



Ricardo Magalhães Martins Korn Moreira

BSc in Computer Science

SheetGit: A Tool for Collaborative Spreadsheet Development

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador: Jácome Cunha, Professor Auxiliar,
Universidade Nova de Lisboa

Júri

Presidente: Prof. Dra. Carmen Morgado
Arguente: Prof. Dr. João Saraiva
Vogal: Prof. Dr. Jácome Cunha



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

December, 2016

SheetGit: A Tool for Collaborative Spreadsheet Development

Copyright © Ricardo Magalhães Martins Korn Moreira, Faculdade de Ciências e Tecnologia, Universidade NOVA de Lisboa.

A Faculdade de Ciências e Tecnologia e a Universidade NOVA de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2]. [1] <https://github.com/joaomlorenco/unlthesis> [2] <http://www.di.fct.unl.pt>

Para a minha família.

ACKNOWLEDGEMENTS

I would like to thank my adviser Jácome Cunha for his support, guidance and patience in relation to both my research and myself. I would also like to extend my gratitude to my university, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, for giving me the opportunity to conduct my Master's Degree and this research.

I would also like to thank NOVA LINCS for financing my trip to Vienna, to present my paper at a workshop and for the opportunity to meet many wonderful people engaged in the world of spreadsheets, whose advice definitely helped improving my work.

I am also thankful to all my friends for the support, encouragement and their great friendship when it was needed the most.

Last but not least, a special thanks for my family for supporting me not only through this thesis but also my entire education and life.

I also want to thank everyone not listed here who contributed to this journey. Even if your names weren't mentioned, you were all parts of the building blocks that in the end created this thesis.

ABSTRACT

Modern development environments include several tools such as debuggers, testing frameworks, or version control. Indeed, version control has become an essential tool for assisting programmers in managing their source code. It helps them keep backups of their own work while providing an accessible history of everything they have done.

End users with no programming experience can also create complex software in the form of spreadsheets, both for personal and business use. In the case of business use, such spreadsheets may reach enormous levels of complexity and size. Complex spreadsheets can be very hard to create, debug and understand, problems that can be alleviated by having a proper version control system.

With this work we present an approach to bring the well-known version control benefits to spreadsheets in a simpler, easy to use and understand manner in order to appeal to end users. Our tool provides an intuitive graphical interface integrated in Excel with numerous features such as automatic version creation, viewing differences between two versions, uniting two versions together as one, and collaboration with other users. An empirical study was also performed on the tool, in the end showing that it is indeed more efficient and effective to use it when performing certain tasks compared to standard Microsoft Excel.

Keywords: Spreadsheets, End users, Microsoft Excel, Version Control, Excel Add-in

RESUMO

Os ambientes de desenvolvimento modernos incluem várias ferramentas como *debuggers*, estruturas de testes, ou controlo de versões. De fato, o controlo de versões tem-se tornado uma ferramenta essencial para ajudar os programadores na gestão do seu código fonte. O controlo de versões ajuda os utilizadores a manter cópias de segurança dos seus trabalhos e ao mesmo tempo fornece uma história acessível de todo o trabalho que foi efetuado. Os programadores não profissionais também podem criar *software* complexo, na forma de folhas de cálculo, tanto para fins pessoais como profissionais. No caso de fins profissionais, as folhas de cálculo podem atingir dimensões e níveis de complexidade muito elevadas. Folhas de cálculo complexas podem ser muito difíceis de desenvolver, corrigir e compreender, problemas que podem ser aliviados através de um sistema de controlo de versões.

Este trabalho apresenta uma forma de trazer os benefícios bem conhecidos do controlo de versões para as folhas de cálculo num formato simples, de utilização e compreensão fácil, de forma a agradar os programadores não profissionais. A ferramenta fornece uma interface gráfica intuitiva integrada com o Microsoft Excel, que permite criar versões automaticamente, visualizar as diferenças entre as versões, unir duas versões numa só e colaborar com outros utilizadores. Foi feito um estudo empírico sobre a ferramenta, mostrando que é mais eficaz e eficiente utiliza-la na execução de certas tarefas comparado com o Excel por si só.

Palavras-chave: Folhas de cálculo, Programadores não profissionais, Microsoft Excel, Controlo de versões, Excel Add-in

CONTENTS

List of Figures	xv
1 Introduction	1
1.1 Motivation	2
1.2 Challenges	3
1.3 Approach	4
1.4 Contributions	4
1.5 Thesis structure	4
2 State of the Art	7
2.1 Microsoft Excel	7
2.2 Coopy	8
2.3 Google Sheets	10
2.4 XLTools	11
2.5 Pathio	12
3 Version Control for Spreadsheets	15
3.1 Version listing	15
3.2 Creating versions	16
3.3 Moving between versions	17
3.4 Creating branches	17
3.5 Version messages and tags	18
3.6 Diffing	18
3.7 Collaboration	19
3.8 Conflict resolution	20
4 SheetGit: A Version Control System for Spreadsheets	23
4.1 Overview	23
4.2 Backend	24
4.3 Ribbon tab	25
4.4 Main Pane	25
4.5 Settings Pane	26
4.6 Diff Pane	28

CONTENTS

4.7	Conflicts Pane	30
4.8	Availability	30
5	Empirical Validation	33
5.1	Design	33
5.1.1	Hypothesis	34
5.1.2	Variables	34
5.1.3	Subjects and Objects	34
5.1.4	Instrumentation	35
5.1.5	Data Collecting Procedure	36
5.1.6	Analysis Procedure and Evaluation of Validity	36
5.2	Execution	37
5.3	Analysis	37
5.3.1	Descriptive Statistics	38
5.3.2	Hypothesis Testing	43
5.4	Interpretation	44
5.4.1	Threats to validity	44
5.4.2	Inferences	45
5.5	Discussion	46
6	Conclusions	47
6.1	Concluding Observations	47
6.2	Future Work	48
	Bibliography	49
A	Pre-Questionnaire	53
B	Tutorial for Spreadsheet Compare	57
C	Tasks for Spreadsheet Compare	67
D	Tutorial for SheetGit	71
E	Tasks for SheetGit	77
F	Post-Questionnaire	81
G	Version list JSON schema	85

LIST OF FIGURES

2.1	An example of the spreadsheet history tab as seen in Microsoft Excel 2016 for Windows.	8
2.2	Spreadsheet Compare 2016 comparing two versions of a spreadsheet from the Enron corpus.	9
2.3	Coopy's graphical user interface as seen in Linux Mint	9
2.4	An example of Coopy's highlighter diff format [8].	10
2.5	A list of versions in a Google Sheets document	11
2.6	A list of spreadsheet versions in XLTools	12
2.7	An example of Pathio's diffing [31].	13
3.1	A representation of a simple version tree	16
3.2	A version tree with a branch	16
3.3	A tree where the visible, current spreadsheet is its third version	17
3.4	Version tree displaying a version message and a version tag	18
3.5	Timeline between two versions	19
3.6	Resolving a conflict in a cell	21
3.7	A version tree in SheetGit after a merge	21
4.1	SheetGit in Excel 2016	24
4.2	SheetGit's Ribbon Tab	25
4.3	SheetGit's main task pane display	26
4.4	SheetGit's main pane in Comparison Mode	27
4.5	SheetGit's Settings Pane	28
4.6	SheetGit's Diff Pane	29
4.7	SheetGit during a diffing procedure	29
4.8	Resolving conflicts in SheetGit	31
5.1	Box plot for the time elapsed in executing Grades's first task	38
5.2	Box plot for the time elapsed in executing Grades's second task	39
5.3	Box plot for the time elapsed in executing Markets's first task	40
5.4	Box plot for the time elapsed in executing Markets's second task	41
5.5	Bar chart with the amount of participants who inputted wrong values in the right version	42

5.6 Bar chart with the amount of participants who inputted correct values in wrong versions 42

INTRODUCTION

Spreadsheets have become an indispensable necessity in many peoples' lives. Their widespread use can be attributed to the fact that they are extremely flexible, easy to create, fast to use and hold countless functions that can be used in various different businesses [38]. Microsoft Excel and other spreadsheet programs hold an install base of over 90% of all computers globally [2], and for 2012, it was estimated that there were 55 million of end users using databases or spreadsheets just in the United States alone [32]. The problem is that, while spreadsheets hold immense potential power, they are extraordinarily prone to error [29].

There have been multiple studies attempting to measure errors in spreadsheets, and they have always found them in abundance [29]. By default, spreadsheets are easy to share and modify, which makes it incredibly difficult to control and maintain their integrity [4]. The amount of user controls in spreadsheets do not approach the level of controls that professional programmers have found to be necessary in a similar application, so errors are more likely to happen and not be detected.

Another point worth mentioning is that due to the lack of development and design standards in spreadsheets, it is normal for spreadsheets to reach incredible levels of complexity and size, making them hard to comprehend and debug, ergo increasing the number of errors when they are used [18][2].

By adding new functionalities and controls that would help both detect and prevent errors from happening, one of the major advantages of spreadsheets, their simplicity and agility, would begin to wane. It is, however, becoming more of a necessity as spreadsheet usage continues to grow and as both businesses and end users come to rely on them to perform crucial tasks that could define their future. As a result, Excel has in fact been at the center of numerous financial crises [12].

So essentially, spreadsheets are being used as cheaper, more agile replacements of professional programs that would normally cost large sums of money to create and manage, but they have few to no controls and tools to prevent errors. If professional programmers are no longer making sure end users will not make mistakes, to prevent these from happening the end users must themselves start to adopt the disciplines and tools that programmers have long used when dealing with complex software [30], one of which being version control.

Version control is known to be beneficial for experienced programmers [26], but it has also been proven that it would be beneficial for end users both for learning and debugging purposes [25]. This is due to them generally learning from existing examples and how they tend to "debug their programs into existence", that is, they search for various alternatives to a solution, and backtrack their changes when required [25]. Version control also helps in understanding spreadsheets as these reach high levels of complexity. With proper version control, one can see how a spreadsheet was built over time and, from its origin, gradually come to understand it.

End users in fact already perform their own versioning manually by, for example, adding a suffix to a filename with the file's version number. This can even be seen in a large company such as Enron inside their corpus [17], where they commonly send 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 very difficult to find. In turn, experienced programmers have sophisticated tools to create and manage their versions such as Git [13] and Subversion [35], that help prevent these issues. Modern development tools such as these could be applied to spreadsheets, but they lack native integration and it would be very inconvenient for end users to have to learn the syntax and concepts behind something akin to Git. One of the big allures of spreadsheets is their simplicity and agility, so tools that require a lot of time to learn are likely to be discarded, but that does not mean it is impossible for many of their features to be available. These features just need to be presented in the right way, which is what we intend to do in this work.

1.1 Motivation

Assume that you're working on an Excel spreadsheet related to a company's finances. To avoid any problems with missing information, it's normal to keep versions of the spreadsheet as it is being worked on, this will prevent data loss and will remain as a fruitful backup if needed in the future.

Now, a second person could be working on the same spreadsheet as you, performing changes concurrently. In Version Control, versions are generally organized in the shape of a tree that grows with each version. The coworker started working based on your spreadsheet, but it has now branched off into creating its own different versions, this would be called a different branch in version control, in the tree.

When dealing with multiple versions of the same spreadsheet, it's important to be able to see the differences between said versions. In Version Control, this is called diffing, though the term is usually applied to text files.

In this case, eventually your coworker will want to join his part of the work with yours in one single spreadsheet. In Version Control, one of the methods of doing this would be by merging. The tree branch would connect back to the trunk where it sprouted from, with all the changes now integrated into it. Of course, if the two users edited at some point the same cells, this would be a conflict that would have to be resolved when performing this merge.

Microsoft Excel has no way of performing these actions. It does maintain backups, you could consider them as versions, but you're unable to branch off previous versions and keep your backups organized at the same time. Collaborative work is limited to working simultaneously on Excel Online in an internet browser, with your spreadsheet saved on Microsoft's servers. Diffing is only possible through Spreadsheet Compare, an application bundled with Microsoft Office, but is completely independent and separate from Excel.

Aside from helping manage versions and streamlining collaboration, version control also helps solve other problems with spreadsheet integrity. The user is always aware of what is the latest version of the spreadsheet, and the relationship between all the versions. So problems of data loss occurring due to a coworker mistaking an old version for the latest will not happen. Even if it did, version control's merging feature would casually update it with the newest changes. If an error shows up, it is always traceable down to who did it and when it happened, so the consequences that may happen over time can be ascertained.

Thus we chose to make this application to unite all of these features into one easy to use application, bringing version control's full benefits to spreadsheet end users.

1.2 Challenges

The main challenge of this work is the user interface. Version control is very complex and has numerous features. In order for spreadsheet end users to use our tool, it must be very intuitive, require a minimal tutorial and be very unobtrusive, otherwise they will not use it for fear of breaking their agile and flexible workflow. They do not have the same training professional programmers do.

Another big challenge is abstraction. It is also closely related to user interface. Not only is it the manner of how we abstract the complicated version control tools into simpler ones, but also how we adapt these tools to spreadsheets, when they were not designed to.

1.3 Approach

Our goal is to bring version control to spreadsheet end users, in an intuitive manner, to help modernize Excel's development tools, to help lower the risk of spreadsheet errors and to help end users create, manage and comprehend complex spreadsheets. For this purpose, we chose to design a version control system for spreadsheets, this including all of its features, chosen because they could be adapted to spreadsheets, and presentation, because end-user developers who have never programmed or and have never version control would need to understand them. All of this is detailed in Section ??.

We then created an add-in, based on our design, for Microsoft Excel because of its high install base in computers, and its undefeated market share [24]. The add-in automatically creates versions, makes branches when needed, allows the user to change between versions and see the difference between them, collaborate with colleagues and unite their changes into another spreadsheet with a carefully designed user interface.

1.4 Contributions

This work makes the following contributions:

- A methodology to create a version control system tailored for spreadsheets, including:
 - A user interface to interact with version trees and their children.
 - A methodology to allow online collaboration on a spreadsheet
 - A process for automatically creating versions and changing between them.
 - An approach to show differences between two spreadsheets.
 - A methodology to present and resolve conflicts in spreadsheets.
- An open-source add-in, termed SheetGit [33], that implements all of the mentioned items.
- We have also published a paper presented at an international spreadsheet workshop - "SheetGit: A Tool for Collaborative Spreadsheet Development" in *Software Engineering Methods in Spreadsheets 2016* [28]

1.5 Thesis structure

The rest of the thesis is structured as follows:

- In Chapter 2 we present the state of the art, the current existing solutions to our problem and how they differ from the one we created.

- In Chapter 3 we present the concepts and methodologies revolving around adapting a version control system to both spreadsheets and end-user developers.
- Chapter 4 explains how SheetGit functions and presents several details about its implementation.
- In Chapter 5 presents our empirical study on SheetGit.
- In Chapter 6 concludes the developed work and discusses future work.

STATE OF THE ART

It is no surprise that solutions already exist for version control in the context of spreadsheets, but we find them to be less powerful and intuitive than they ultimately could be.

In this chapter we discuss these other approaches to version control, comparing them to our own solution.

2.1 Microsoft Excel

Microsoft Excel's [27] 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. 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. Excel does not have such a feature; the user is given a list of versions, as seen in Figure 2.1, and upon clicking a version, he/she can see the spreadsheet as it was at that time, and is given the option to either revert to that version or abort.

