August 2016

A High Fidelity Interface for Documents Merging Tool Using a Language Analysis Oracle

Arwa Mohammed Alsubhi
University of Wisconsin-Milwaukee

A HIGH FIDELITY INTERFACE FOR DOCUMENTS MERGING TOOL USING A LANGUAGE ANALYSIS ORACLE

by

Arwa Alsubhi

A Thesis Submitted in
Partial Fulfillment of the
Requirements for the Degree of

Master of Science
in Computer Science

at

The University of Wisconsin-Milwaukee

August 2016

ABSTRACT

A HIGH FIDELITY INTERFACE FOR DOCUMENTS MERGING TOOL USING A LANGUAGE ANALYSIS ORACLE

by

Arwa Alsubhi

The University of Wisconsin-Milwaukee, 2016
Under the Supervision of Professor Ethan Munson

Revision is an important step in the writing process in order to obtain a good written work. It is mostly needed in academia, industry, and government. Usually, it is done by one reviser or more who is not the author of the written piece. The role of revisers is not limited to correcting any spelling or grammar mistakes, but also ensuring the coherence of the writing as well as the words used by the author to express his/her idea correctly to the readers. In addition, revisers help the author to put his/her writing in the appropriate format. One approach to do the revision is individually in a parallel way where each reviser modifies the original document. As a result, the author ends up with multiple versions of his/her work. For this situation, many merging control systems have been developed to enable the user to merge the revised versions with the original document in order to represent the changes that were made in the revised versions in an easily understandable way. Although these merging tools provide the users with much of the relevant information about the changes and who made them, the interfaces of these tools do not allow users to filter the corrections so that the users' attention can be focused on the most important changes. For example, if there are format changes and grammar corrections, in addition to editing changes that could change the meaning of the author's original writing, we believe that users would prefer to pay attention to the changes that could change the meaning and then check the format changes, after taking a look at grammar corrections.

In this thesis we developed a new merging interface that enables the user to filter the changes, based on their level of importance, to give them special attention. In addition, the interface provides the users with a user-friendly control panel that allows the user to choose among conflicting changes. This will help users produce a correct merged document.

A usability study was conducted with ten graduate students from the University of Wisconsin–Milwaukee to test whether a high fidelity prototype of this interface would help users to better understand the changes that were made in the two revisions as well as choose the best revisions. While the study found both positive and negative qualities in the prototype, most participants valued the change classification feature, suggesting that it is worthy of further research.

TABLE OF CONTENTS

LIST OF FIGURES

To my parents,

my husband,

and especially my daughters

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Ethan Munson for his guidance and support during the course of my research. I would like to extend my gratitude toward the committee members for taking time to assist by providing their feedback for this thesis.

Chapter 1 Introduction

1.1 Problem statement

Revision is an important step in the writing process in order to produce a well-written work. It is most commonly done in academia, industry, and government [1]. Usually, it is done by one or more revisers who are not necessarily the author of the written work. The role of revisers is not limited to correcting spelling or grammatical mistakes, but also includes ensuring that the writing is coherent and accurately expresses the author's idea to the readers. In addition, revisers help the author put his or her writing in the appropriate format.

Many approaches have been adapted to support collaborative revision. A round- robin method can be used by passing a Microsoft Word document with track changes by email to the next reviewer sequentially, until the writer is satisfied with the document. This system of sequential revision reduces change conflicts that might occur when two collaborators modify the same text in different ways. However, waiting for a collaborator to complete his or her revision is time-consuming for the team [2].

Web-based word processing such as Google Docs is another collaborative writing technique that allows contributors to work simultaneously on the same document. A co-writer can access the shared document any time from any computer with no need to install a particular application on his or her device. Also, working offline is an available option by installing a free application in devices of co-workers who can't access the Internet or have a slow or inconsistent connection, which makes online editing difficult. Contributors' works will be synchronized in the shared document when the network connection is available. Despite all of these features of synchronized collaborative writing, there are potential barriers that might make team members not want to use this method for their writing. Security is the most important issue related to cloud computing, because there is no guarantee of keeping confidential documents inaccessible to unauthorized users. Also, sometimes a shared document might be lost, in the case of data corruption or irretrievable loss on cloud servers, unless co-workers have kept a copy of the document in their computers, which is uncommon [3].

The third common method for cooperative revision is where revisers work individually in a parallel way, with each reviser modifying the original document to make a new version. Unlike the previous techniques, a reviewer's contribution is isolated from others, which can have a positive impact on the quality of writing. Displaying a reviser's changes to all team members could lead to drawing others' attention to these changes to make different modifications in the same place, rather than focusing on other parts of the text that need improvement [4]. Displaying changes could also lead revisers to avoiding touching others' changes that might need consideration, in order to maintain the social relationship [5]. Also, web-based and sequential approaches may deter the collaborator from contributing effectively to the revision process because of concerns about what other contributors think about his or her work [5]. For the reasons discussed above, our research focuses on the parallel revision technique.

With parallel revisions, the author ends up with multiple versions of his or her work. As a result, a number of merging systems have been developed to help the user merge the revised versions with the original document. These systems try to represent the changes that were made in each of the versions in an easily understandable way. Although these merging tools provide the users with much of the relevant information about the changes and who made them, there are gaps in what they provide. The user merges the multiple versions of a document either by using a combining feature that integrates into the word processor or by using external merging tools. By using the merging feature in a word processor such as MS-Word, the user needs multiple steps to complete merging more than two versions of a document. The first step is to combine the original version with the first revised versions and then accept or reject the changes. Second, the first step is done with the second revised version. Finally, the user can combine the two documents resulting from the previous steps. The main problem with this method is that there is no way to show the users the conflicting changes in both revised versions. In contrast, external merging software allows the user to merge up to three versions of a document and show the changes line by line, as well as the conflicting changes, but there is no way to let the user choose the best change in the case of a disagreement.

In addition, the interfaces of all existing merging software do not provide a way for the user to filter the corrections, based on their level of importance, in order to give primary attention to the most important changes. For example, if there are format changes and grammar corrections, in addition to sentence changes which could change the meaning of the author's original writing, the user would probably prefer to give attention to the changes that could change the meaning of the document. The author might choose to pay less attention to suggested format changes and grammar corrections.

1.2 Objective

The objective of this thesis is to experiment with a new merging interface that enables the user to filter document changes, based on their level of importance, in order to give special attention to the most important changes. In addition, the thesis will assess providing users with a user-friendly control panel that allows them to choose the best change in case of conflicts. This will help users produce a correct merged document. Overall, the research seeks to develop an interface that users will find more satisfying than existing interfaces for document merging.

