaluate whether or not users are more effective and efficient while using SheetGit.

Finally, regarding external validity, related to the strength to generalize the results of this study to industrial practice, we have selected two spreadsheets from the real-world: one from an actual company and another from the EUSES corpus, which in turn retrieved it from a Google search as part of a real-world example. Although the spreadsheets are real-world spreadsheets, the environment is not. Nevertheless, the participants represent a wide range of spreadsheet users, and thus, we believe that results are generalizable.

4.5 Discussion

The empirical study we conducted reveals promising results for SheetGit. Most participants wrote on the post-questionnaire that SheetGit helped them greatly in performing their tasks and that they thought it was a necessary tool.

Despite that participants had a short amount of time to learn a completely new perspective on managing backups and versions with our add-in, they accomplished their tasks on average faster than those that did not use SheetGit. Even if the first task did not achieve statistical significance, the users did in fact finish on average faster than those without SheetGit, which is impressive if one considers that they had to learn a new interface and perspective on Excel. That said, it could be faster by, for example, giving ahead of time a small highlight to every cell that would be changed. This way, users have a much better notion of the version in its entirety and the train of thought behind the changes.

Regarding errors, most found were related to users correcting errors on versions that were not the latest. This may be due to a lack of understanding or just an honest mistake due to the seamless nature of the interface, as this sort of situation happened even with users that finished both tasks fairly fast and otherwise correctly. A warning could be shown in case changes are attempted on versions that are not the latest to prevent this sort of error. The version tree could also be better labeled, much like SheetGit's diff tab, which has a detailed explanation of its functionality and appearance directly on the interface.

5 Conclusion

Spreadsheets are the most used programming environment in the world. However, they still lack many of the advanced features that modern programming environments offer, and in particular lack a proper version control system. We chose to alleviate this problem by bringing SheetGit to Excel. SheetGit functions as an integrated add-in for Excel and aids users by providing various functionalities of version control, such as automated version creation, collaboration, version comparison, and uniting two versions together in one spreadsheet. We offer all these features directly in Excel in a graphical and intuitive manner.

The empirical validation we performed showed that SheetGit does improve the users' efficiency when performing some tasks, while receiving praise from the participants for its concept, ease of use and necessity in the spreadsheet world.

Version control systems still have some features which were not included in SheetGit such as cherry picking, rebase and many others. But careful consideration must be put into these more advanced control versions features as they must be abstracted and adapted to spreadsheets and their end-user developers. Otherwise the user interface will just become more complex which is against the original purpose of the application. As future work we plan to study how such advanced features can be incorporated to SheetGit.

References

1. Abraham, R., Erwig, M.: Autotest: A tool for automatic test case generation in spreadsheets. In: Proceedings of the 2006 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '06). pp. 43–50. IEEE CS (2006)
2. Abraham, R., Erwig, M.: Mutation operators for spreadsheets. IEEE Trans. Software Eng **35**(1), 94–108 (2009), http://dx.doi.org/10.1109/TSE.2008.73
3. Bradley, L., McDaid, K.: Using bayesian statistical methods to determine the level of error in large spreadsheets. In: 31st International Conference on Software Engineering. pp. 351–354. IEEE (2009)
4. Cunha, J., Fernandes, J.P., Mendes, J., Saraiva, J.: Embedding, evolution, and validation of model-driven spreadsheets. IEEE Transactions on Software Engineering **41**(3), 241–263 (March 2015). https://doi.org/10.1109/TSE.2014.2361141
5. Dou, W., Xu, L., Cheung, S., Gao, C., Wei, J., Huang, T.: Venron: A versioned spreadsheet corpus and related evolution analysis. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion. pp. 162–171 (2016)
6. Engels, G., Erwig, M.: Classsheets: Automatic generation of spreadsheet applications from object-oriented specifications. In: Procs. of the 20th IEEE/ACM Int. Conf. on Automated Software Engineering. pp. 124–133. ACM (2005)
7. Erwig, M., Burnett, M.: Adding apples and oranges. 4th Int. Symp. on Practical Aspects of Declarative Languages pp. 173–191 (2002)
8. Fisher, M., Rothermel, G.: The euses spreadsheet corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. SIGSOFT Softw. Eng. Notes **30**(4), 1–5 (2005)
9. Hermans, F., Murphy-Hill, E.: Enron's spreadsheets and related emails: A dataset and analysis. In: Proceedings of the 37th International Conference on Software Engineering. pp. 7–16. ICSE '15, IEEE Press, Piscataway, NJ, USA (2015)
10. Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., Gibbs, S., Pohthong, A.: Robust statistical methods for empirical software engineering. Empirical Software Engineering **22**(2), 579–630 (Apr 2017)
11. Kuttal, S.K., Sarma, A., Rothermel, G.: On the benefits of providing versioning support for end users: An empirical study. ACM Trans. Comput.-Hum. Interact. **21**(2), 9:1–9:43 (Feb 2014). https://doi.org/10.1145/2560016
12. Moreira, R.: Sheetgit: A tool for collaborative spreadsheet development. In: Milazzo, P., Varró, D., Wimmer, M. (eds.) Software Technologies: Applications and Foundations. pp. 415–420. Springer International Publishing, Cham (2016)
13. Scaffidi, C., Shaw, M., Myers, B.: Estimating the numbers of end users and end user programmers. In: Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on. pp. 207–214 (Sept 2005). https://doi.org/10.1109/VLHCC.2005.34