It is worth noting that Microsoft has actually developed an official tool to detect spreadsheet differences called *Spreadsheet Compare* [34] 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*, as seen in Figure 2.2, the spreadsheets are put together, side by side, with their formatting stripped and, depending on the type of change, instead having specific colors highlight the altered cells. Additionally, there's one button to show

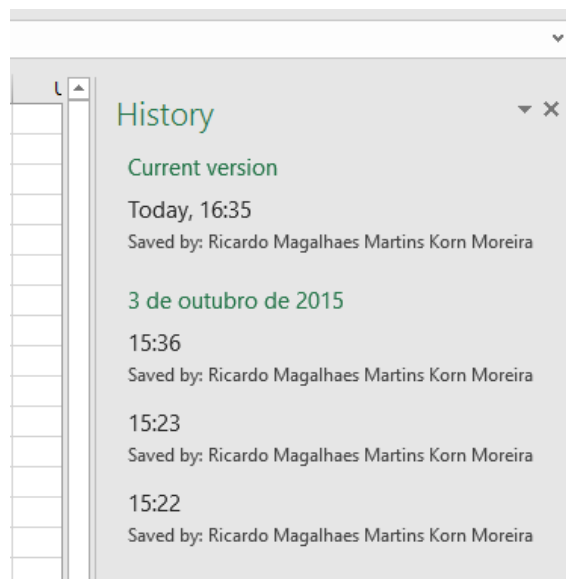


Figure 2.1: An example of the spreadsheet history tab as seen in Microsoft Excel 2016 for Windows.

cell formulas in place of their resulting values and another one to show the spreadsheet's real formatting. Unfortunately, there's 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, and while that is indeed correct, it's not very useful for human users as it doesn't represent what happened in reality. When it does detect row and column changes, it only notes them down in the list of all changes and doesn't specifically show that in the spreadsheet.

2.2 Coopy

Coopy [9] is a spreadsheet version control tool which supports diffing, patching (applying a diff file's changes to another spreadsheet), merging and conflict resolution on spreadsheets and database tables. It is separate from spreadsheet programs, and focuses on keeping data in synchronization across multiple spreadsheet technologies by converting the sheets to an intermediate format called CSVS. The meaning of the acronym is not officially known, but the last "S" is likely to form the plural of the term CSV (Comma-separated values), due to the fact that each CSVS file is essentially multiple concatenated CSV files.

This CSVS format is based on the well known CSV format but with support for multiple sheets per file, unambiguous header rows, and a clear representation of NULL. There is currently no known software other than Coopy that supports CSVS files.

While Coopy does have a large feature-set, its interface is aimed more towards professional programmers with its use of concepts, information and syntax that end users normally would not understand (e.g. *diffing*, *branch*, *merge*) as can be seen in Figure 2.3.

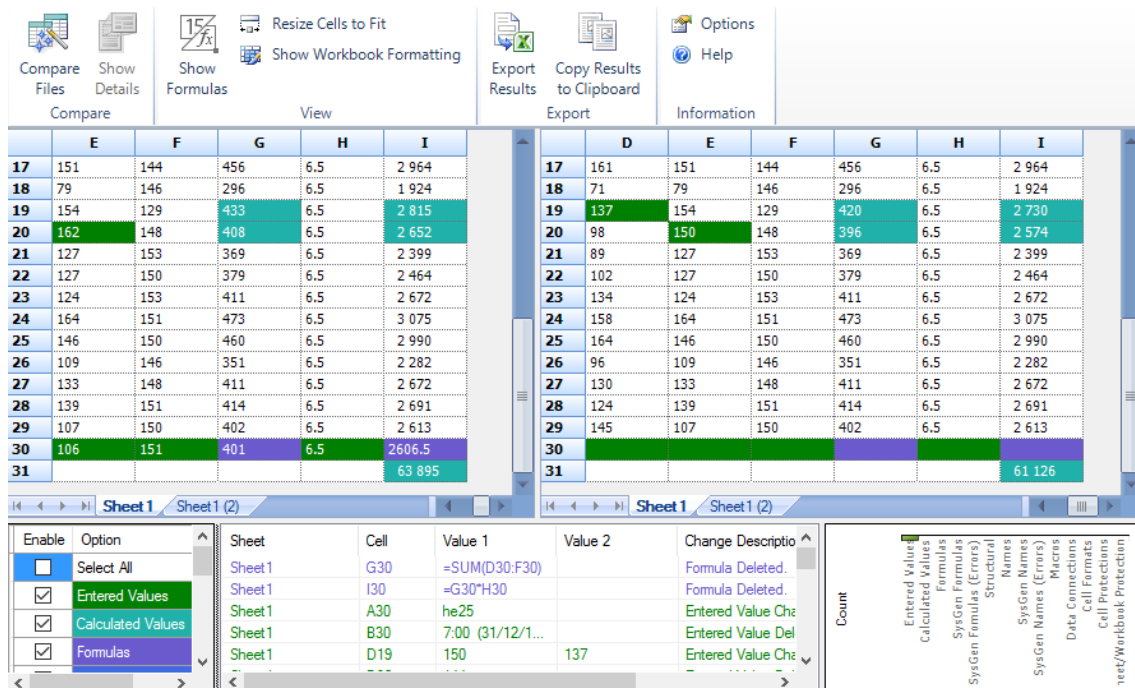


Figure 2.2: Spreadsheet Compare 2016 comparing two versions of a spreadsheet from the Enron corpus.

Coopy also does not support the more popular .XLSX Excel format; instead only supports the older XLS through *Gnumeric* and its *libspreadsheet* library.

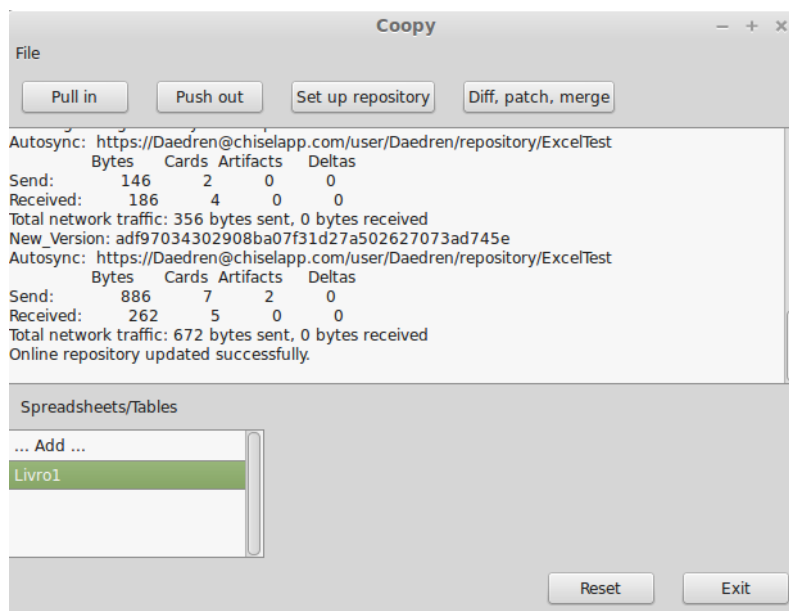


Figure 2.3: Coopy's graphical user interface as seen in Linux Mint

Coopy has two main methods to present spreadsheet differences[10][7]: i) the tDiff text format is meant to retain the aesthetics of regular diff files, and is intended to not

only be parsed by programs, but also for end users to be able to read the differences [11]. But it can be difficult for a user with no experience to read it, and not only that but the diff's complexity can grow when dealing with large spreadsheets. What we are aiming with our solution is a method that can be read by anyone who understands a spreadsheet, with at most a minimal tutorial, while also being a method that can be applied to other types of spreadsheet content, such as the formatting, something tDiff cannot do [11].

ii) The highlighter diff format. This format is meant to be presented in an actual spreadsheet yet it can also be parsed for analysis and patching. However much like tDiff, it deals solely with cell content and no sort of cell formatting or formulae [8]. In Figure 2.4 we can see that two tables have merged together, with colors marking what cells were newly added, removed or changed. There is also a new control row and column, situated at the top row and the leftmost column, which identify the type of changes that happened in them. The markers in these control sections (e.g. '+++', '+') can potentially be hard for users with no experience with diffs to understand.

Having this information condensed in one single table, makes it easier to evaluate larger spreadsheets in one glance, unlike Spreadsheet Compare's approach, which requires one to look at various places at once. However, the spreadsheet may become very disorganized and complex if the number of changes between versions is high.

!			+++	
@@	bridge	designer	quark	length
+	Brooklyn	J. A. Roebling	strange	1595
+++	Manhattan	C. Lindenthal	charm	1470
->	Williamsburg	D. Duck->L. L. Buck	up	1600
+	Queensborough	Palmer & Hornbostel	down	1182
+	Triborough	O. H. Ammann	strange	1380,383
+	Bronx Whitestone	O. H. Ammann	charm	2300
+	Throgs Neck	O. H. Ammann	hairy	1800
+	George Washington	O. H. Ammann	moody	3500
---	Spamspan	S. Spamington		10000

Figure 2.4: An example of Coopy's highlighter diff format [8].

2.3 Google Sheets

Google Sheets' [16] revision history system 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 versions right next to the actual spreadsheet, as shown in Figure 2.5, each having the list of people who edited the sheet 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 will be tinted with the unique color of their author. A curious thing to note is that Google Sheets marks a cell as edited even if the end result of the edits ends up the same as it started; this includes any formatting and formula changes.

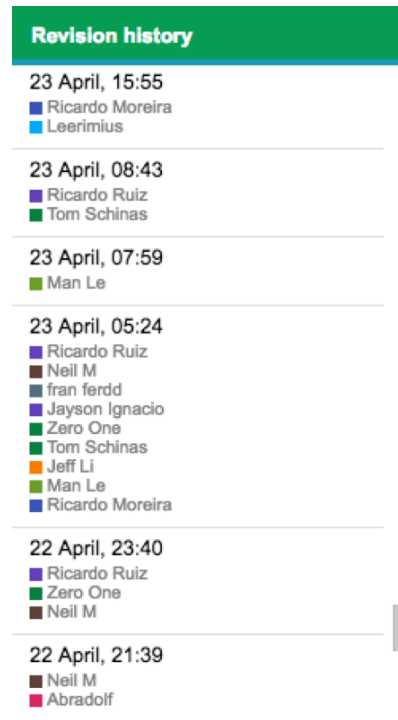


Figure 2.5: A list of versions in a Google Sheets document

Due to new versions being saved with every change done to the document, Google Sheets joins revisions that happened in a short period time between each other into a single one to make it easier to navigate through the list. The user can click a button at all times to see all of the revisions separately. Similarly, in order to save space, the document will at times suffer actual revision pruning if its file size or age is too high. This compresses various revisions into a single one, as if they all happened at the same time.

We seek for our solution to behave very similarly to Google Sheets's system, but with more control over the automatic commits and with a more robust way to show differences between the spreadsheets, as Google Sheets merely points out the locations that were changed and gives no indication of what actually happened.

2.4 XLTools

XLTools [36] is a suite of various utilities for Microsoft Excel all in one *Visual Studio Tools for Office* add-in, normally called a VSTO add-in, one of them being called Version Control [37]. 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 has also access to the list of revisions, where they can choose to compare or save individual sheets, as shown in Figure 2.6. 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.

Spreadsheet comparing can only be applied to one worksheet at a time. This will open a new Excel window with both old and new versions of the worksheet, having the new cell values tinted in red and its text in bold. The tool doesn't 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 and boldface cell formatting as the add-on uses, edited cells in the document may be confused with cells that were never touched.

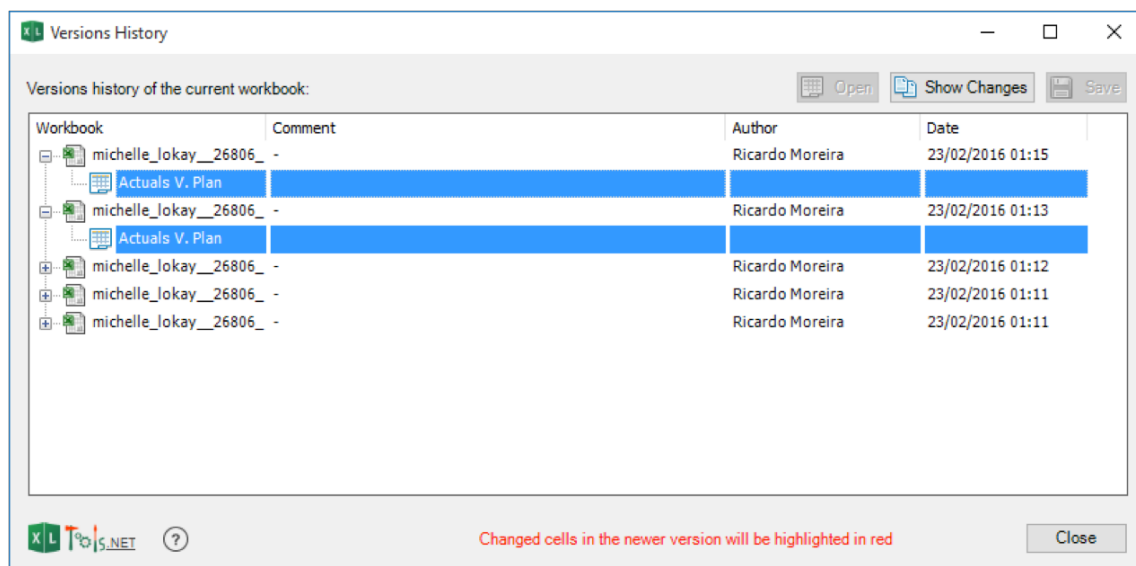
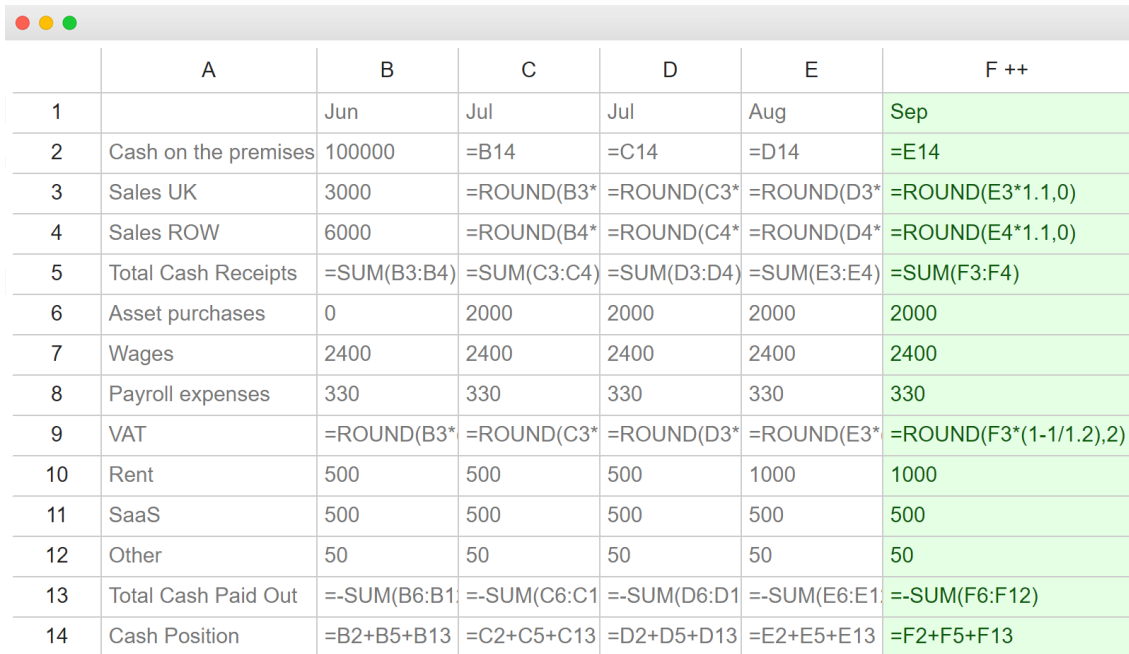


Figure 2.6: A list of spreadsheet versions in XLTools

2.5 Pathio

Pathio [31] 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, as shown in the Figure 2.7. Pathio does not work without an internet connection nor does it support branching and merging.