1.3 The Oracle Approach

Much research has been done to improve the natural language processing software that helps people enhance their writing, especially when they write in a foreign language. A contextual spelling checker is implemented to detect words that are spelled correctly but that don't convey the writer's intended meaning. These words, called *homophones*, have the same sound but are spelled differently and have different meaning. An example is *buy*, *by*, and *bye*; these words sound the same, but they have different meanings. Similarly, a context-based grammar checker has been developed to identify grammar errors by considering the surrounding text. However, these tools still can't detect errors that human eyes can spot. In addition, no software exists that can easily help us analyze document versions as we envision them. As a result, we pretend that we have that system by assuming an "oracle" exists that can help us.

1.4 Key research activities

The first step in our research was to create sets of sample documents, where each set included three MS Office documents (one is the original version and the other two are revised versions), plus a text file that describes the hand coded changes that were made in the two revised versions. The second step was to implement a prototype that took these four files and displays the three versions beside a proposed merged file. The implemented interface has the following features:

- Allow the user to filter the changes that were made in the two revised versions into the following categories: font, spelling, grammar, and editing.
- Allow the users to locate the places of changes in two revised versions that were made in the original version.
- Allow the users to choose between the two different changes from revised versions that were made on the same place in the original document.
- Enable the users to save the merged file as HTML.

Finally, we conducted a user study to test the usability of the proposed interface to evaluate the proposed interface. Ten graduate students from the University of Wisconsin–Milwaukee participated in the study. The participants came from different cultures: five were Arabic, three were South East Asian, and two were domestic students. Tasks based on real-world scenarios adapted from collaborative writing in academia were presented in the study. The tasks focused on finding a specific type of modification that had been made to the original document based on our changes classifications and accepting/rejecting the correction. Positive and negative feedback were collected through observations and questionnaires, leading to ideas for future improvements of the interface and more research investigations.

1.5 Thesis Outlines

This thesis is organized as follows: Chapter 2 provides background on version control system, related works on changes classification and demonstration of selected existing merging software interfaces. Chapter 3 explains how we collected and analyzed requirements for the design of merging document interface prototype. Chapter 4 describes the details of the implementation of high fidelity merging document interface. Chapter 5 describes our usability test of this interface, and describes the interesting results that we found from participant experience of using this interface. Finally, Chapter 6 includes the conclusion and the future works.

Chapter 2 Background and Related Research

This chapter provides the background which motivated our research, along with a selection of important related works. The first section reviews some concepts in Version Control Systems. Then, we will review the related works in categorizing changes, as well as methods of representing them. Finally, a demonstration of interfacing with existing merging tools will be given.

2.1 Version Control System (VCS)

A Version Control System (VCS) is essential for any work that is performed collaboratively. It is designed to cope with multiple versions of a file by tracking the differences among them, checking who made the changes, and noting the time and date when the changes were made. There are two approaches of VCS[6][7]: centralized and distributed. In centralized version control, a single copy of a file is stored in a remote central repository, and available to be accessed by all coworkers. Any modifications on this central copy might be made by one member of the team, and will be stored and represented for other members. In contrast, distributed version control allows each collaborator to revise the file locally, on their devices, which contains versions history. Then, it detects the differences between all versions of the file and combines these changes in a single version. The major benefits of distributed version control over centralized are [8]:

- The contributor is able to work anywhere, even with the absence of Internet connection, while most centralized version control systems require Internet access to participate in the work.

- Since each team member has an updated version history, the presence of the entire history of the repository is guaranteed, even when catastrophic failure occurs to the remote server or any collaborators' devices.
- The absence of central authority led to increased contributions toward teamwork, where every coworker was able to submit his/her changes without permission, as with the centralized version control system.
- The need of a large amount of resources, such as storage space. Memory is reduced significantly by using a distributed version control system, unlike centralized version control, where large number of collaborators might work on the project using a central server.
- The participant in a software project feels more confident with proposing and experimenting with new features for the developed system, without fear of causing the whole project to fail. The modification will be done locally before they are transferred to the central server. This encourages the creative ideas within teamwork.
- Common operations, such as merging, will be done faster in the distributed system. They use local repositories, leading to a reduction of time, relative to what is needed to complete these operations on the central system.

Considering the significance of versioning control in team work, researchers work to either modify existing applications or design tools to support it. Munson and Thao [9] implemented a framework called the version-aware document, which preserves a complete version history of an XML document without the need to centralize or share storage. This is an extension of their previous work on efficient algorithms for merging changes in XML documents. Their system had an efficient representation of the differences among XML document versions, and provided simple GUI application to show changes and a conflict resolver if there were any conflicting modifications [10].

Later work by Thao’s group produced a version aware plug-in for Microsoft Word that maintains a complete history of document revisions and can display it as a graph. The tool tracks changes and who made them, and can merge the changes from two documents into one [11].

Other research looked at how to support versioning by doing some modifications on an open source office suite. The LibreOffice source code was modified to support the unique document element identifiers needed by the XML version aware model [9]. This effort was intended to permit the LibreOffice document system to preserve the whole version history, as well as provide full merging and differencing services [12].

2.2 Change Classification

Text changes, which were made in multiple versions of a file, have been categorized by using different taxonomies in order to serve different purposes. Many categories have been suggested, either for software source codes or textual edits.

It is very common in the software lifecycle to make multiple versions of the software, especially when the work is done collaboratively among team members. The issue of identifying the source that leads to the software failure is raised when more than one programmer works on the code. Many researchers have attempted to design tools to assist the collaborative programmers in their awareness of the changes made on software, and show which of them might be causing the errors.

Stoerzer et al [13] designed a classification tool, called JUnit/CIA, to categorize the changes made on Java programs. Based on the effects of these changes, which were responsible for a program failure, the tool marks the changes with the colors of red, yellow, or green. The red indicates that the change has a high probability of causing failures. The change marked with yellow may cause a problem, while the green sign represents a successful associated change. The aim of this classification is to help the programmers identify the changes (between two versions of the Java program) that caused the test failure.

Kim et al [14] proposed a software change classifier, based on machine-learning approaches, in order to determine whether the changes are a bug or clean change.

Fluri and Gall [15] implemented an Eclipse plugin, called CHANGEDISTILLER. It extracts the changes from two java source code files and represents them in a classified manner. The changes are represented by the level of significance, which may be low, medium, high, or crucial. The significance level of a change indicates the impact it would have on other code entities if it is accepted. Also, CHANGEDISTILLER determines whether a modification preserves or changes the function of the code.

In a document editing process, the most common categorization has been used widely in the field. Identifying the corrections made on the original document involves insertion, deletion, relocation, or conflict. When new entities are added to the original text, it will be labeled as insertion. When the opposite happens, the label will be deletion. The conflict category is raised when multiple versions are merged. It occurs when the text is modified in one version, while the same text is deleted from other versions, or when the same text is modified differently in each version [10] [16].

A Wikipedia article goes through multiple revisions, and each revised version has many differences from the original article. A classifier was developed based on a machine that learns to categorize the modifications made to the (English Language) Wikipedia article, into 21 complex classes. These 21 categories, grouped into three main categories, include surface changes, meaning changes, and Wikipedia policy. Spelling, grammar, paraphrasing, relocating and markup language entity edits are considered surface modifications. On the other hand, any change in the article's information, references, and templates is labeled as a meaning change modification. Also, each of these meanings classify the categories further, separating them into insertion, deletion, and modification. The last main classification is called Wikipedia policy because it refers to changes made because of system policies and includes vandalism and revert[17].

Zhang and Litman [18] [19] designed an automatic revision detector for argumentative writing done by high school students. This automatic detector classifies the detected revisions between two drafts of student writing to the reason of the editing (claim, evidence, rebuttal, etc.).

Recently Ping et al [20] proposed revision classifier to determine the significant of changes whether corrections changes the meaning of the original writing or just a paraphrasing.

2.3 Changes Representation

To differentiate the modifications made in revised versions, during the revision process from original entities, many techniques were adapted. Some are color-coding, symbol-coding, or graphical visualization.

The changed text is displayed by using different background or foreground colors, where each color indicates the type of change or certain editors [21].

To avoid overwhelming the user with a lot of changes made, especially in a large document, Zhang and Jagadish [22] presented a new way to represent the changes, which are made in plain text to the users. The changes that are relevant to a selected text from the source file will appear inside different brackets. The text inside '('and ')' is not changed. The inserted text will be located inside '{+' and '+}, while the deleted string appears inside '[-'and '-]'.

Whitaker [21] suggests using railroad or tramline diagrams to represent changes. Each track or line indicates modification from a revised version, when the same text is changed differently in other versions. He claims that using this method would help the user easily choose the best change, rather than comparing them mentally.

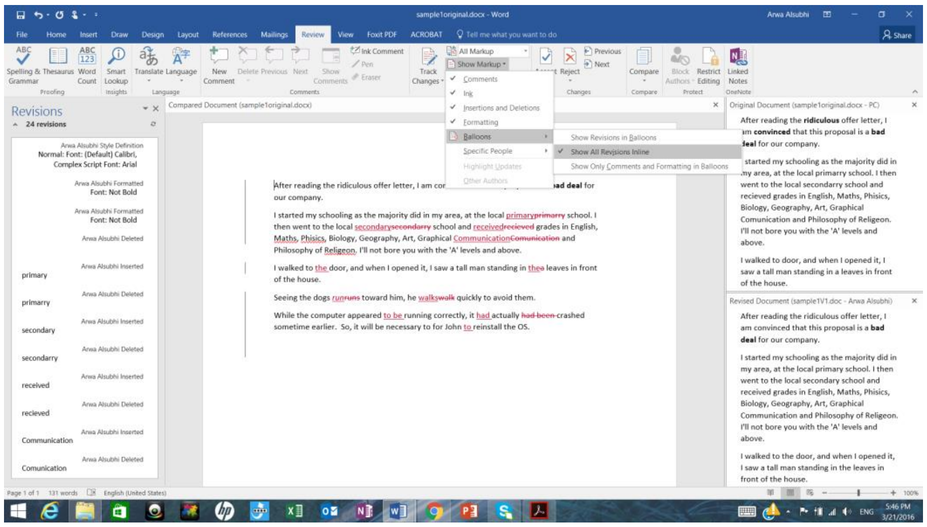
2.4 Interfaces of Existing Merging Software

This section gives a close look of existing merge tool interfaces and its features. The examples of commercial tools will be discussed first, then the open source software.

2.4.1 Commercial Software

2.4.1.1 Microsoft Word 2016

Microsoft Word 2016 allows users to merge two versions of a document. The two versions display in a pane, to the right of the merged document. All changes in the merged document appear in different text colors, except the format changes. There are two ways to show the revisions, either in balloons or inline (figure 1). If the balloon mode is chosen, the deleted text, as well as who made it, will appear in a balloon on the right margin of the merged document. Each balloon is connected to the corresponding inserted text, by line, while the inserted text appears underlined and with different colors in the merged document (figure 2-2). On the other hand, inline mode represents the deleted text with strikethrough, followed by underlined text. Both are represented with different text colors (figure 2-1). The user can get information about who made the changes, and when, in two ways. They can either hover on a change or use the revisions pane, which appears on the left side of the merged document. In addition, the user can accept or reject all changes made on the original document, or just accept or reject the individual change.



Revisions

24 revisions

Arwa Alsubhi Style Definition
Normal: Font: (Default) Calibri, Complex Script Font: Arial

Arwa Alsubhi Formatted
Font: Not Bold

Arwa Alsubhi Formatted
Font: Not Bold

Arwa Alsubhi Deleted

Arwa Alsubhi Inserted
primary

Arwa Alsubhi Deleted
primarry

Arwa Alsubhi Inserted
secondary

Arwa Alsubhi Deleted
secondarry

Arwa Alsubhi Inserted
received

Arwa Alsubhi Deleted
recieved

Arwa Alsubhi Inserted
Communication

Arwa Alsubhi Deleted
Comunication

Compared Document (sample1original.docx)

After reading the ridiculous offer letter, I am cor **ad deal** for our company.

I started my schooling as the majority did in my area, at the local primary~~primarry~~ school. I then went to the local secondary~~secondarry~~ school and received~~recieved~~ grades in English, Maths, Phisics, Biology, Geography, Art, Graphical Communication~~Comunication~~ and Philosophy of Religeon. I'll not bore you with the 'A' levels and above.

I walked to the door, and when I opened it, I saw a tall man standing in the~~a~~ leaves in front of the house.

Seeing the dogs *run*~~runs~~ toward him, he walks~~walk~~ quickly to avoid them.