	A	B	C	D	E	F ++
1		Jun	Jul	Jul	Aug	Sep
2	Cash on the premises	100000	=B14	=C14	=D14	=E14
3	Sales UK	3000	=ROUND(B3*	=ROUND(C3*	=ROUND(D3*	=ROUND(E3*1.1,0)
4	Sales ROW	6000	=ROUND(B4*	=ROUND(C4*	=ROUND(D4*	=ROUND(E4*1.1,0)
5	Total Cash Receipts	=SUM(B3:B4)	=SUM(C3:C4)	=SUM(D3:D4)	=SUM(E3:E4)	=SUM(F3:F4)
6	Asset purchases	0	2000	2000	2000	2000
7	Wages	2400	2400	2400	2400	2400
8	Payroll expenses	330	330	330	330	330
9	VAT	=ROUND(B3*	=ROUND(C3*	=ROUND(D3*	=ROUND(E3*	=ROUND(F3*(1-1/1.2),2)
10	Rent	500	500	500	1000	1000
11	SaaS	500	500	500	500	500
12	Other	50	50	50	50	50
13	Total Cash Paid Out	=-SUM(B6:B1	=-SUM(C6:C1	=-SUM(D6:D1	=-SUM(E6:E1	=-SUM(F6:F12)
14	Cash Position	=B2+B5+B13	=C2+C5+C13	=D2+D5+D13	=E2+E5+E13	=F2+F5+F13

Figure 2.7: An example of Pathio's diffing [31].

VERSION CONTROL FOR SPREADSHEETS

Version control has been proven to be beneficial for end users, but it was not created with spreadsheets or end user developers in mind, thus in this chapter we introduce some concepts and methodologies in how to abstract and adapt the functionalities of version control into this setting. In each of the following sections, we present a feature usually available in version control systems.

3.1 Version listing

Despite most of the applications presented in Chapter 2 showing their list of versions in a linear dropdown list, we instead prefer presenting the list of versions in a tree format. The tree format is not something new in the version control world, yet no one had considered using it in spreadsheets.

The tree grows vertically from top to bottom in a chronological manner, an example can be seen in Figure 3.1, with each node being a different version. So, the oldest version would be at the base of the tree, and as new versions get added, the tree grows down. Since the tree in the mentioned figure has a fairly abstract appearance, it is important for something akin to arrows to be placed in between nodes, otherwise users may misunderstand which way the timeline flows. This issue may not occur if the tree is chosen to be presented in a more realistic fashion.

The problem with having regular linear lists for versioning is that they cannot easily depict a version's parent. This can lead to confusion because these lists are generally ordered in a chronological manner. This masks instances where a version is actually derived from some older version [25]. With a tree's graphical representation, we can display all of this information at once, such as creating a branch in the tree when the user restores to that version.



Figure 3.1: A representation of a simple version tree

In Figure 3.2's case, the version labeled "V4", which stands for "Version 4", in the gray branch was made after V3 in the purple branch. In a linear list V4 would show up after V3, which is not wrong, but it creates the impression that the former was created based on the latter. But in reality, their sole relation is V1, which was the base used for the purple development branch. Through this graphical representation, we can keep the version list organized in a chronological manner without losing any information on the versions' parentage.

As the graph grows more complicated, it is best to label each version so they gain a sense of individuality. Users will find it easier to pinpoint or memorize information related to a version if it has a name. In our earlier example, and in SheetGit, we put the number of the version next to the node so it is visible at all times.

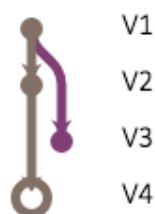


Figure 3.2: A version tree with a branch

Having information about these parent versions will help users understand how spreadsheets they did not create actually came to be, because as mentioned before, end users tend to learn from existing examples and when facing a problem, they prefer to attempt various possible solutions, backtracking when required, something which is made easier with a proper version control system.

3.2 Creating versions

Creating versions should be done automatically, as to prevent any sort of data loss due to the user forgetting to create a version, and to not interrupt the user as he/she works.

Other metrics could also be implemented, such as "Create a version after every 3 changes by the user" or "Create a version if the user, after performing a change, idles for over 10 seconds".

An option to enable manual versioning should exist for more advanced users who have a good notion of how the version control system functions. End user developers may acquire this sense over time with experience.

3.3 Moving between versions

This is what is generally called *Checkout* in Git. Versions are only useful if a person can actually use them, either to just give it a glance to find specific information or to perform a complete rollback.

Since a graphical version tree would be employed with this work's methodology, it is possible to have the user interact directly with the tree to perform actions. This would also help them understand how the tree and the whole system behind it functions.

Thus, we propose having the user interact with the version nodes of the tree to change versions, such as clicking them. In this case, it is also important to show an indication of what version of the spreadsheet the user is currently seeing. In Figure 3.3's case, we have the current version displayed as a larger node with a white interior as to make it stand out from its siblings.



Figure 3.3: A tree where the visible, current spreadsheet is its third version

3.4 Creating branches

As suggested earlier, branches are used to keep versions's parentage visible to the end user. Much like versions, branches should be created automatically. More specifically, they should be created whenever a user attempts to create a version when they are not located at the tip, the latest node, of a branch.

Using the same Figure 3.2 as earlier, the user in that case would have moved from V2, the latest version before any branches existed, to V1 and then performed some changes to the spreadsheet. This would cause the automated creation of the purple branch and a new version labeled V3.

3.5 Version messages and tags

The version tree can grow confusing after it acquires a large amount of version nodes. Versions can be organized by the user by letting them add version messages and tags to the versions. Tags are messages to label specific points in history as important.

Tags should be shown directly on the graph, so they can be visible even if the user is not looking at that version in specific. An example of this based on the previous figures would be to put the tag next to the version number.

Version messages can be more detailed, and accessed by interacting with the version node. For example, hovering with the mouse over the version node should show a speech bubble with all the information pertaining to it. Since version creation is automated by default, version messages could also be automated in the same vein, by for example detailing it with the changes that were done in that specific version.

An example detailing these methodologies can be seen in Figure 3.4, where the mouse would be hovering the third version to show a message, and the sixth version has an always visible tag.

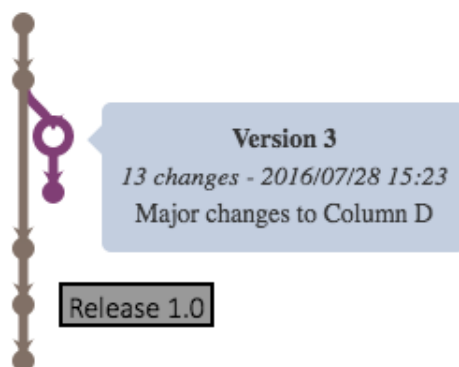


Figure 3.4: Version tree displaying a version message and a version tag

3.6 Diffing

Considering the user can already change versions, and have the effect show directly on the Excel spreadsheet, it is only intuitive to keep the diffing process the same.

In order to make a scaleable yet easy to understand comparison in complex spreadsheets we suggest thinking about a timeline with one version at one end, and the other at the opposite end, as in Figure 3.5 Between the two versions would be all of the cells with any differences between the two. Supposing the user starts in "Version A" and walks through the timeline to the other end, as the user passes through the cells, their values would change in the visible spreadsheet to how they are in Version B. So when the user

reaches the end, the spreadsheet at that point would be exactly like it is in Version B. This would all apply in vice-versa as well, from Version B to A.

Using our figure as an example, in Version A, the A1 cell would have the value "hello", but if we move one step to the right in the timeline, it would display as "goodbye". At this point if we moved back to the left it would revert, and if we instead moved onwards to the right we would move to different cells, the same process applying to them, gradually turning into Version B. This would also apply to entire lines or columns if they were edited in one step, such as their creation and deletion, as seen in the second step of Figure 3.5

This allows the user to see the differences step-by-step rather than being overwhelmed with all of them at the same time. This would scale reasonably well with spreadsheet size and complexity as long as the user has control of the speed of traversal through 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 also have to be 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 area's color or by drawing a circle around it.

Diffing should also be accessible directly through Excel and the version tree graph. An example would be to right-click a node and choosing to either diff or move (*checkout*) to it. Another one could be to have a button toggle the two different options for when clicking the nodes. SheetGit uses the latter option.

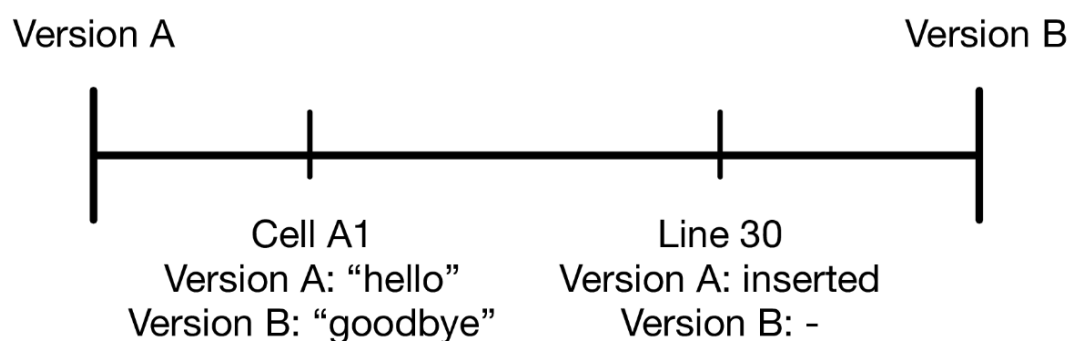


Figure 3.5: Timeline between two versions

3.7 Collaboration

In order to avoid confusion and workspace clutter, users cannot see each other's branches, only their own. Instead, they interact with each other through a single branch, the left-most branch of the graphs shown until now, which we call the trunk. This allows users

to use their branches as temporary, in development work, that is hidden to other people, while the trunk is meant for finalized versions that are ready to be shown to other colleagues.

Users cannot create versions normally in the trunk because multiple users can interact with it, creating the possibility of conflicts arising if two users edited the same cells concurrently. Thus the trunk can only be updated through merges as to detect and resolve any conflicts that may arise.

It is important for the trunk to be distinct from the other branches, whether it be through color or some other visual change, as it behaves very differently from other branches.

3.8 Conflict resolution

Conflicts should be solved directly on the spreadsheet, like how text conflicts are resolved in the actual text file in Git.

Conflicted cells, lines, or rows should be highlighted for the user to differentiate them from the rest of the spreadsheet. When one interacts with the conflicted area, one should be immediately shown what are the options to take, where do they stem from, and what was the original value before the two conflicted versions changed it. This gives the user context and allows a person to deliberate while being able to look at the current status of the spreadsheet around the conflict.

The user should be able to choose the options directly in Excel but not by editing the cell directly. This can lead to human error, so the options should be immutable and absolute. If the user decides neither option is suddenly of his interest, those changes should be left for after the conflicts have been resolved to keep the version history coherent.

Figure 3.6 shows a way to handle conflict resolution. Dropdowns are used to select options, which contain information about the branch where they are from, the version's cell value and what the original cell value used to be.

After the merge, the program should also create a new version in the trunk, with two lines pointing towards both its parents in the trunk and the branch that was merged. The example tree in Figure 3.2 and Figure 3.3 can be seen after the merge in Figure 3.7.

bh

Actual	Plan
(51 497)	45 000
<CONFLICT>	16 250
<CONFLICT>	2 500
(9 470)	8 753
<CONFLICT>	8 875
T: {"Value":1058.427,"Original":858.427}	29 545
B: {"Value":558.0,"Original":858.427}	13 306
<CONFLICT>	44 000
<CONFLICT>	3 750
<CONFLICT>	1 603
-	-

Figure 3.6: Resolving a conflict in a cell

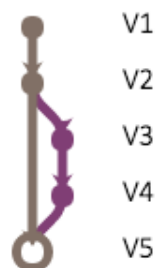


Figure 3.7: A version tree in SheetGit after a merge

SHEETGIT: A VERSION CONTROL SYSTEM FOR SPREADSHEETS

As version control has been proven to be beneficial for end users, what remains is to actually create a solution with an attractive interface and reliable functionalities that cater to their needs.

In this chapter we will discuss the details of our application, which implements the concepts and methodologies detailed in the previous chapter.

4.1 Overview

We have decided to create our solution as an add-in to keep it as closely knit to Excel as possible. An overview of the application within Excel can be seen in Figure 4.1 (top ribbon and right-hand side pane), this makes it more pleasant to work with, as the tool is inside the spreadsheet application itself, rather than having to be run externally. This also enables us to present information directly in the active spreadsheet, and grant us easy access to Excel's proprietary file formats.

There are two main types of add-ins for Excel, the VSTO, which stands for *Visual Studio Tools for Office*, add-ins made in C# or Visual Basic, and the new type that Microsoft simply calls *Office add-ins*. The latter are simply webpages 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, macOS and iPad.

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

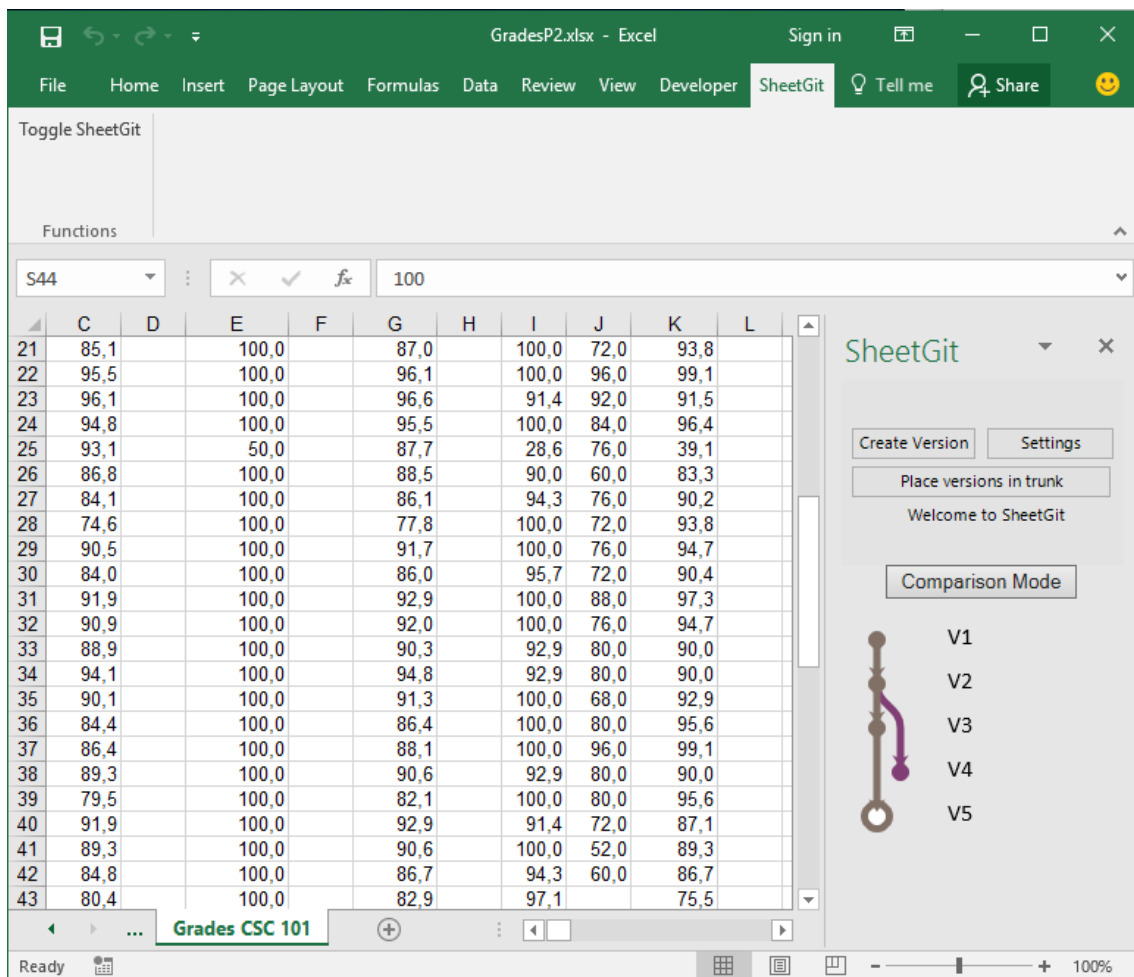


Figure 4.1: SheetGit in Excel 2016

the Javascript add-ins due to their sandboxed nature. As a result, our solution will only support Microsoft Excel in the Windows platforms.