While the computer appeared to be running correctly, it had actually ~~had been~~ crashed sometime earlier. So, it will be necessary to for John to reinstall the OS.

Original Document (sample1original.docx - PC)

After reading the **ridiculous** offer letter, I am **convinced** that this proposal is a **bad deal** for our company.

I started my schooling as the majority did in my area, at the local primarry school. I then went to the local secondarry school and recieved grades in English, Maths, Phisics, Biology, Geography, Art, Graphical Comunication and Philosophy of Religeon. I'll not bore you with the 'A' levels and above.

I walked to door, and when I opened it, I saw a tall man standing in a leaves in front of the house.

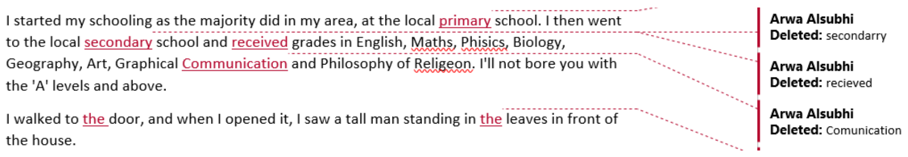
Revised Document (sample1V1.doc - Arwa Alsubhi)

After reading the ridiculous offer letter, I am convinced that this proposal is a **bad deal** for our company.

I started my schooling as the majority did in my area, at the local primary school. I then went to the local secondary school and received grades in English, Maths, Phisics, Biology, Geography, Art, Graphical Communication and Philosophy of Religeon. I'll not bore you with the 'A' levels and above.

I walked to the door, and when I opened it, I saw a tall man standing in the leaves in front of the house.

Figure 2-1: Microsoft Word displays changes in merged document when "In line" option is chosen



I started my schooling as the majority did in my area, at the local primary school. I then went to the local secondary school and received grades in English, Maths, Phisics, Biology, Geography, Art, Graphical Communication and Philosophy of Religeon. I'll not bore you with the 'A' levels and above.

I walked to the door, and when I opened it, I saw a tall man standing in the leaves in front of the house.

Arwa Alsubhi
Deleted: secondarry

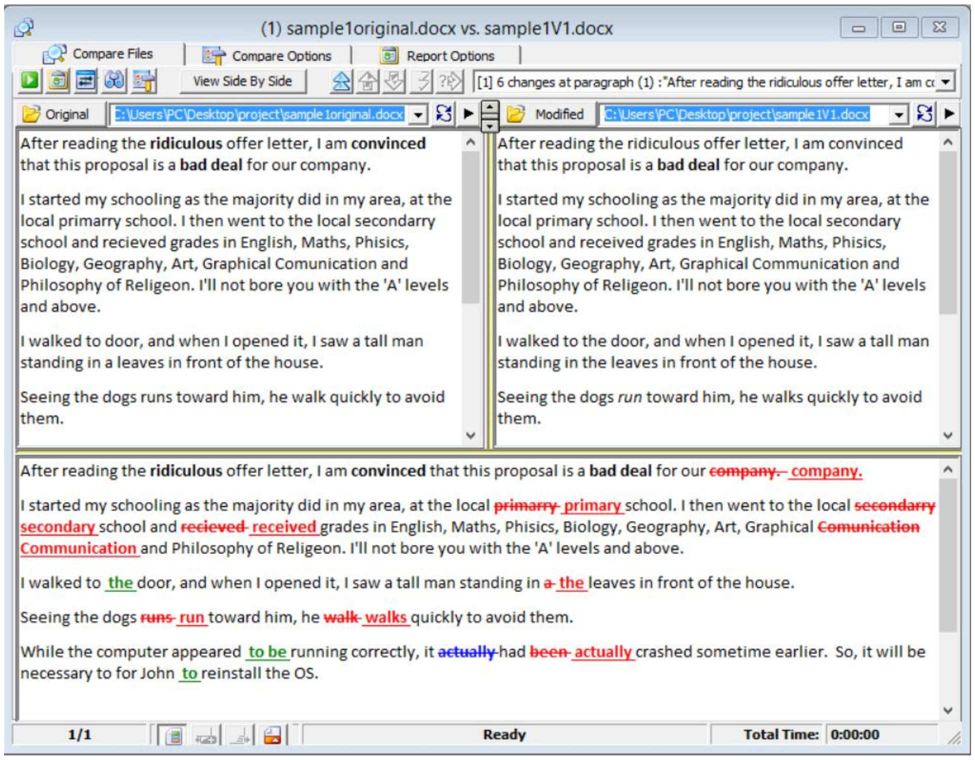
Arwa Alsubhi
Deleted: recieved

Arwa Alsubhi
Deleted: Comunication

Figure 2-2 Microsoft Word displays the changes in balloons

2.4.1.2 Diff Doc

Diff Doc [23] compares two versions of a document from various types, including Word, Excel, PowerPoint, pdf, text, html and xml. The two versions display side by side, and the merged document is below them. Changes in the merged file are distinguished by using different foreground colors. The red color, with strikethrough, represents the original modified text. Next to it is the modification in red, with underline. The green text indicates an insertion, while the blue with strikethrough represents a deleted text. In the tools, there is no support for accepting the changes, but the user can save the merged file. Unlike the MS Word comparison tool, there is no detection for format changes (figure 2-3).



(1) sample1original.docx vs. sample1V1.docx
Compare Files
Compare Options
Report Options
View Side By Side
[1] 6 changes at paragraph (1) :"After reading the ridiculous offer letter, I am co
Original
C:\Users\PC\Desktop\project\sample1original.docx
Modified
C:\Users\PC\Desktop\project\sample1V1.docx
1/1
Ready
Total Time: 0:00:00

After reading the **ridiculous** offer letter, I am **convinced** that this proposal is a **bad deal** for our company.

I started my schooling as the majority did in my area, at the local primarry school. I then went to the local secondarry school and recieved grades in English, Maths, Phisics, Biology, Geography, Art, Graphical Comunication and Philosophy of Religeon. I'll not bore you with the 'A' levels and above.

I walked to door, and when I opened it, I saw a tall man standing in a leaves in front of the house.

Seeing the dogs runs toward him, he walk quickly to avoid them.

After reading the ridiculous offer letter, I am convinced that this proposal is a **bad deal** for our company.

I started my schooling as the majority did in my area, at the local primary school. I then went to the local secondary school and received grades in English, Maths, Phisics, Biology, Geography, Art, Graphical Communication and Philosophy of Religeon. I'll not bore you with the 'A' levels and above.