4.2 Backend

The application creates a folder in the user's Program Files by default, and also in the user's Application Data. The Application Data folder is used to store the settings and all the Git repositories. So, the user can create and edit his/her Excel spreadsheets anywhere in the computer, and the repositories will always be stored in that one folder. This way, the process is completely invisible to the user.

When the user creates a spreadsheet, the application in turn creates a Git repository in the Applications Data folder and a remote one in Bitbucket if online connectivity is enabled. This repository tracks the spreadsheet and a JSON file. This JSON contains additional information about the versions such as the list of changes, the parent of each commit and its branch. Portions of the JSON are used for managing the list of versions

visible to the user and the diffing and merging procedures. The JSON file follows the schema visible in Appendix G, which follows the IETF JSON Schema Internet Draft Version 4 standard [19][20][21]. In the span of this work, our tool only detects changes in cell values and formulae, though there are no technical limitations that prevents SheetGit in the future from spanning far more content types in Excel.

4.3 Ribbon tab

The ribbon is the strip of buttons and icons located above Excel's work area that was first introduced with Excel 2007. SheetGit has its own ribbon tab, shown in Figure 4.2 with a sole *Toggle SheetGit* button. This button will toggle SheetGit's visibility and behavior on and off as the user wishes.

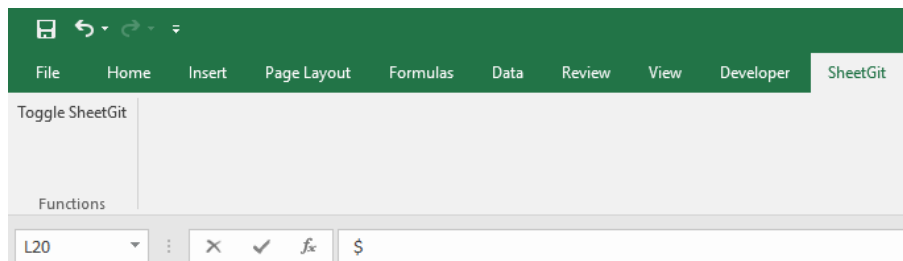


Figure 4.2: SheetGit's Ribbon Tab

4.4 Main Pane

The application makes use of the Excel task pane, to not disturb the user's work yet remaining visible at all times. The main pane can be seen in Figure 4.3. The objective is to keep the list of versions in the task pane, and updated in real time, so users can understand how it is being built and have an idea of how their changes are spread throughout the versions.

The task pane is created using Windows Forms, and the main pane contains an embedded web browser so we can make use of Javascript which makes it easier to show graphical information.

We make use of a version tree, as detailed in the previous chapter, to show our list of versions, with the base version being at the top and having the tree grow downwards. The tree is generated by the Javascript library GitGraph [15].

As described before, SheetGit uses a toggle to switch what happens when a version is clicked. The *Comparison Mode* button serves this purpose, switching between moving to the selected version and comparing the selected version with the current one. As seen in Figure 4.4, the pane will be tinted red to show the user that the application is in Comparison Mode.

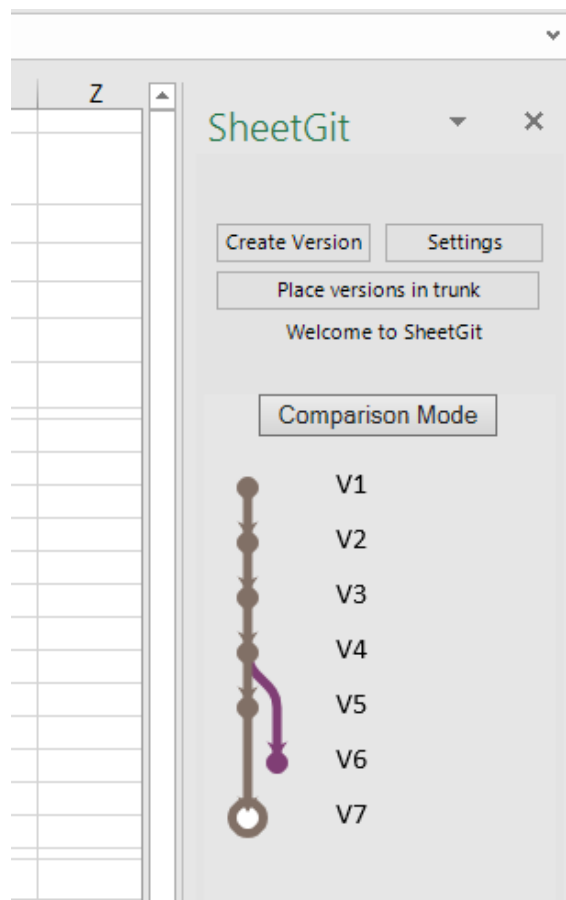


Figure 4.3: SheetGit's main task pane display

The *Create Version* button manually creates a version if there are any recorded changes to the spreadsheet. The button is disabled if the application is in its default mode where it creates a version for each change.

The *Settings* button changes the SheetGit to the Settings Pane. This pane is explained in detail on Section 4.5

The *Place versions in trunk* button serves to perform merges into the trunk.

When information or errors need to be shown to the user, the text that currently displays "Welcome to SheetGit" is temporarily changed to the message.

4.5 Settings Pane

The settings pane, as seen in Figure 4.5 allows the user to change how certain parts of the application function, and to provide it with information for collaboration.

The *Back* button returns the user to the Main Pane.

SheetGit does support collaboration between multiple users, with the spreadsheets being hosted at Bitbucket [1]. The *Bitbucket account* fields and the *Grant Permission* button are to temporarily provide the user's Bitbucket account information to the application.

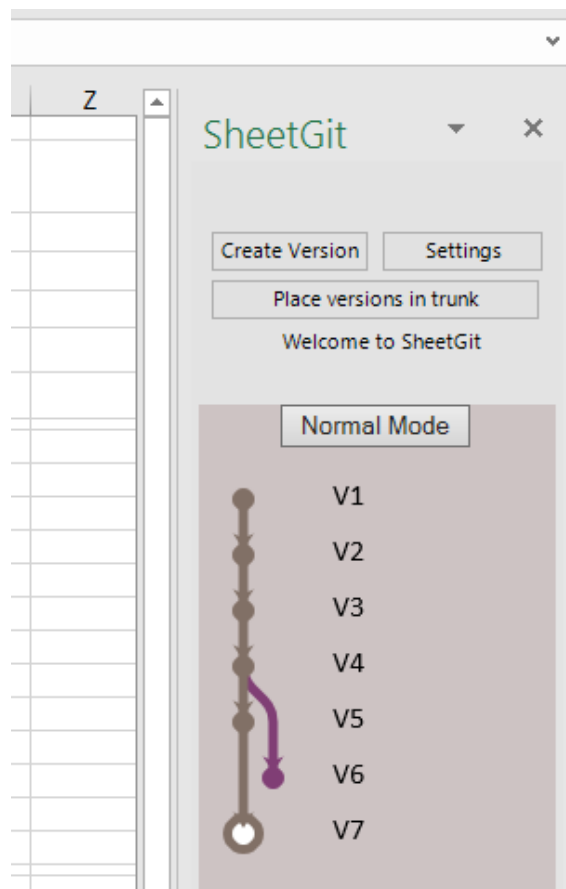


Figure 4.4: SheetGit's main pane in Comparison Mode

The button will initiate the OAuth2.0 protocol to grant SheetGit permission to create and manage the user's Bitbucket-hosted spreadsheets.

The *User information* fields and the *Update* button are for the user to write his/her name and email as it will be displayed on each version. Currently SheetGit does not show this information for each version, but it is shown on Bitbucket's website and in any application that can read local Git repositories.

The *Versioning metrics* dropdown allows users to choose how versioning is performed between

1. Every workbook change
2. Every 5 workbook changes
3. After 5 seconds of idle time
4. Manual versioning

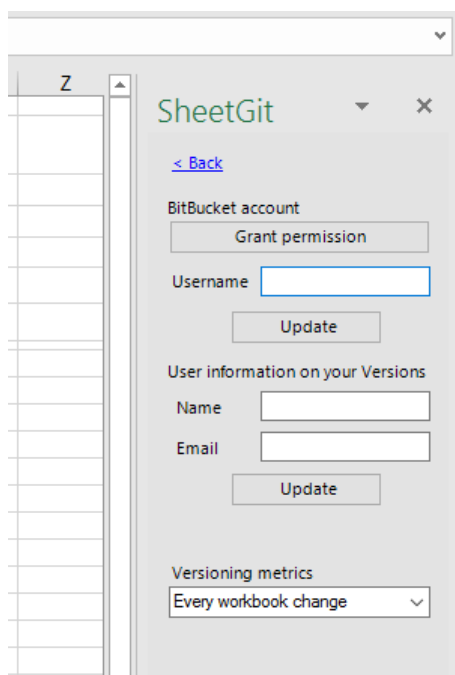


Figure 4.5: SheetGit's Settings Pane

4.6 Diff Pane

The Diff Pane, as shown in Figure 4.6, contains all functionality related to viewing differences between two versions.

The slider moves through each cell that has differences between the two versions, as detailed in the previous chapter. 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 around.

To help the user understand which cells are changing, the text on the right of the slider, shows what cell the slider is pointing towards, alongside the values and formulae for the cell in both versions. The cells' colors also change when the slider passes through them, aiding the user in finding them. This can be seen in both Figure 4.6 and Figure 4.7, where in the latter picture, the left side is how the spreadsheet is normally, and on the right is how it is when the slider passes by a cell. In this case it is tinted green because it was a value that was changed, in case of formulae, the tint is instead purple. The legend for the colors is shown in the diff pane's informative text, also shown in Figure 4.6.

Differences are detected by means of comparing worksheets to a prior copy of them, using the VSTO API to check which parts of a cell have changed, whether it be its value or formatting. So, all changes to a spreadsheet are saved as the versions are created and are then compared in the diffing procedure.

The text below the *Exit Comparison* button explains how the diffing process works, while the one next to the slider points out the details of the current cell in the two versions.

The actual *Exit Comparison* button returns the user to the Main Pane. Any changes to

the spreadsheet as result of the diffing are reverted.

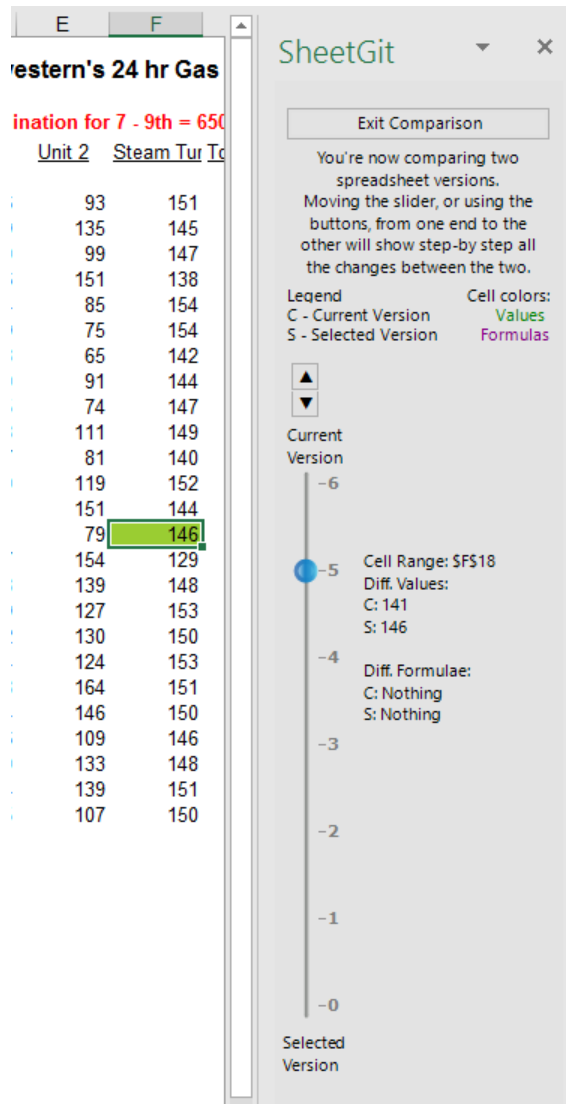


Figure 4.6: SheetGit's Diff Pane

Actual	Plan	Variance	Actual	Plan	Variance
(51 497)	45 000	(96 497)	(51 497)	45 000	(96 497)
3 460	16 250	(12 790)	2 460	16 250	(13 790)
3 154	2 500	654	3 054	2 500	554
(9 470)	8 753	(18 223)	(9 470)	8 753	(18 223)
858	8 875	(8 017)	858	8 875	(8 017)
3 316	29 545	(26 229)	3 316	29 545	(26 229)
(152)	12 206	(12 458)	(152)	12 206	(12 458)

Figure 4.7: SheetGit during a diffing procedure

4.7 Conflicts Pane

When one decides to merge, one of the two following cases will happen:

Three-way merge

In case the trunk has had new versions since the branch was created, there is the possibility of conflicts arising, so a three-way merge algorithm is employed, with the user having to solve any conflicts that arise. This is the case that would happen if the previous example of Figure 4.3 was executed. This algorithm is used in most big version control solutions such as Git [14] and diff3 [22].

Fast-forward merge

In case the trunk has not changed since the branch was created, since there are no conflicts, no actual merge of versions to perform, they could technically be seen as just one branch. As such, the application will apply the fast-forward method used in Git [14], which has the trunk point to the latest version of the branch, making them effectively one branch altogether. The user will see their branch becoming part of the trunk, and will require no further input from them. This is meant to simplify the version tree, as the three-way merge could be applied in this situation as well.

The Conflicts Pane, as seen in Figure 4.8, will automatically arise during a three-way merging process with conflicts, and will naturally disappear when these are dealt with.

The pane itself just has text explaining what is happening and what the user should do to fix the conflicts, alongside a large red note and number showing how many are conflicts remaining. This number is automatically updated as the situation changes.

When conflicts are detected, automatic versioning stops until they have been resolved. Conflicting cells are shown with a special "<CONFLICT>" value, and a drop down that has the two different versions of the cell. In the dropdown, the options begin with either a "T" or a "B". "T" stands for trunk, the version we are trying to merge into, while "B" stands for branch, the version we were on before beginning the merging process. Followed by this prefix is the data type that is to be changed, in this case it is a regular cell value but it could be a formula. The "Original" parameter stands for the value in the parent of the two conflicting versions.

Once the user selects one of options, the cell will take that value and return to normal. Once all conflicts are resolved, the program will return to normal functionality.

4.8 Availability

The source code is available in Github as part of SpreadsheetsUNL, a group of spreadsheet related tools developed in Universidade Nova de Lisboa [33].

	V	W
15	e: ((Prog 1 + Prog 2 + ProgQuest Total) 3	
16		
17	ing Questions	
18	2	3
19	100,0	100,0
20	80,0	86,7
21	90,0	100,0
22	80,0	100,0
23	70,0	86,7
24	40,0	<CONFLICT>
25	70,0	T: {"Value":65.0,"Original":86.7}
26	50,0	B: {"Value":26.0,"Original":86.7}
27	90,0	93,3
28	90,0	100,0
29	90,0	80,0
30	70,0	73,3
31	70,0	80,0
32		100,0
33	90,0	93,3
34	70,0	100,0

SheetGit

Conflicts have been found between the trunk and your branch.

You will see them as cells in yellow saying <CONFLICT>.

Click them and use the dropdown

Conflicts remaining: 2

The first letter in a dropdown option shows which branch it originated from.

B - Branch, the current version
T - Trunk

Figure 4.8: Resolving conflicts in SheetGit

EMPIRICAL VALIDATION

An empirical validation is widely recognized as essential in order to validate a new application. Thus, we have prepared an empirical study, which is described and analyzed in this chapter. Our motivation to perform this study is the need to understand if users are more efficient and effective at performing certain tasks in spreadsheets with our tool rather than without it.

In Section 5.1 we detail the design of our study and in section 5.2 we explain how it was executed. In Section 5.3 we then analyse the collected data followed by its interpretation in Section 5.4.

5.1 Design

The aim of our study is to evaluate the effectiveness and efficiency of users using our application when performing certain tasks, compared to simply using Excel and Spreadsheet Compare, which comes bundled with Microsoft Office.

As we have previously described, it is common for errors to occur when editing spreadsheets. Our ambition is to mitigate this problem. Thus, evaluating the effectiveness and efficiency of users using SheetGit is quite important.

The application's target audience are spreadsheet end users, therefore our subjects were intended to be people with at least minimal Excel experience and few to no programming experience. Participants performed two tasks per spreadsheet across two different spreadsheets provided by us. Those spreadsheets were retrieved from EUSES [6] and from the spreadsheet corpus VEnron [5], which is originally based on the Enron company's email corpus [17].

The study we have designed was applied in an academic environment, with freshmen computer science students. To provide incentive for participation, we decided to raffle a

voucher with the value of fifty euro for a retail store chain in Portugal that sells cultural and electronic products, the winner was a student named João Silva.

5.1.1 Hypothesis

In theory, using SheetGit reduces the number of errors and improves the user's speed in performing certain tasks compared to Excel and Spreadsheet Compare. However, this needs to be tested. So, we could informally state 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, two hypotheses are being tested: 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 only 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} : $\mu_d > 0$, that is, the time to perform a given set of tasks using SheetGit is less than using only 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 only Excel and Spreadsheet Compare. $H_{C_0} : \mu_d \leq 0$, where μ_d is the mean difference of the correctness grades (effectiveness).

Alternative hypothesis, H_{C_1} : $\mu_d > 0$, that is, the correctness grade when using SheetGit is smaller than when using only Excel and Spreadsheet Compare.

Measures needed: correctness grade for each spreadsheet.

5.1.2 Variables

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

5.1.3 Subjects and Objects

The subjects of this study should be end user developers, people who have at least minimal experience with Excel, but not much if any programming experience. In this case specifically, our subjects should not have any experience with version control, because it would possibly form a bias towards our tool.

In the end, the study was performed with freshmen computer science students from the Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. They were chosen because they are likely to have minimal experience with Excel, yet not enough programming experience to know about version control.

In order to find the population with the desired requirements to this study, we created a selection questionnaire (Appendix A in Portuguese) as a way to evaluate the subjects's knowledge of Excel and version control.

The objects of this study are three distinct spreadsheets that will be described later in section 5.1.4. One spreadsheet is used as a tutorial, explaining half of the participants how to use SheetGit, and the other half, Spreadsheet Compare. Spreadsheet Compare being a tool bundled by Microsoft with Excel, we thought it would be easier to perform the tasks with the tool's help, yet since the subjects may not know about its existence, a tutorial was provided. The remaining two spreadsheets are used in the tasks that the participants have to complete.

5.1.4 Instrumentation

As we have been describing, our study is supported by three distinct spreadsheets. Each spreadsheet contains multiple versions, in the form of Git repositories for SheetGit and as regular separate Excel files for Spreadsheet Compare.

The subjects are not asked to understand the context of the spreadsheet. Thus, the three spreadsheets were chosen to be simple to understand. The tutorial one, termed Southpoint, was taken from the VEnron spreadsheet corpus [5] and calculates the total gas usage when given input values. The second spreadsheet, termed Grades, was taken from the EUSES corpus [6], and manages and calculates grades for students of an university course. The third spreadsheet was retrieved from Enron as well and is termed Markets. This one calculates the income of Enron's global markets.

Participants received a set of tasks (Appendix C and E), two different questions for each of the three spreadsheets. The two questions were similar in content across the three spreadsheets, as to see if the users could perform similar tasks in different spreadsheets.

The first question 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 question asks users to unite two versions of a spreadsheet into one, assuming they both had the same parent. Users would then have to compare the three spreadsheets (the parent and the two different children) and then perform a three-way merge.

In order to understand the hardships participants can encounter during the study, two questionnaires were prepared: one to answer before the study, the pre-questionnaire (Appendix A), and another to answer afterwards, the post-questionnaire (Appendix F).

Before the participants left the room, we collected the spreadsheets with their proposed solutions from their computers.

5.1.5 Data Collecting Procedure

We have planned several steps to run our study, with two different options: perform the tasks with and without SheetGit is help, as an attempt to compare the efficiency and effectiveness of performing the set tasks. Therefore, the first option consists of five phases:

1. Filling the pre-questionnaire (Appendix A);
2. Attending and performing the tutorial on SheetGit (Appendix D)
3. Performing the set of tasks on the two given spreadsheets (Appendix E);
4. Filling the post-questionnaire (Appendix F);
5. Collecting all spreadsheets, questionnaires and answers.

Regarding the second option, the participants would instead attend a tutorial on Spreadsheet Compare. Even though the tool is bundled with Microsoft Office, the users might not have any experience with it, thus we decided to create a tutorial for it.

In steps (2) and (5) we would directly interact with the participants, teaching them the tutorial answering any questions in the former step, and by retrieving their results and materials in the latter. No help is provided to the participants while they were executing the tasks in step (3).

5.1.6 Analysis Procedure and Evaluation of Validity

The analysis of the collected data is achieved through the comparison of the group of participants that performed the tasks using our application with the group of participants that perform the tasks with just Excel and Spreadsheet Compare. Since the study is comprised of several tasks, the participants note down the time it took for each user to complete the tasks.

To perform the comparison, we calculate the average timespan it took for the participants to complete the tasks in each group, and then compare both of them. To ensure the validity of the collected data, we would offer various kinds of support to the participants, such as the tutorial for the tools, availability to answer any questions during the tutorial, and supervision in a manner that does not disturb them while they work. In this last point, supervision serves to help participants in case they have issues that if solved would not influence the study results.

5.2 Execution

The study was performed in a classroom during a freshman Computer Science course. A total of sixteen college students participated across two sessions, one for the tasks with SheetGit, the other without it.

Initially, we had scheduled to perform the study outside classes, recruiting people through email with scheduled sessions across the week. Despite providing incentives to participate, such as attending a small workshop for the users to learn version control and the chance at winning the voucher mentioned before, not enough participants showed up.

SheetGit and Excel 2013 was installed beforehand in the classroom where the study was conducted. Before each session, we personally verified if the environment was correctly set and ready for the participants. When they were already in the classroom and seated, we introduced the purpose of the study, explaining what we developed so far and why their participation was important.

Afterwards, the participants started filling the pre-questionnaire, with general information about themselves (gender, age range, scholar year) as well as some questions about their previous experience with spreadsheets, programming and version control, if any.

We then provided an interactive tutorial with SheetGit and Spreadsheet Compare, depending on which session it was, and answered all participants's questions. They then had to solve the two tasks per spreadsheet by themselves.

Regarding the sessions, the participants were split between the two types of tasks, as to have an equal amount of people with and without SheetGit. We also decided to alternate which spreadsheet the participants started the study with. In other words, some participants started with the Markets spreadsheet and others with the Grades spreadsheet. This is important to get more realistic and even results, 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.

Lastly, we have asked participants to answer the post-questionnaire in order to evaluate the confidence that they had on their performance during the study and afterwards we have collected the modified spreadsheet files, the questionnaires, the answers of each spreadsheet as well as the times to perform each task, so we could analyze them at a later time.

5.3 Analysis

In order to perform a quantitative analysis of the study, we used all the subjects's results: Nine subjects that used SheetGit and nine that did not.

5.3.1 Descriptive Statistics

Subjects:

Basic information about the subjects was gathered, namely their gender, age, familiarity with spreadsheets and version control. From the eighteen subjects, sixteen were male and two were female. All of the subjects were below twenty years old. All of the students were freshmen in the Computer Science degree. 62% of the subjects had experience with Excel and only 25% have had experience with version control.

Time spent (efficiency):

There were noticeable differences in the time the participants used to perform the study. Figures 5.1 and 5.2 show the time it took for the participants to perform each of the Grades spreadsheet's tasks. The Y axis contains the time in seconds while the X axis points out whether SheetGit was used in turn of Spreadsheet Compare. Likewise, Figures 5.3 and 5.4 are the equivalent for the Markets spreadsheet.