I walked to the door, and when I opened it, I saw a tall man standing in the leaves in front of the house.

Seeing the dogs *run* toward him, he walks quickly to avoid them.

After reading the **ridiculous** offer letter, I am **convinced** that this proposal is a **bad deal** for our ~~company.~~ company.

I started my schooling as the majority did in my area, at the local ~~primarry~~ primary school. I then went to the local ~~secondarry~~ secondary school and ~~recieved~~ received grades in English, Maths, Phisics, Biology, Geography, Art, Graphical ~~Comunication~~ Communication and Philosophy of Religeon. I'll not bore you with the 'A' levels and above.

I walked to the door, and when I opened it, I saw a tall man standing in ~~a~~ the leaves in front of the house.

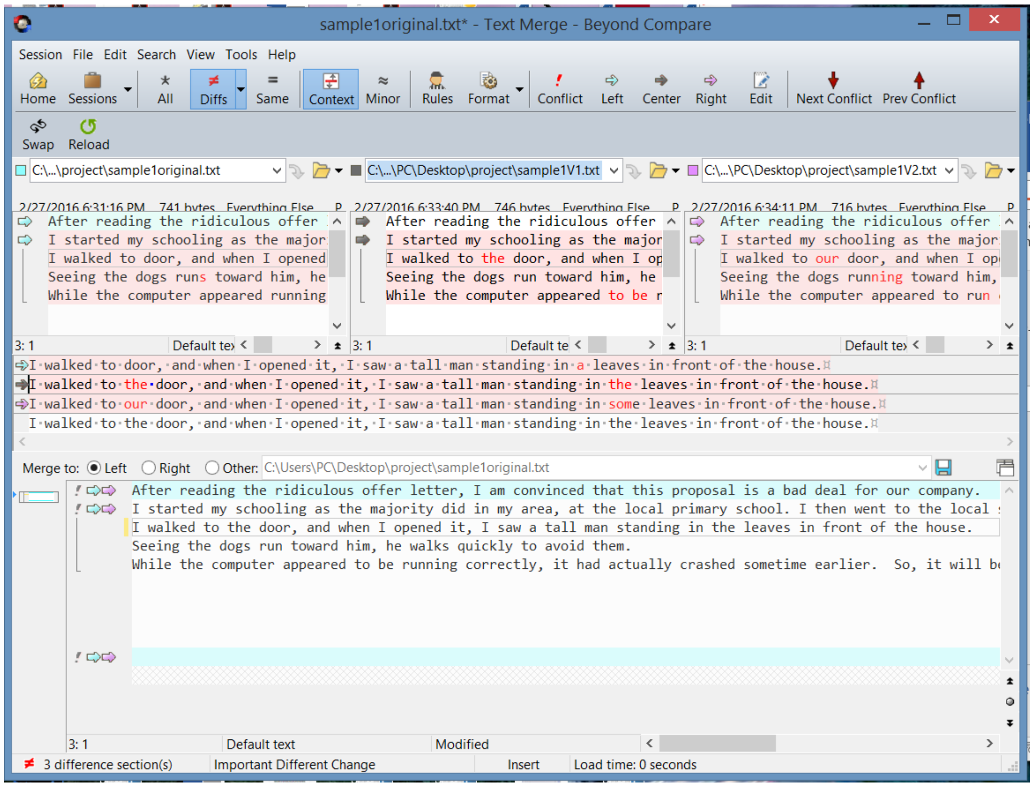
Seeing the dogs ~~runs~~ run toward him, he ~~walk~~ walks quickly to avoid them.

While the computer appeared to be running correctly, it ~~actually~~ had ~~been~~ actually crashed sometime earlier. So, it will be necessary to for John to reinstall the OS.

Figure 2-3 Diff Doc Merging software interface

2.4.1.3 Beyond Compare 4

Beyond Compare 4 [24] is designed to compare and merge folders and files. It supports a large variety of files, including MS Word, XML, text, Java, python C++, etc. Users will see the three version of a file, side by side, above the merged version. There is a feature to filter the view. One has the option to display only changes, only matches, or both in three versions. There is an option to ignore the unimportant changes, such as whitespace, comments, and character case. The important modifications will appear in red text, in all versions, while the unimportant ones are colored in blue. All changes made in the two revised versions will be included in a merged file automatically, but when a conflict happens (the same section of text has changes in both revised versions), the software will highlight it in red. It assists users in focusing on resolving the conflict by choosing one of the three versions. They have to click on the arrow next to the desired choice. Users cannot choose a certain change that appears in the conflict section. Instead, they are allowed to choose the whole section (paragraph) (figure 2-4).


sample1original.txt* - Text Merge - Beyond Compare
Session File Edit Search View Tools Help
Home Sessions All Diffs Same Context Minor Rules Format Conflict Left Center Right Edit Next Conflict Prev Conflict
Swap Reload
Merge to: Left Right Other:

```
C:\...\project\sample1original.txt
2/27/2016 6:31:16 PM   741 bytes   Everything Else
After reading the ridiculous offer
I started my schooling as the major
I walked to door, and when I opened
Seeing the dogs runs toward him, he
While the computer appeared running
3: 1   Default te>

C:\...\PC\Desktop\project\sample1V1.txt
2/27/2016 6:33:40 PM   746 bytes   Everything Else
After reading the ridiculous offer
I started my schooling as the major
I walked to the door, and when I op
Seeing the dogs run toward him, he
While the computer appeared to be r
3: 1   Default te

C:\...\PC\Desktop\project\sample1V2.txt
2/27/2016 6:34:11 PM   716 bytes   Everything Else
After reading the ridiculous offer
I started my schooling as the major
I walked to our door, and when I op
Seeing the dogs running toward him,
While the computer appeared to run
3: 1   Default te>

I walked to door, and when I opened it, I saw a tall man standing in a leaves in front of the house.
I walked to the door, and when I opened it, I saw a tall man standing in the leaves in front of the house.
I walked to our door, and when I opened it, I saw a tall man standing in some leaves in front of the house.
I walked to the door, and when I opened it, I saw a tall man standing in the leaves in front of the house.

C:\Users\PC\Desktop\project\sample1original.txt
After reading the ridiculous offer letter, I am convinced that this proposal is a bad deal for our company.
I started my schooling as the majority did in my area, at the local primary school. I then went to the local
I walked to the door, and when I opened it, I saw a tall man standing in the leaves in front of the house.
Seeing the dogs run toward him, he walks quickly to avoid them.
While the computer appeared to be running correctly, it had actually crashed sometime earlier.  So, it will b

3: 1   Default text   Modified
3 difference section(s)   Important Different Change   Insert   Load time: 0 seconds
```

Figure 2-4 Beyond Compare 4 merging software interface

2.4.2 Open Source Software

2.4.2.1 KDiff3

KDiff3 [25] is designed to support combining two or three text files/directories. Like the previous merging tools, the three versions will display side by side, with the highlighted and modified text. The user is able to accept all changes made in one line, from one version. They can also reject the other changes made, in other versions, on the corresponding line.