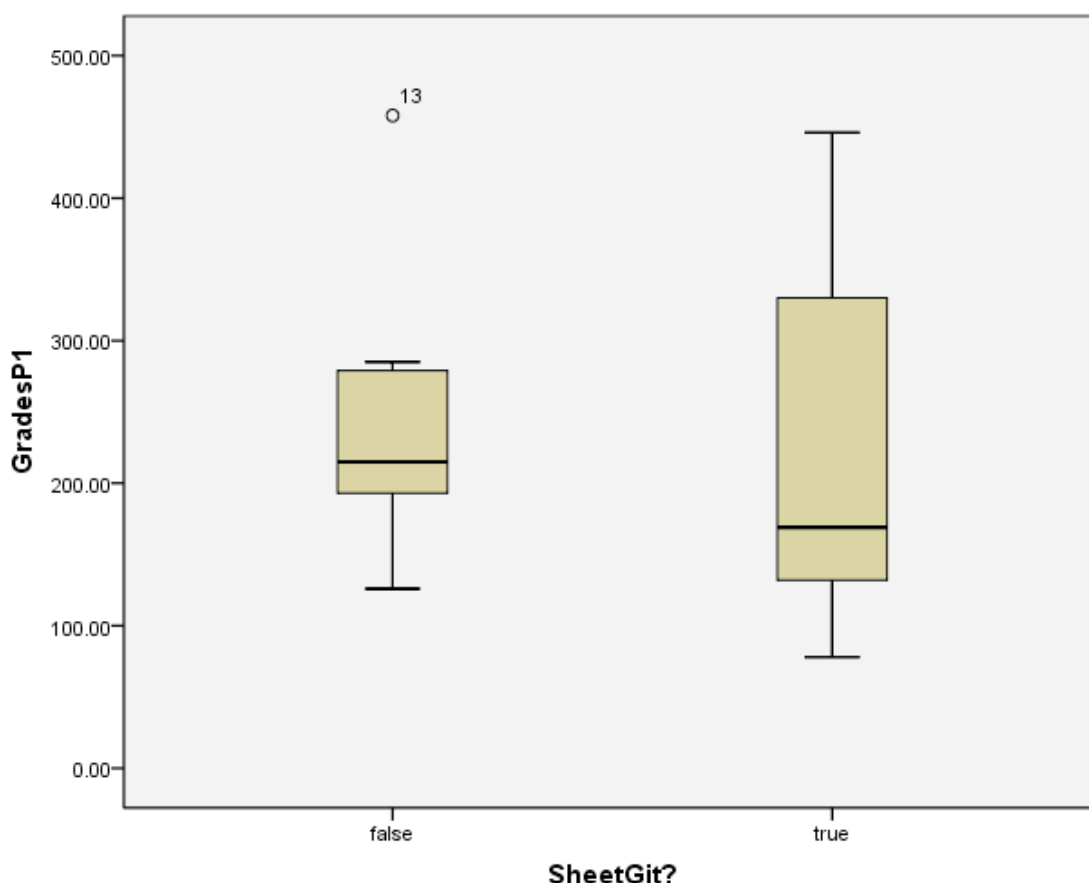


Figure 5.1: Box plot for the time elapsed in executing Grades's first task

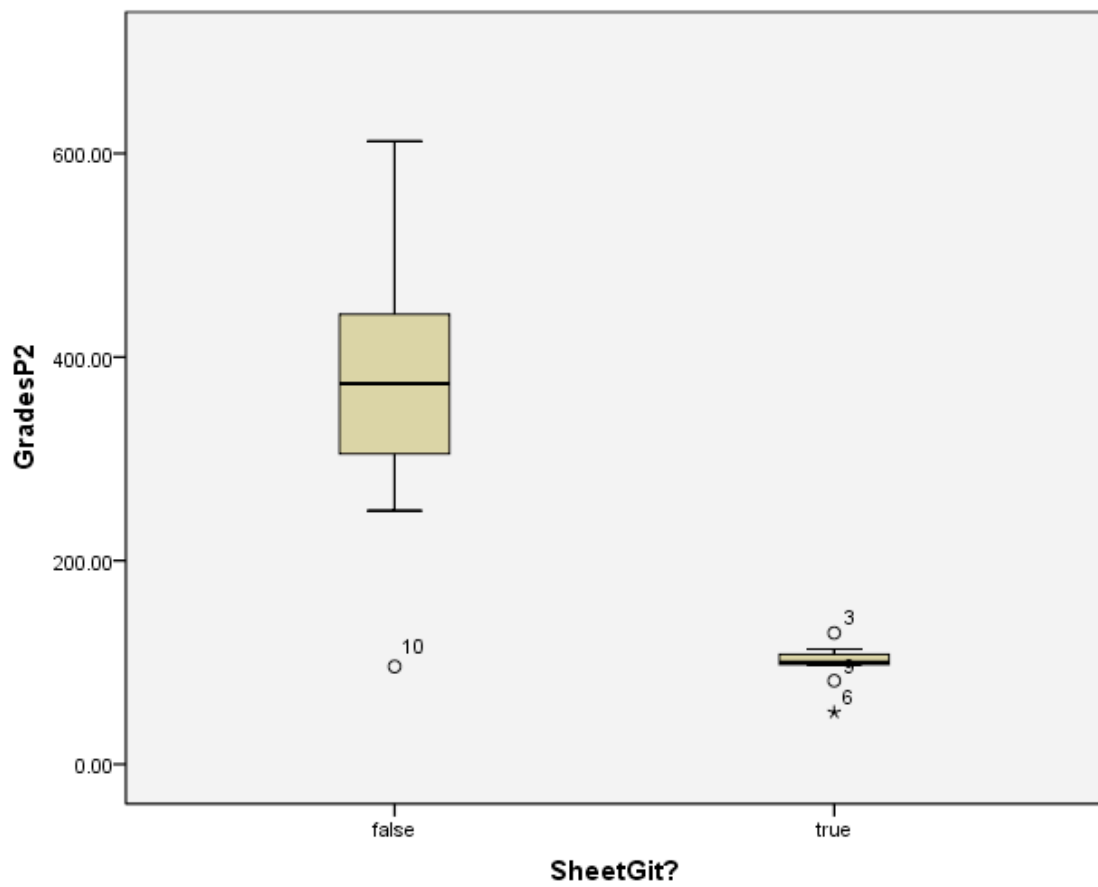


Figure 5.2: Box plot for the time elapsed in executing Grades's second task

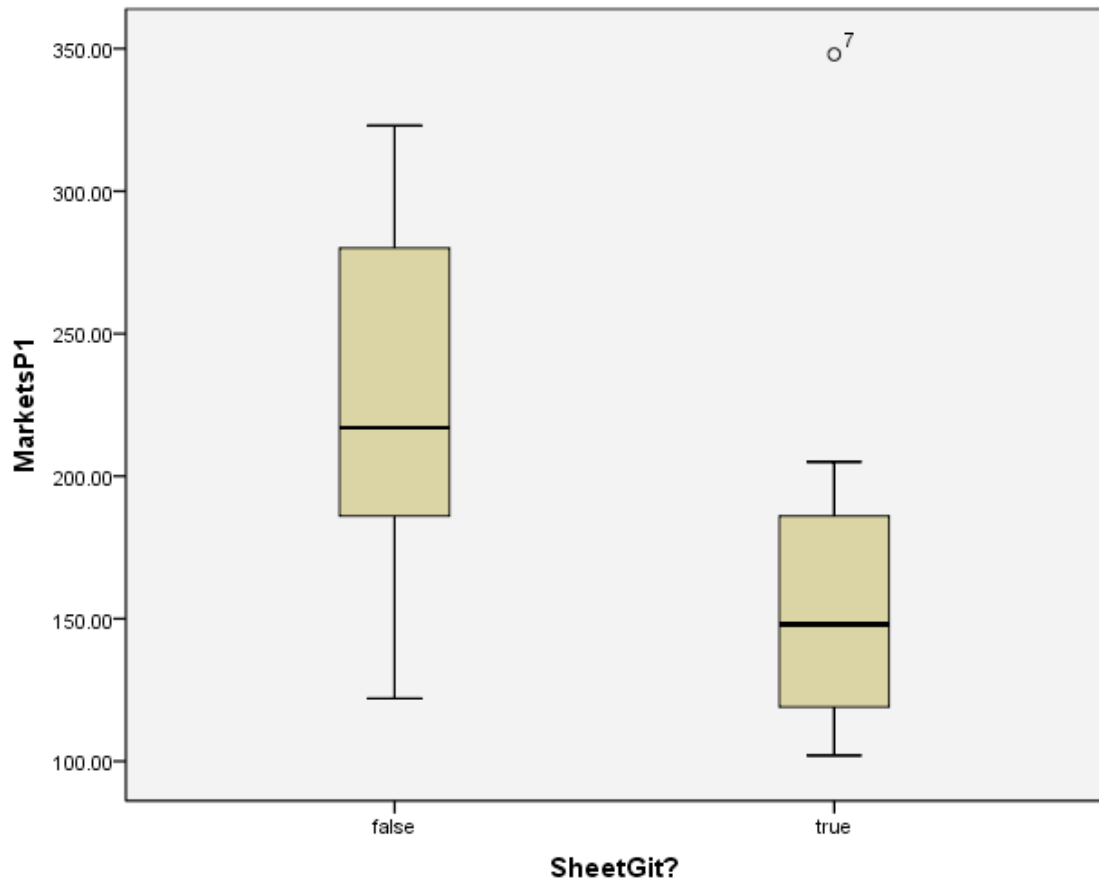


Figure 5.3: Box plot for the time elapsed in executing Markets's first task

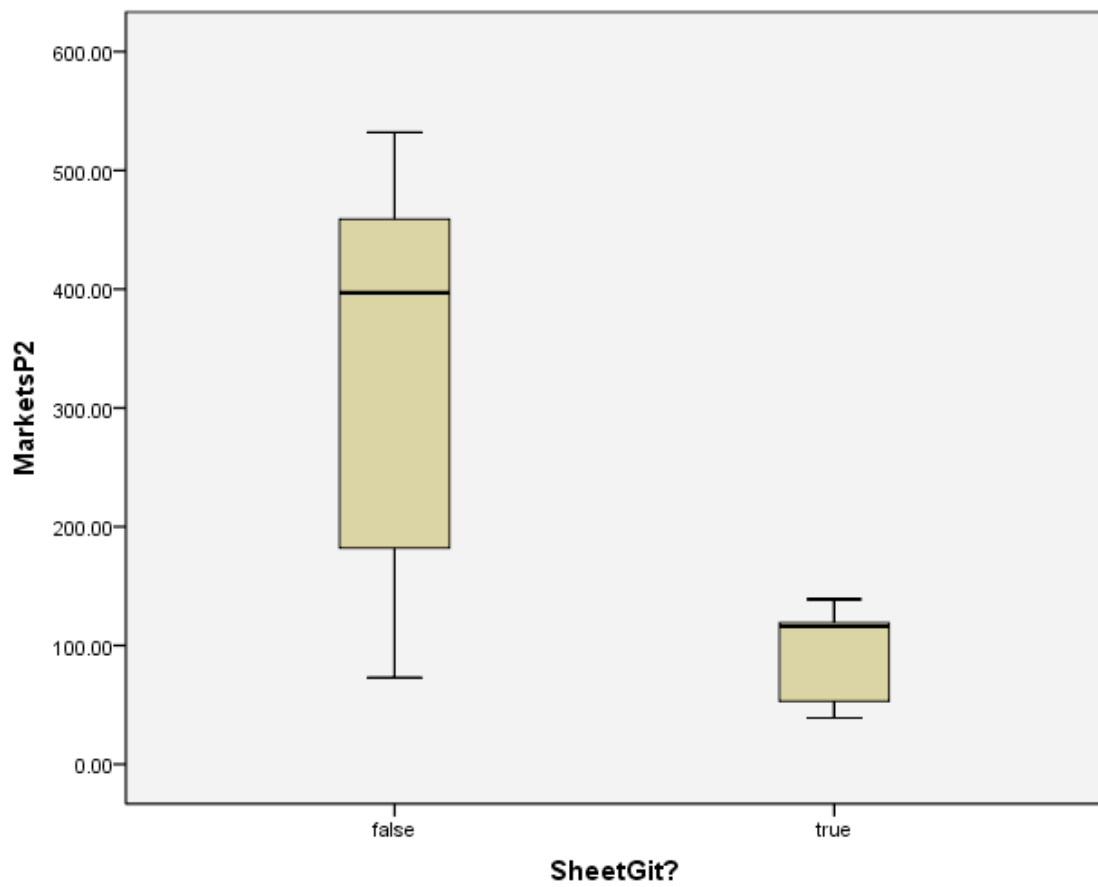


Figure 5.4: Box plot for the time elapsed in executing Markets's second task

Correctness grade (effectiveness):

In regards to the correctness grade, 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 5.5 shows the number of participants who committed the former error, while Figure 5.6 the latter. The charts show all the spreadsheets' results together, the Y axis showing the number of people and the X axis whether SheetGit was used or not in place of Spreadsheet Compare.



Figure 5.5: Bar chart with the amount of participants who inputted wrong values in the right version

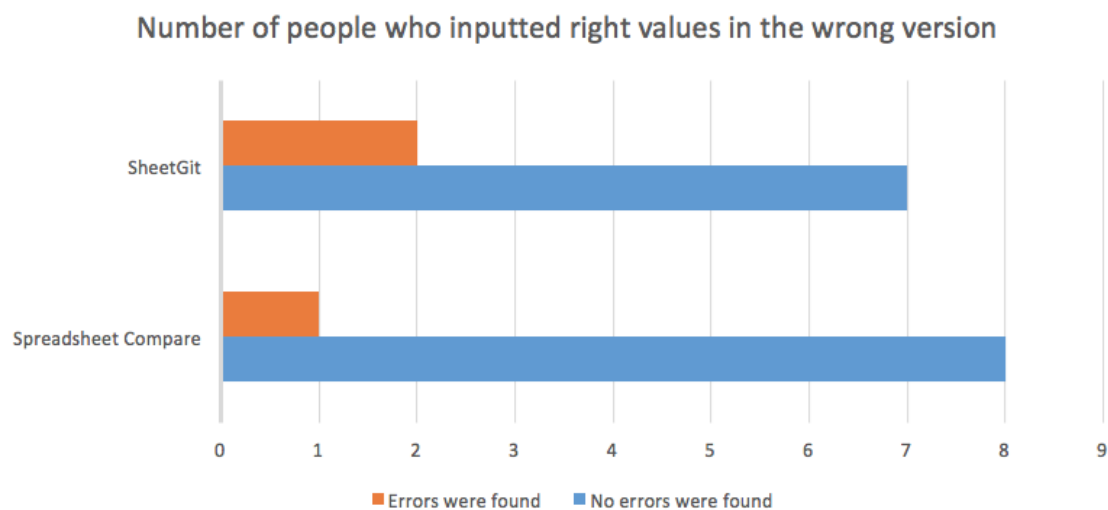


Figure 5.6: Bar chart with the amount of participants who inputted correct values in wrong versions

5.3.2 Hypothesis Testing

To test our hypothesis on efficiency, we ran a Welch’s unequal variances t-test to determine if there’s any statistical significance in our study, as suggested in [23]. The results are presented in Table 5.1. GradesP1 and P2 are the first and second tasks of the Grades spreadsheet respectively, the same rule applying for the Markets spreadsheets.

Table 5.1: Welch’s unequal variances t-test results

	t	df	Two-tailed P	Mean difference	Std. error difference
GradesP1	0.418	14.689	0.682	22.11111	52.91689
GradesP2	5.302	8.357	0.001	262.55556	49.51565
MarketsP1	1.664	15.691	0.116	55.77778	33.52118
MarketsP2	3.859	8.791	0.004	228.00000	59.07933

This is followed by calculating Cohen’s d to determine our effect size in Table 5.2, also suggested in [23].

Table 5.2: Calculation of Cohen’s d

	SheetGit?	Mean	Std. deviation	Std. error mean	Cohen’s d
GradesP1	false	244.4444	94.00015	31.33338	0.196974
	true	222.3333	127.92869	42.64290	
GradesP2	false	361.3333	146.91664	48.97221	2.499612
	true	98.7778	21.94754	7.31585	
MarketsP1	false	224.3333	65.92989	21.97663	0.784395
	true	168.5556	75.93601	25.31200	
MarketsP2	false	323.4444	94.00015	57.66742	1.819253
	true	95.4444	38.51659	12.83886	

Comparison of times

From the t-test results we can deduce that only the P2 tasks have statistical relevance, and through Cohen’s d we can see that in P2’s case the difference between the two means can be classified as very large. So SheetGit in these tasks helped the participants in a scale of more than one standard deviation, which is very impressive. For the P1 tasks, there was no statistical relevance within the study.

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.

5.4 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.

In regards to the first task, related to diffing and correcting errors, there was no statistical relevance 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, it is simply providing a different interface. 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 up the 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.

We intend to improve SheetGit based on these results, this will be explained in more detail in Section 6.2.

5.4.1 Threats to validity

The goal of the study was to show that it is better to use SheetGit to perform these tasks than to not use it. Multiple validity threats exist, these were analysed and split into four

categories as defined in [3]: Internal validity, conclusion validity, construct validity and external validity.

Internal validity

In order to minimize any effects on the independent variables that would reflect on the casuality, several actions were taken. First, half the participants started with the Markets spreadsheet, and the other half with the Grades spreadsheet. 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.

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 in order which increases our statistical power.

Construct validity

The 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.

External validity

This validity is related to the strength to generalize the results of this study to industrial practice. Due to this, we have selected two spreadsheets from the real-world: one from an actual company [17] and another from the EUSES corpus, which in turn retrieved it from a Google search as part of a real-world example [6]. 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.

5.4.2 Inferences

Since this study was performed in a very specific environment, we cannot generalize it to every case. Nevertheless, the environment used to perform this study was as similar as possible to a real one, in which end users are normally non-professional programmers and in which spreadsheets are already developed with a specific purpose. Therefore, the used

spreadsheets were based on real cases, and the majority of the students which preformed the study had few to no programming experience. Our application was developed mainly for end users, so it could be useful if applied in a professional industry.

5.5 Discussion

The empirical study we have 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 necessary for such a tool to exist.

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 all 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 is diff tab, which has a thorough and detailed explanation of its functionality and appearance directly on the interface.

C H A P T E R



CONCLUSIONS

In this last chapter we present some concluding remarks in section 6.1 and future work in section 6.2

6.1 Concluding Observations

Spreadsheets are the most used programming environment in the world. However, the notorious agility and flexibility of the spreadsheet comes with its problems as well. They still lack many of the modern tools and features that modern programming environments offer, in particular the lack of control and validation makes spreadsheets prone to error. We chose to alleviate this problem by bringing version control, a tool that's widely used in the programming world, to spreadsheet end users in the form of SheetGit.

SheetGit functions as an integrated add-in for Excel and aids the users by providing various functionalities of version control, such as automated version creation, collaboration with other users, version comparison, and uniting two versions together in one spreadsheet. All of this directly in Excel, interacting with the spreadsheet itself, all in a graphical and intuitive manner.

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

It must be noted that both version control and spreadsheets have an extremely large list of features that just could not be implemented over the length of the work, but could still be integrated together in future versions of SheetGit, as we detail in the next section.

6.2 Future Work

SheetGit, the add-in itself, can still be improved in many ways:

1. Version control still has some features which were not included in SheetGit that can still be integrated into the spreadsheet world, such as *cherry picking*, *rebase* and many others. But careful consideration must be put into these 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.
2. SheetGit could always detect more types of Excel changes, such as cell validation and Visual Basic code, which will be addressed in future work.
3. While version messages, tags and automated version pruning were initially intended to be implemented for this version, the development of these was stalled so other important features could be finished. They are still important features that could help in the understanding and navigation on the version tree.

BIBLIOGRAPHY

- [1] *Atlassian Bitbucket*. URL: <https://www.bitbucket.org> (visited on 09/20/2016).
- [2] L. Bradley and K. McDaid. “Using Bayesian statistical methods to determine the level of error in large spreadsheets.” In: *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*. IEEE. 2009, pp. 351–354.
- [3] T. D. Cook, D. T. Campbell, and A. Day. *Quasi-experimentation: Design & analysis issues for field settings*. Vol. 351. Houghton Mifflin Boston, 1979.
- [4] Deloitte. *Spreadsheet Management: Not what you figured*. 2009. URL: <http://www2.deloitte.com/us/en/pages/audit/articles/spreadsheet-management.html>.
- [5] 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.
- [6] M. Fisher and G. Rothermel. “The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms”. In: *ACM SIGSOFT Software Engineering Notes*. Vol. 30. 4. ACM. 2005, pp. 1–5.
- [7] P. Fitzpatrick. *Diff formats in Coopy*. URL: http://share.find.coop/doc/patch_format.html (visited on 07/29/2016).
- [8] P. Fitzpatrick. *Specification of the highlighter diff format*. URL: http://share.find.coop/doc/spec_hilite.html (visited on 01/29/2016).
- [9] P. Fitzpatrick. *Coopy*. 2016. URL: <http://share.find.coop/doc/index.html> (visited on 01/29/2016).
- [10] P. Fitzpatrick. *Coopy’s ReadMe document*. 2016. URL: <https://github.com/paulfitz/coopy/blob/master/README.md> (visited on 01/29/2016).
- [11] P. Fitzpatrick and J. Panico. *Diff formats in Coopy*. 2011. URL: http://share.find.coop/doc/tdiff_spec_draft.html (visited on 07/29/2016).
- [12] S. Gandel. *Damn Excel! How the ‘most important software application of all time’ is ruining the world*. 2013. URL: <http://fortune.com/2013/04/17/damn-excel-how-the-most-important-software-application-of-all-time-is-ruining-the-world/> (visited on 01/18/2016).

BIBLIOGRAPHY

- [13] *Git*. URL: <https://git-scm.com/> (visited on 02/05/2016).
- [14] *Git Branching - Basic Branching and Merging*. URL: <https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging> (visited on 10/11/2016).
- [15] *Gitgraph.js*. URL: <https://github.com/nicoespeon/gitgraph.js> (visited on 02/07/2016).
- [16] *Google Sheets*. 2016. URL: <https://www.google.com/sheets/about/> (visited on 01/29/2016).
- [17] F. Hermans and E. Murphy-Hill. “Enron’s Spreadsheets and Related Emails: A Dataset and Analysis”. In: *Proceedings of the 37th International Conference on Software Engineering - Volume 2*. ICSE ’15. Florence, Italy: IEEE Press, 2015, pp. 7–16. URL: <http://dl.acm.org/citation.cfm?id=2819009.2819013>.
- [18] F. F. J. Hermans. “Analyzing and Visualizing Spreadsheets”. PhD thesis. 2013.
- [19] *JSON Hyper-Schema: Hypertext definitions for JSON Schema*. URL: <https://tools.ietf.org/html/draft-luff-json-hyper-schema-00> (visited on 09/20/2016).
- [20] *JSON Schema: core definitions and terminology*. URL: <https://tools.ietf.org/html/draft-zyp-json-schema-04> (visited on 09/20/2016).
- [21] *JSON Schema: interactive and non interactive validation*. URL: <http://tools.ietf.org/html/draft-fge-json-schema-validation-00> (visited on 09/20/2016).
- [22] S. Khanna, K. Kunal, and B. C. Pierce. “A formal investigation of diff3”. In: *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer. 2007, pp. 485–496.
- [23] B. Kitchenham, L. Madeyski, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong. “Robust Statistical Methods for Empirical Software Engineering”. In: ().
- [24] Knowledge@Wharton. *Rivals Set Their Sights on Microsoft Office: Can They Topple the Giant? - Knowledge@Wharton*. 2007. URL: <http://knowledge.wharton.upenn.edu/article/rivals-set-their-sights-on-microsoft-office-can-they-topple-the-giant/> (visited on 01/20/2016).
- [25] S. K. Kuttal, A. Sarma, and G. Rothermel. “On the Benefits of Providing Versioning Support for End Users: An Empirical Study”. In: *ACM Trans. Comput.-Hum. Interact.* 21.2 (Feb. 2014), 9:1–9:43. ISSN: 1073-0516. DOI: 10.1145/2560016. URL: <http://doi.acm.org/10.1145/2560016>.
- [26] L. Mitchell. *You’re Not Using Source Control? Read This!* 2014. URL: <http://www.lornajane.net/wp-content/uploads/2013/01/source-control-whitepaper-v1.1.pdf> (visited on 01/11/2016).
- [27] *Microsoft Excel*. URL: <https://products.office.com/en/excel> (visited on 01/29/2016).

-
- [28] R. Moreira. “SheetGit: A Tool for Collaborative Spreadsheet Development”. In: *Software Engineering Methods in Spreadsheets 2016*. To appear co-located with STAF 2016.
- [29] R. R. Panko. “What we know about spreadsheet errors”. In: *Journal of Organizational and End User Computing (JOEUC)* 10.2 (1998), pp. 15–21.
- [30] R. Panko and J. Halverson R.P. “Spreadsheets on trial: a survey of research on spreadsheet risks”. In: *Proceedings of HICSS-29: 29th Hawaii International Conference on System Sciences* 2 (1996), pp. 326–335. DOI: 10.1109/HICSS.1996.495416.
- [31] Pathio. 2016. URL: <http://www.pathio.com/> (visited on 09/12/2016).
- [32] C. Scaffidi, M. Shaw, and B. Myers. “Estimating the numbers of end users and end user programmers”. In: *Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on*. 2005, pp. 207–214. DOI: 10.1109/VLHCC.2005.34.
- [33] SheetGit. 2016. URL: <http://spreadsheetsunl.github.io/sheetgit/>.
- [34] Spreadsheet Compare. URL: <https://support.office.com/en-us/article/Overview-of-Spreadsheet-Compare-13fafa61-62aa-451b-8674-242ce5f2c986?ui=en-US&rs=en-US&ad=US> (visited on 07/29/2016).
- [35] Subversion. URL: <https://subversion.apache.org/> (visited on 08/05/2016).
- [36] XLTools. 2016. URL: <https://xltools.net/> (visited on 01/30/2016).
- [37] XLTools’ Version Control for Excel Spreadsheets. 2016. URL: <https://xltools.net/excel-version-control/> (visited on 01/30/2016).
- [38] ZeusDB. *What is Spreadsheet Risk?* 2014. URL: <https://www.zeusdb.com/blog/what-is-spreadsheet-risk/> (visited on 01/18/2016).

A P P E N D I X



PRE-QUESTIONNAIRE

Questionário Pré-Sessão

Este questionário tem como objectivo seleccionar pessoas com alguma experiência no uso da ferramenta Excel e averiguar se têm alguma experiência com controlo de versões

1. Em que ano está inscrito?:

1º Ano

2º Ano

Outro: _____

2. Sexo:

Masculino

Feminino

3. Idade:

<20

20-22

23-25

>25

Curso: _____

4. Já trabalhou em alguma ferramenta de edição de folhas de cálculo? (Exemplo: Microsoft Excel, LibreOffice, OpenOffice, etc.)

Sim

Não

5. No contexto de folhas de cálculo, saberia fazer o somatório de diferentes células?

Sim

Não

6. Na fórmula seguinte, qual contém apenas referências relativas?

=G4+D13

=\$G\$4+\$D\$13

Não sei o que são referências relativas.

7. Já programou anteriormente? (Em linguagens como C, Python, PHP, Java, etc)

Sim

Não

8. Caso a resposta anterior tenha sido sim, já usou controlo de versões nos seus programas? (Exemplo: Git, Subversion, Fossil, etc)

Sim

Não

Só deve responder às perguntas seguintes caso tenha respondido 'Sim' à pergunta anterior. Caso contrário, o questionário termina aqui.

9. Já trabalhou em simultâneo com outras pessoas utilizando controlo de versões?

Sim

Não

10. Qual é o resultado de efetuar 'Push' de uma versão?

O repositório local é atualizado com as mudanças novas.

O repositório remoto é atualizado com as mudanças novas.

Ambos os repositórios são atualizados devido à ação.

11. Qual dos seguintes comandos serve o mesmo objetivo que o comando 'Rebase'?

Checkout

Merge

Reset

12. Escreva o seu email da faculdade para ser notificado caso ganhe o sorteio.

Email: _____

APPENDIX



TUTORIAL FOR SPREADSHEET COMPARE

Tutorial - Spreadsheet Compare

Introdução

No contexto desta investigação poderá utilizar uma ferramenta, designada *Spreadsheet Compare*, criada pela Microsoft para efetuar comparações entre folhas de cálculo.

Neste estudo, pretendemos analisar até que ponto é vantajoso utilizar o *Spreadsheet Compare* para um número de tarefas em prol de outras alternativas.

Se tiver alguma dúvida durante este tutorial, por favor diga ao supervisor, para que o mesmo o possa esclarecer.

Por favor abra as folhas de cálculo **Southpoint1**, **Southpoint2** e **Southpoint3** disponíveis na pasta **SC** → **Tutorial** → **Pergunta 1**. Estes três ficheiros são três versões da mesma folha de cálculo. Sendo a '*Southpoint1*' a mais antiga, e a '*Southpoint3*' a mais recente.

Escolher as folhas de cálculo

Clique no ícone **Spreadsheet Compare** no Ambiente de Trabalho, e siga os seguintes passos:

1. Clique no botão *Compare Files* no canto superior esquerdo, como é possível ver na Figura 1.
2. Clique no ícone da pasta mais acima, ao lado de **Compare**, e selecione **Southpoint2**
3. Clique no ícone da pasta mais abaixo, ao lado de **To**, e selecione **Southpoint3**
4. Clicar no botão **OK**.

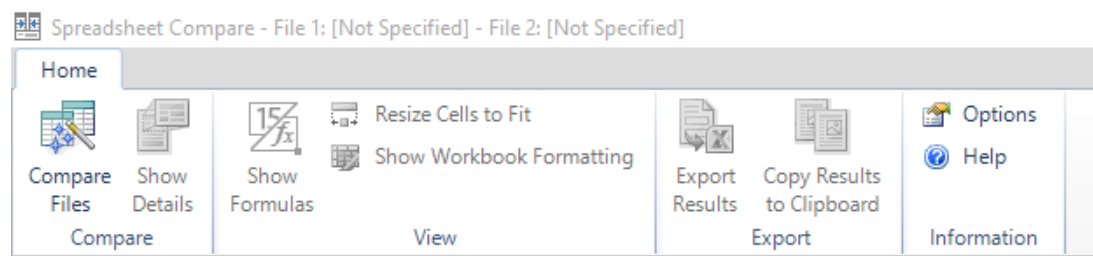


Figure 1: Botões da ferramenta

Visualizar diferenças

O programa coloca as duas folhas lado a lado com as suas diferenças visíveis ao utilizador. As células com diferenças são as que têm uma cor diferente do branco. Cada cor tem um significado diferente, como uma mudança de apenas texto ou uma mudança de fórmula. Na Figura 2 podemos ver que a verde escuro são células cujos valores foram mudados, o azul turquesa refere-se a células com fórmulas cujo valor final foi alterado. A legenda está sempre visível no canto inferior esquerdo, como pode ver na Figura 3.

134	124	153	411	6.5	2 672
159	166	151	476	6.5	3 094
164	146	150	460	6.5	2 990
96	109	146	351	6.5	2 282
130	133	148	411	6.5	2 672
126	141	154	421	6.5	2 737
145	107	150	402	6.5	2 613
					61 341

Figure 2: A aparência de uma folha de cálculo no *Spreadsheet Compare*

Enable	Option
<input checked="" type="checkbox"/>	Calculated Values
<input checked="" type="checkbox"/>	Formulas
<input checked="" type="checkbox"/>	SysGen Formulas
<input checked="" type="checkbox"/>	SysGen Formulas Errors
<input checked="" type="checkbox"/>	Structural
<input checked="" type="checkbox"/>	Names
<input checked="" type="checkbox"/>	SysGen Names
<input checked="" type="checkbox"/>	SysGen Names Error
<input checked="" type="checkbox"/>	Macros
<input checked="" type="checkbox"/>	Data Connection
<input checked="" type="checkbox"/>	Cell Format
<input checked="" type="checkbox"/>	Cell Protection
<input checked="" type="checkbox"/>	Sheet/Workbook Protection

Figure 3: Tipos de mudança suportados pela aplicação

Mostrar fórmulas

Por defeito, as folhas de cálculo apresentadas no *Spreadsheet Compare* mostram os resultados finais das fórmulas nas células, tal como o Microsoft Excel. É possível ver as fórmulas das células clicando no botão *Show Formulas*, como aparece em Figura 1, que faz as células apresentar as suas fórmulas em vez do resultado final. Se clicar no botão novamente, a aplicação volta a mostrar os resultados finais.

Perguntas

Para responder à pergunta seguinte, use as mesmas folhas de cálculo que utilizou anteriormente.

Pergunta 1) Assuma que está na versão mais recente da folha de cálculo, a Southpoint3, e o seu patrão diz-lhe que parece existe um erro na folha de cálculo, mas lembra-se que já tinha sido corrigido anteriormente. Procure e corrija o erro.

Início:

Fim:

Responda à pergunta seguindo os seguintes passos.

1. Escreva a hora, minutos e segundos atual à frente do campo **Início** acima. Pode ver a hora no seu computador.
2. É necessário encontrar a versão antiga com o erro corrigido.
3. Abra o **Spreadsheet Compare** e selecione as folhas de cálculo **Southpoint2** e **Southpoint3**. Isto para verificar se o erro ocorreu entre as duas versões.
4. Verifique se existe um erro nas diferenças uma a uma. Não existe nenhum erro óbvio.
5. Abra o **Spreadsheet Compare** e selecione as folhas de cálculo **Southpoint1** e **Southpoint2**. Talvez o erro esteja entre estas versões.
6. Verifique se existe um erro nas diferenças uma a uma. Na **célula I5** a fórmula foi alterada entre a **Southpoint1** e a **Southpoint2**, e está errada. A fórmula está a calcular sobre os valores da linha de baixo, e não da sua própria linha.
7. Abra a **Southpoint3** no **Microsoft Excel**, altere a célula I5 para o valor que encontrou na **Southpoint1**. Nós queremos o erro corrigido na versão mais recente, caso contrário perdemos toda a informação entre a **Southpoint1** e **Southpoint3**.
8. Guarde a folha de cálculo com o nome **Southpoint4**.
9. Escreva a hora, minutos e segundos atual à frente do campo **Fim** acima. Pode ver a hora no seu computador.
10. Feche quaisquer janelas do **Spreadsheet Compare** e **Microsoft Excel** que tenha abertas.

Na próxima pergunta, utilize as folhas de cálculo dentro da pasta **Pergunta 2**

Pergunta 2) Assuma que esteve a trabalhar sobre a folha de cálculo Southpoint4, e como resultado criou uma nova versão, a Southpoint5B, mas o seu colega disse-lhe que também esteve a trabalhar sobre a mesma folha e tem agora uma folha denominada Southpoint5A.

Junte o seu trabalho com o do seu colega, numa folha de cálculo só. Em caso de dúvida, utilize os valores da sua folha de cálculo.

Início: Fim:

Responda à pergunta seguindo os seguintes passos.

1. Escreva a hora, minutos e segundos atual à frente do campo **Início** acima. Pode ver a hora no seu computador.
2. Abra os ficheiros **Southpoint5A** e **Southpoint5B** no **Spreadsheet Compare**.
3. Abra o **Southpoint5B** e **Southpoint4** no Microsoft Excel. Vamos usar o **Southpoint5B** como base para unir as mudanças do **Southpoint5A**. Necessitamos do **Southpoint4** também aberto para passos seguintes.
4. No **Spreadsheet Compare**, note que o seu colega adicionou a **Linha 30** à **Southpoint5A**. Como nada foi adicionado por si nesse local na **Southpoint5B**, em princípio podemos afirmar que é uma mudança nova do seu colega.
5. Mas a linha pode já ter existido no **Southpoint4**. Veja no **Microsoft Excel** se a linha existia originalmente. Se sim, então não foi alterada pelo seu colega, foi apagada por si. Caso contrário foi mesmo adicionada pelo seu colega.
6. Neste caso a linha não existia originalmente, logo copie a **Linha 30** da **Southpoint5A** para a **Southpoint5B** no Microsoft Excel.
7. Existe outra diferença, a **célula D19** tem uma diferença entre a **Southpoint5B** e a **Southpoint5A**.
8. Veja o estado original da célula no **Southpoint4**. Como está igual à versão **Southpoint5A**, então a célula foi alterada só por si. Como estamos a usar a sua folha como base, não é necessário fazer alterações. Pode ver este caso ilustrado na Figura 4.
9. Existe uma última diferença, a **célula E20**. Note que neste caso, nenhum dos valores das novas folhas de cálculo é equivalente à do **Southpoint4**.
10. Podemos concluir que a célula foi alterada em ambas as versões. Como a pergunta menciona que em caso de dúvida, para utilizar os seus próprios dados, não necessita de fazer alterações. Se o enunciado dissesse o contrário, teria de copiar a célula para a sua folha de cálculo. Pode ver este caso ilustrado na Figura 5.
11. Guarde a **Southpoint5B**, com estas novas alterações, como uma nova folha de cálculo com o nome **Southpoint6**.
12. Escreva a hora, minutos e segundos atual à frente do campo **Fim** acima. Pode ver a hora no seu computador.

-
13. Feche quaisquer janelas do **Spreadsheet Compare** e **Microsoft Excel** que tenha abertas.

Célula D19

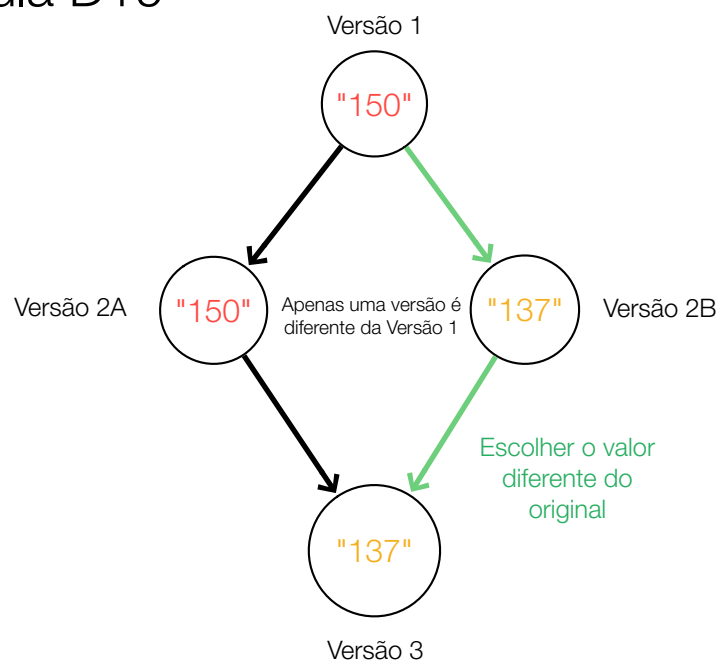


Figure 4: Resolução de conflitos na célula D19

Célula E20

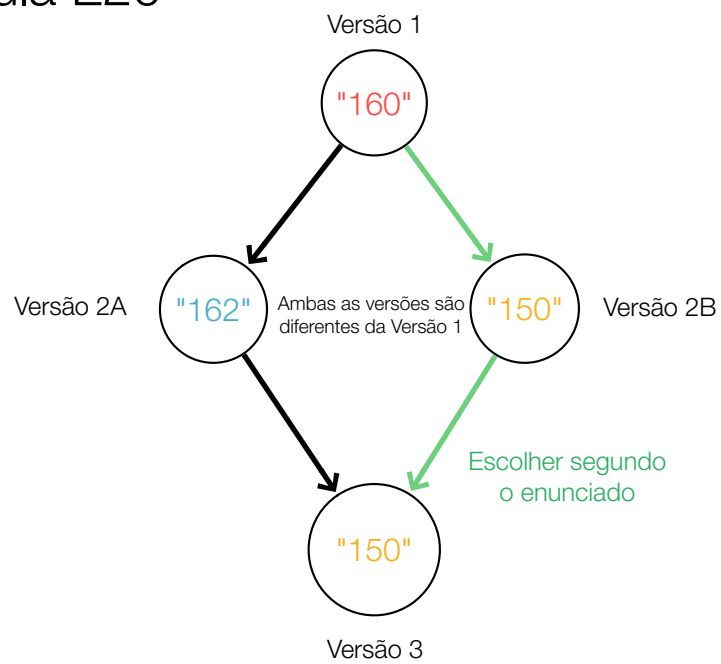


Figure 5: Resolução de conflitos na célula E20

A P P E N D I X



TASKS FOR SPREADSHEET COMPARE

Para as perguntas seguintes, utilize as folhas de cálculo na pasta SC → Grades

Perguntas da folha de cálculo Grades

1. Para esta pergunta utilize as folhas dentro da pasta Pergunta 1.
Assuma que é um professor e que criou uma folha de cálculo para calcular as notas dos seus alunos. De repente notou que uma das notas tem um erro. Procure o erro e corrija-o numa folha de cálculo chamada 'Grades7'

A folha Grades1 é a versão mais antiga, enquanto a Grades6 é a mais recente.