2.4.2.2 WinMerge

WinMerge [26] is an open source software that is designed to compare and merge two files/folders for the Microsoft Windows operating system. It is designed to compare text files, line by line, with the feature of applying all changes on the original text file.

2.5 Summary

Version control is a required feature in any collaborative work. It keeps all versions of a file/document, during the development process, as references. Moreover, it identifies the changes made in each version and merges them in one file/document. It is an important part of any version control system. There are two models of version control: distributed and centralized. The distributed model is recently more favored, as it has significant advantages which enhance teamwork. Considering the importance of version control in a collaborative edit, many efforts have been made to design tools which support versioning, or modify applications to be versioning aware.

Despite the existing interfaces of merging tool software do only a decent job in representing the modifications made in multiple versions of a text file, they often do not give the user freedom to choose a single change from one version, in case of conflicts, rather than select the whole line or paragraph.

Chapter 3 Requirements and Design

This chapter defines the primary users of the merging documents tool. Then we address requirements of our design; defining the conceptual model provides a clear idea of what the proposed interface will look like and how it will operate. Lastly, the prototype of the interface is demonstrated in detail.

3.1 Users

Knowing the target users of a developed system and their characteristics helps the designers produce user-centered products that meet users' needs effectively. The potential user of this prototype is the original author of a written piece that needs to be revised by one or more editors. Our focus was on one group of merging tool users, college students.

3.1.1 College students

We assume that the author is a graduate student in academia who is either international or domestic. The most important characteristic of the user is the skill level in using editing software, which can be basic, intermediate, or proficient. Also, time is a critical factor in students' academic lives due to the many deadlines they have to meet. In addition, international students who write in English as a second language might have a large number of corrections made on their written pieces.

3.2 Requirement Specifications

Our aim in this research is to design a user interface that helps people understand the corrections that have been made in revised versions and then choose the best revisions easily and efficiently to get the best merged result. Thus, we need to address and analyze the user requirements.

By identifying the users and analyzing the interfaces of existing merging tools (Chapter 2), we defined the requirements of the interface as the following:

1. It should be easy to use and learn.
2. It should display all versions of the document for user reference alongside the merge result.
3. There should be an understandable way to connect the changes made in both revisions to their corresponding places in the original version.
4. There should be an easy way for the user to filter the changes based on the level of importance for the user.
5. The changes that were made in the two versions of the file should be represented in an understandable way in the merge result.
6. There should be a convenient way for the user to view revision details for a selected correction.
7. There should be an easy way for the user to accept a specific modification even if this modification was made differently in both versions.

3.3 Conceptual Model

In order to produce a well-designed product, it is important to start constructing a conceptual model at the beginning of the design cycle of any system before prototyping. The role of a conceptual model is to identify what tasks the user can accomplish by using the software and how the system should behave regarding user interaction. We developed our conceptual model by identifying several usage scenarios, as seen in the following:

Usage scenario 1: A user wants to know what kind of corrections were made on both edited versions.

A graduate student wrote 100 pages of thesis manuscript and sent the document electronically to two professors on the thesis committee. Later, she received two versions of the document with many modifications. In such a large document with many corrections in both versions, it is frustrating to go over every single change to understand what kind of modifications were made. The solution is to filter the changes based on certain categories and use color coding to distinguish different types of categories.

Usage scenario 2: In the view displaying the three versions of a document as references, a user wants to know the corresponding place in the original document of specific corrections that were made on either edited version.

Displaying only the merged result might cause some issues for the user. In the merged result, changes are usually represented by a combination of strikethrough or underline and different colors. Despite this being a helpful way to perceive and understand the changes, it disturbs the reading flow [27], which has a negative impact on the user when deciding which changes to accept. Displaying three versions of a document beside the merged result diminishes the negative impact of change representation in the merged result. The user can read the changes in the context of either revised document and decide which is the best fit. Now the problem is how the user finds the corresponding place in the original document for specific modifications in either edited version to compare them in the context of each version. Two methods could be applied to solve the linking issue: using connection lines between the correction in either version and its corresponding place in the original file or highlighting the corresponding place in the original file when the user goes over any changes in both revised files.

Usage scenario 3: In the merged result, a user wants to know revision details about certain modifications.

Listing the revision information for all changes that were made in both edited documents or displaying them in balloons in the margin can be overwhelming and confusing to the user. A better method is to provide the information about certain modifications upon request by the user. This can be achieved by clicking on the desired correction to show more details about it; for example, in what version the change was made and what kind of correction it is (grammar, spelling, etc.).

Usage scenario 4: In order to obtain a good document, a user wants to choose a correction that was made in one or both edited versions, or keep the original text.

Combining three versions of a document results in conflicts when two different changes were made in both revised documents at the same place. One paragraph contains three conflicting changes; in the first position, the user decides to choose the changes that were made in the second revised document, while in the second position, the user thinks keeping the original text preserves his intended meaning. Finally, in the third position, the user finds that accepting the correction from the first revised document enhances that paragraph. To solve this issue, a drop-down or pop-up menu could be associated with a change to enable the user to apply it.

3.4 Prototyping

Prototyping is an essential process in designing a user interface. Converting the conceptual model into the visual medium helps the designers discover any issues that might arise in the designed interface and find alternatives to solve them. Also, prototyping is used to validate the requirements that were gathered at the early stages of the system design life cycle. Many methods could be used to produce a low-fidelity prototype, such as storyboarding, sketching, or using index cards. In order to produce our low-fidelity prototype of the merging documents tool interface, we used the Balsamic Mockups [28] application. Compared to drawing our design by hand on paper, using this software saved us a significant amount of time. This software provides a large number of user interface elements to drag and drop into the design easily, which allowed us to produce the prototype and its alternative faster and more easily with a professional look.

3.4.1 Initial prototype and the alternative

Figure 3-1 shows our initial low-fidelity prototype, which is reflected in our conceptual model. We divided the changes into checkboxes, where each one represents one category of change types. The three versions of a document are displayed side by side horizontally, and the merged result is shown beneath them. We selected the two revised documents and identified the changes by putting the text inside parentheses when the filtering changes are applied, highlighting the original document title to draw the user’s attention. To show more revision detail about certain corrections in the merged result, we show the revised version number after the changed text, and a pop-up window including that information is shown when the user clicks on that number. The reason behind displaying the versions horizontally is to connect the changes in both versions with the corresponding place in the original document by drawing dashed lines.

We assessed the low-fidelity prototype informally, simply relying on the author's judgement while performing simple tests with the interface. Two issues arose in this assessment about the initial design. First, using dashed lines to connect changes to the corresponding places in the original document resulted in too much complexity when reading the text in the three versions and caused visual discomfort, especially with large documents. Second, the space and location that specified the merged result at the bottom of the window were awkward for the user to work with to get a good combined result.

The alternative design (Figure 3-2) solves these issues. We split the screen into two sides to give more space for the merged result. The left side shows the merged result, while the right side displays the three versions vertically. We used the highlighting method instead of dashed lines to connect the changes in both revised versions to the corresponding places in the original version. We suggested using drop-down menus to accept or reject changes from the three versions.

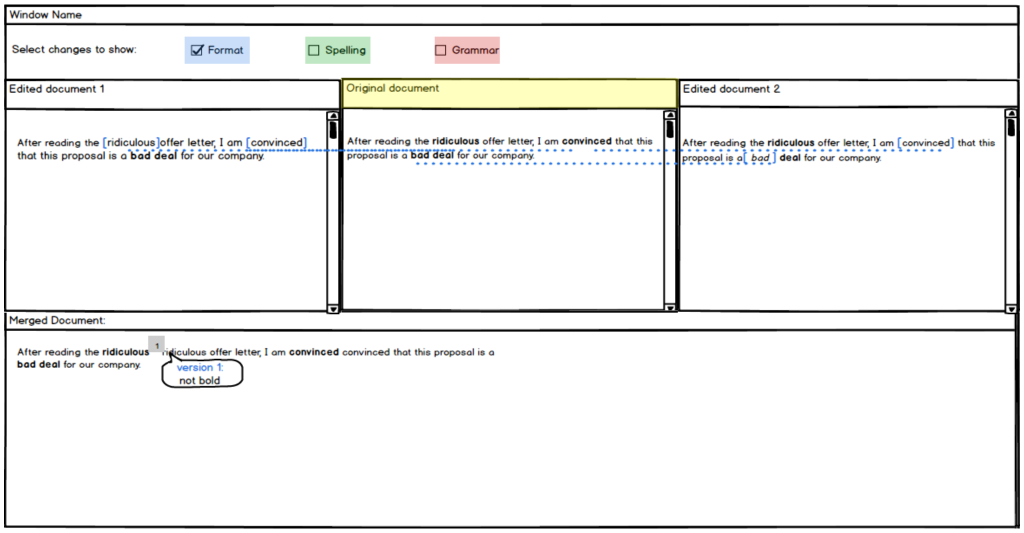

Window Name
Select changes to show:
Format
Spelling
Grammar
Edited document 1
Original document
Edited document 2
After reading the [ridiculous]offer letter, I am [convinced] that this proposal is a bad deal for our company.
After reading the ridiculous offer letter, I am convinced that this proposal is a bad deal for our company.
After reading the ridiculous offer letter, I am [convinced] that this proposal is a [bad] deal for our company.
Merged Document:
After reading the ridiculous ridiculous offer letter, I am convinced convinced that this proposal is a bad deal for our company.
1
version 1: not bold

Figure 3-1. The initial low-fidelity prototype of the merging documents tool interface.

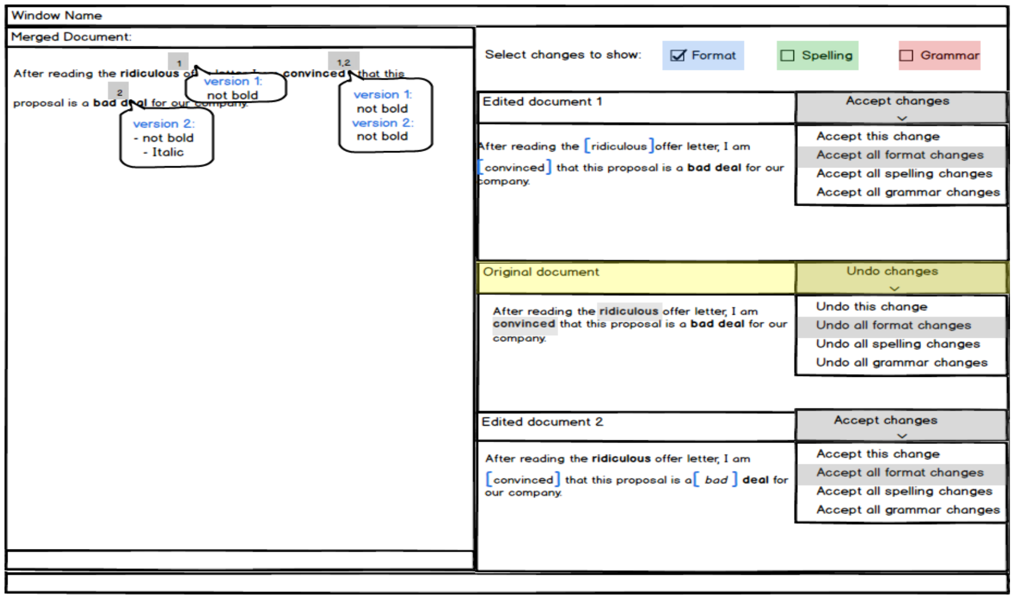

Window Name
Merged Document:
After reading the ridiculous offer letter, I am convinced that this proposal is a bad deal for our company.
1
version 1: not bold
1,2
version 1: not bold
version 2: not bold
2
version 2: - not bold - Italic
Select changes to show:
Format
Spelling
Grammar
Edited document 1
Accept changes
After reading the [ridiculous]offer letter, I am [convinced] that this proposal is a bad deal for our company.
Accept this change
Accept all format changes
Accept all spelling changes
Accept all grammar changes
Original document
Undo changes
After reading the ridiculous offer letter, I am convinced that this proposal is a bad deal for our company.
Undo this change
Undo all format changes
Undo all spelling changes
Undo all grammar changes
Edited document 2
Accept changes
After reading the ridiculous offer letter, I am [convinced] that this proposal is a [bad] deal for our company.
Accept this change
Accept all format changes
Accept all spelling changes
Accept all grammar changes

Figure 3-2. The alternative low-fidelity prototype of the merging documents tool interface.