Início: Fim:

2. Para esta pergunta utilize as folhas dentro da pasta Pergunta 2.
Assuma novamente que é um professor e que alterou a pauta das notas, mas outro professor da mesma cadeira efetuou alterações ao mesmo tempo que você. Junte as alterações do seu colega com as suas numa folha de cálculo chamada 'Grades3'. Em caso de dúvida, os valores da sua folha devem tomar precedência.

A folha Grades1 é a versão base, enquanto a GradesA pertence ao cliente e a GradesB a si.

Início: Fim:

Para as perguntas seguintes, utilize as folhas de cálculo na pasta SC → Markets.

Perguntas da folha de cálculo Markets

1. Para esta pergunta utilize as folhas dentro da pasta Pergunta 1.
Criou e entregou uma folha de cálculo as finanças de uma empresa. No entanto, o seu cliente disse que existem erros pois os resultados não estão certos com os cálculos do lado dele, mas que em versões anteriores estavam corretos. Sabe também que o erro está especificamente numa fórmula. Procure o erro e corrija-o numa folha de cálculo chamada 'Markets7'

A folha Markets1 é a versão mais antiga, enquanto a Markets6 é a mais recente.

Início: Fim:

2. Para esta pergunta utilize as folhas dentro da pasta Pergunta 2.
Esteve a criar uma nova versão da folha de cálculo, mas o seu cliente acabou de-lhe enviar uma versão nova com alterações. Junte as mudanças feitas pelo seu cliente com as da sua nova versão numa folha de cálculo nova chamada 'Markets3'. Em caso de conflito, utilize os dados da sua versão para os resolver.

A folha Markets1 é a versão base, enquanto a MarketsA é a do cliente e a MarketsA é a sua.

Início: Fim:

**Feche o Microsoft Excel após terminar.
Obrigado pela sua participação :)**

A P P E N D I X



TUTORIAL FOR SHEETGIT

Tutorial - SheetGit

Introdução

No contexto desta investigação produzimos uma ferramenta, designada *SheetGit*, que procura criar e gerir versões de folhas de cálculo.

Na prática, a ferramenta cria versões da folha de cálculo enquanto trabalha, podendo posteriormente regressar a versões antigas, efetuar comparações e até partilhar o seu trabalho com colegas de forma segura e automática.

Neste estudo, pretendemos analisar até que ponto é vantajoso utilizar a ferramenta que nós desenvolvemos em alguns tipos de tarefa e averiguar se a nossa interface é fácil de compreender e utilizar.

Se tiver alguma dúvida durante este tutorial, por favor diga ao supervisor, para que o mesmo o possa esclarecer.

Funcionalidades do SheetGit

Esta secção irá explicar algumas das funcionalidades da aplicação. Por favor abra a pasta **SG** → **Tutorial** → **Pergunta 1** e finalmente o ficheiro no interior usando o **Microsoft Excel**.

Árvore de versões

No lado direito da aplicação irá ver um painel novo, dedicado ao *SheetGit*, e em particular um gráfico igual à Figura 1. A figura representa a lista de versões guardadas pela aplicação. O ponto mais acima é a **versão mais antiga** enquanto o ponto mais abaixo é a **versão mais recente**, como também pode ver pelo texto ao seu lado, *V1*, *V2*, *V3* significam Versão 1, 2 e 3. Este gráfico é automaticamente atualizado consoante o *SheetGit* crie versões.

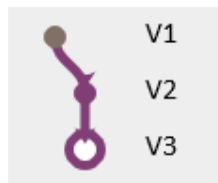


Figura 1: Árvore de versões do ficheiro *Southpoint*

Mudança de versões

É possível mudar entre as várias versões sem ter de fechar o Excel. Clique no ponto ao lado do texto **V2** na árvore.

A folha de Excel muda imediatamente para como estava na altura da versão 2. O ponto branco, agora situado no V2, mostra qual é a versão atual em que nos situamos.

Visualizar diferenças

A nossa aplicação permite ver as diferenças entre versões. Clique no botão **Comparison Mode** seguido do ponto **V3**.

Agora está no modo de comparação. O *slider* começa na posição mais acima, que se refere à versão **V2**, a atual. Conforme este é movimentado para baixo, a folha de cálculo transforma-se na versão **V3**, porque cada passo do *slider* refere-se a uma diferença entre as duas versões. Quando o *slider* chega ao ponto mais baixo, o que tem o valor 0 ao lado direito, a folha de cálculo estará igual à versão **V3**.

Agora clique no botão **Exit Comparison**, depois em **Normal Mode** seguido do ponto **V3** e passe à secção seguinte de perguntas.

Perguntas

Pergunta 1) Suponha que o seu patrão mandou-lhe um email a dizer que parece existir um erro na versão mais recente da folha de cálculo Southpoint, mas lembra-se que este erro já tinha sido corrigido anteriormente. Procure e corrija o erro.

Início: Fim:

Responda à pergunta seguindo os seguintes passos.

1. Escreva a hora, minutos e segundos atual à frente do campo **Início** acima. Pode ver a hora no seu computador.
2. É necessário encontrar a versão antiga com o erro corrigido.
3. Clique no botão **Comparison Mode**.
4. Clique no ponto da versão **V2**.
5. Movimente o *slider* para baixo ou clique no botão com a seta para baixo até chegar ao ponto 0, vendo as diferenças entre as duas versões. Não existe nenhum valor suspeito entre estas duas versões.
6. Clique no botão **Exit Comparison**, depois em **Normal Mode** seguido do ponto **V2**.
7. Clique novamente no botão **Comparison Mode** seguido do ponto **V1**.
8. Movimente novamente o *slider* para baixo e veja as diferenças que surgem. Na **célula I5** a fórmula está incorreta e após o *slider* passar essa célula, o valor é corrigido. Logo o valor está errado em **V2** mas estava correto em **V1**.
9. Clique no botão **Exit Comparison**, depois em **Normal Mode** seguido do ponto **V1**.
10. Clique na **célula I5** e depois seleccione e copie a fórmula no topo do Excel, não copie a célula diretamente.
11. Clique no ponto **V3**, a versão mais recente.
12. Cole a fórmula correta na **célula I5**. Note que apareceu uma versão nova, a **V4**.
13. Escreva a hora, minutos e segundos atual à frente do campo **Fim** acima. Pode ver a hora no seu computador.
14. Guarde o ficheiro e feche o Microsoft Excel.

Ramos da árvore

Quando um utilizador faz alterações, a aplicação coloca estas em ramos diferentes, impercetíveis a outros utilizadores. Na Figura 2, temos o tronco cinzento e um ramo a roxo.

O ramo a cinzento, o mais à esquerda denominamos de *Trunk* (significa 'tronco' em português) e é um ramo especial porque é um ramo partilhado com outros utilizadores da mesma folha de cálculo.

No fundo, os utilizadores fazem alterações em ramos pequenos alternativos e quando estiverem prontas a mostrar a outros, estas são colocadas no *trunk*, tal como na Figura 2.

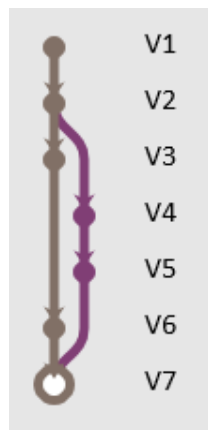


Figura 2: As mudanças do ramo vermelho a serem integradas no *trunk*.

Pergunta 2) Assuma que esteve a trabalhar sobre a folha de cálculo SouthpointB, e como resultado criou uma nova versão, mas o seu colega disse-lhe que também esteve a trabalhar sobre a mesma folha e tem agora a sua própria versão.

Junte o seu trabalho com o do seu colega, numa folha de cálculo só. Em caso de dúvida, utilize os valores da folha de cálculo do seu colega.

Início: Fim:

Responda à pergunta seguindo os seguintes passos.

1. Escreva a hora, minutos e segundos atual à frente do campo **Início** acima. Pode ver a hora no seu computador.
2. Abra o ficheiro **Southpoint** dentro da pasta **Pergunta 2**.
3. Note que existe no gráfico da árvore duas cores, o tronco cinzento e um ramo roxo. A versão **V5** do ramo roxo pertence-lhe enquanto o **V6** cinzento pertence ao seu colega pois está no tronco.
4. Clique no ponto da versão **V5**.
5. Clique no botão **Place versions in trunk**. Este botão serve para colocar as mudanças do ramo roxo no tronco partilhado por todos.

6. Note que apareceu a **Linha 30**, feita pelo seu colega. Mas como a sua versão não tinha nada lá, foi automaticamente colocada.
7. A célula **E20** contém um conflito pois ambas as versões **V5 e V6** alteraram esse campo. Clique na célula com o conflito e selecione no *dropdown* a escolha que começa por **T:**, que significa *trunk*. Isto porque o enunciado pede que em caso de dúvida para escolher as alterações do colega.
8. Escreva a hora, minutos e segundos atual à frente do campo **Fim** acima. Pode ver a hora no seu computador.
9. Salve a folha de cálculo e encerre o Microsoft Excel.

A P P E N D I X



TASKS FOR SHEETGIT

Para as perguntas seguintes, utilize a folha de cálculo na pasta SG → Grades.

Perguntas da folha de cálculo Grades

1. Abra a folha de cálculo Grades dentro da pasta Pergunta 1.
Assuma que é um professor e que criou uma folha de cálculo para calcular as notas dos seus alunos. De repente notou que uma das notas tem um valor claramente errado. Procure o erro e corrija-o.
Início: Fim:
Encerre o Microsoft Excel antes de continuar.
2. Abra a folha de cálculo Grades dentro da pasta Pergunta 2.
Assuma novamente que é um professor e que alterou a pauta das notas, mas outro professor da mesma cadeira efetuou alterações ao mesmo tempo que você. Junte as alterações do seu colega com as suas numa folha de cálculo só. Em caso de dúvida, os valores da sua folha devem tomar precedência.
Início: Fim:

Feche o Microsoft Excel após concluir.

Para as perguntas seguintes, utilize a folha de cálculo na pasta SG → Markets.

Perguntas da folha de cálculo Markets

1. Abra a folha de cálculo Markets dentro da pasta Pergunta 1.
Criou uma folha de cálculo sobre as finanças de uma empresa. No entanto, o seu cliente disse que existem erros pois os resultados não estão certos com os cálculos do lado dele, mas que em versões anteriores estavam corretos. Sabe também que o erro está especificamente numa fórmula. Procure o erro e corrija-o.

Encerre o Microsoft Excel antes de continuar.

Início: Fim:

2. Abra a folha de cálculo Markets dentro da pasta Pergunta 2.
Esteve a criar uma nova versão da folha de cálculo que criou para o seu cliente, mas ele acabou de lhe enviar uma versão nova com alterações. Junte as mudanças feitas pelo seu cliente com as da sua nova versão. Em caso de conflito, utilize os dados da sua versão para os resolver.

Início: Fim:

**Feche o Microsoft Excel após terminar.
Obrigado pela sua participação :)**



POST-QUESTIONNAIRE

Questionário Pós-Sessão

1. Selecione a resposta que corresponde ao quanto concorda ou discorda com as seguintes frases.
 - (a) Estou confiante que respondi corretamente a todas as tarefas da folha de cálculo **Markets**. (Selecione uma)
 - Concordo plenamente.
 - Concordo.
 - Nem concordo nem discordo.
 - Discordo.
 - Não se aplica.
 - (b) Estou confiante que respondi corretamente a todas as tarefas da folha de cálculo **Grades**. (Selecione uma)
 - Concordo plenamente.
 - Concordo.
 - Nem concordo nem discordo.
 - Discordo.
 - Não se aplica.
2. A ferramenta ajudou-o a desempenhar as tarefas propostas com maior facilidade?
 - Sim.
 - Não.
3. Pensa que a ferramenta deveria ajudar mais quando estava a desempenhar as tarefas?
 - Sim.
 - Não.
4. Pensa que era necessária a existência de uma ferramenta como a nossa?
 - Concordo plenamente.
 - Concordo.
 - Nem concordo nem discordo.
 - Discordo.
5. A interface da ferramenta é de fácil compreensão?
 - Sim.
 - Não.
6. O que acha da nossa ferramenta?

7. Sugestões de melhoria:



VERSION LIST JSON SCHEMA

```
1 {
2   "type": "object",
3   "$schema": "spreadsheetsunl.github.io/sheetgit",
4   "required": true,
5   "patternProperties": {
6     "^([a-zA-Z0-9])+$": {
7       "type": "object",
8       "required": false,
9       "properties": {
10        "author": {
11          "type": "object",
12          "required": true,
13          "properties": {
14            "email": {
15              "type": "string",
16              "required": true
17            },
18            "name": {
19              "type": "string",
20              "required": true
21            }
22          }
23        },
24        "branchChanges": {
25          "type": "object",
26          "required": false,
27          "patternProperties": {
28            "^(\\[A-Z]+\\[0-9]+(?:\:\\[A-Z]+\\[0-9]+)?)$": {
29              "type": "object",
30              "required": true,
31              "properties": {
```

APPENDIX G. VERSION LIST JSON SCHEMA

```

32         "Original": {
33             "type": "string",
34             "required": false
35         },
36         "ExcelDatatype": {
37             "type": "string",
38             "required": false,
39             "$ref": "#/definitions/datatypes"
40         }
41     },
42     "definitions": {
43         "datatypes": ["Value", "Formula"]
44     }
45 },
46 }
47 },
48 "branch": {
49     "type": "string",
50     "required": true
51 },
52 "changes": {
53     "type": "object",
54     "required": false,
55     "patternProperties": {
56         "^(\\$[A-Z]+\\$[0-9]+(?:\\:\\$[A-Z]+\\$[0-9]+)?)$": {
57             "type": "object",
58             "required": true,
59             "properties": {
60                 "Original": {
61                     "type": "string",
62                     "required": false
63                 },
64                 "ExcelDatatype": {
65                     "type": "string",
66                     "required": false,
67                     "$ref": "#/definitions/datatypes"
68                 }
69             },
70             "definitions": {
71                 "datatypes": ["Value", "Formula"]
72             }
73         },
74     }
75 },
76 "message": {
77     "type": "string",
78     "required": false
79 },
80 "parent": {
81     "type": "string",

```

```
82     "required": false
83   },
84   "timestamp": {
85     "type": "string",
86     "required": true
87   }
88 }
89 }
90 }
91 }
```