3.5 Summary

We identified the primary user of our proposed interface. We also addressed the requirements of our design based on the target user's characteristics and the existing interfaces of merging tools. We identified the functionalities of our suggested interface by describing the conceptual model based on some usage scenarios. Finally, the initial prototype was built to meet the specified requirements and user needs. Some problems related to the initial design were solved by using alternative prototypes.

Chapter 4 Implementation

This chapter discusses the implementation of a high-fidelity interface for merging documents. Firstly, we introduce the programming language and its libraries that we used to build our prototype. Then, we provide the key challenges that we faced during the implementation and the relevant solutions or alternative methods. Finally, we give a demonstration of the interface and its features.

4.1 Python programming language

During a software development process, it is very important to produce a rapid prototype rather than full software, which is usually expensive, time-consuming, and might need to be reconstructed several times to solve any problems that might appear during testing. In contrast, an initial prototype is easy and fast to rebuild in case of any issues raised during testing because it has part of the full design's functions. Although Python has slow performance compared to other programming languages such as Java and C++, there are several reasons that make it the suitable tool to build rapid prototypes. Python has very simple syntax, which leads to more productivity; the designer is able to produce a piece of software with less coding compared to other programming languages. Moreover, Python has very useful built-in functions and a large number of libraries that serve different purposes such as image processing, natural language processing, or machine learning. Automatic memory management – the process of locating or reclaiming objects when they are no longer used – is another beneficial characteristic of Python. In addition, Python is very flexible in accepting calling or being part of a system that is written in other programming languages such as C or C++ – a very useful feature to easily convert the final design into higher performance programming languages [29]. Considering all these advantages, we chose Python to produce our high-fidelity interface for merging documents.

4.1.1 Python libraries

The objective of this thesis is to design a graphical user interface for merging documents that helps the user understand the modifications that were made in multiple versions of a Microsoft Word document and yields the best merged result. For that reason, we used two libraries: Python-docx and Wxpython, one for dealing with Microsoft Word documents and the other one to produce the graphical user interface.

4.1.1.1 Python-docx

Python-docx is designed to help programmers create Microsoft Word documents with the docx extension or deal with an existing one. We used this library to read Microsoft Word documents in order to display them on the interface. Unlike plaintext Microsoft Word documents based on a certain hierarchy structure, the Python-docx module defines three object types to enable the designer to easily handle this structure. The ‘Document’ object is the top level in the hierarchy structure and represents the entire document. Inside a document there is one or more paragraphs, and the ‘Paragraph’ object is defined to cope with paragraphs. Also, each paragraph object consists of one or multiple Run’ objects, where a new run is created whenever the text style is changed. Each run has attributes such as text, size, font, bold, and italic [30] [31].

4.1.1.2 Wxpython

We used this library to implement the graphical user interface (GUI) for several reasons. First, it is an open source library extension of the wxWidget cross-platform GUI, which is written in C++ and gives the interface fast performance. Also, an application that is designed by using this library can run on various systems with little or no modifications in the code. Moreover, Wxpython has many useful widgets that serve our goal to design an efficient and easy-to-use interface for merging documents [32] [33].

4.2 Implementation challenges

Any software programmer confronts some difficulties while coding a new piece of software and must overcome these problems either by finding a solution to the challenge or coming up with alternatives. As we developed our proposed interface, we encountered two main challenges.

The first problem we encountered involved reading the common format (i.e., bold and italic) of an MS Word document text using the Python-docx library. The Python-docx library, as mentioned earlier, organizes text into paragraphs, which are further divided into runs. The library expects that each run represents a section of text with a different formatting style. However, we found that Microsoft Word splits misspelled words into multiple runs, without there being any change of style. In order to get the text and its format to display correctly in our interface, we developed an approach to recognize the word even if it is split over multiple runs, and then retrieved its format using Wxpython library functions.

Another challenge we faced was identifying the corrections that were made in both revised versions, which our prototype marked by underlining words using different colors and patterns to indicate the type of the corrections. We used a rich text box widget to display the formatted text of MS Word documents on the interface, but this widget does not support the combination of colored and patterned underlines. We needed different colors and patterns for underlines to highlight different types of modifications in both revised documents, even if an overlap occurred where the same spot had multiple types of changes, such as grammar and font. We tried to draw the underlines over the rich text box widgets, but the result was unsatisfying. These newly-drawn underlines were very slow and shifted down from words a little when the scrollbar was used. Thus, we used an alternative method to represent the changes; we put the corrected text inside parentheses with a different shape and color to indicate the type of changes based on our change classifications.

4.3 Demonstration of the high-fidelity prototype

This section illustrates our proposed interface and its functionalities. First, we state some design principles that we followed in our implementation. After that, a discussion of the input data will be given. Finally, the components of the interface and its capabilities will be explained in greater detail.

4.3.1 Design principles

In order to provide a good prototype that met the users' needs, it is very important to follow the general interaction design guidelines to ensure the quality of the final product. We considered five main design principles [34] during the implementation process of our high-fidelity user interface for merging documents:

1. **Visibility**

   It is important to make all parts of the interface visible to the users, in order to let the users, perform the intended functions. The high-visibility functions of any user interface will give the users a clear idea to what they can do and how to perform it easily with less effort and in less time.

2. **Feedback**

Users must be informed about the actions they perform. There are several forms of feedback that the system provides to its users: visual changes, audio, tactile, or any combination of these. Providing users with relevant feedback makes it clear that the software is responding to users ’actions.

3. **Constraints**:

   In some cases, it is important to restrict users from performing certain system functions (i.e., the wrong task) in specific situations. Typically, designers can apply this principle by deactivating a targeted part of the interface.

4. **Consistency:**

   People use their previous experiences with other systems in a new one. For that reason, programmers must design an interface that follows the common rules of performing normal tasks such as clicking the left mouse button to select an object on the interface. This is called external consistency. Programmers should also apply the internal consistency in their product which includes consistent visual design and using similar system behaviors to respond to similar user actions.

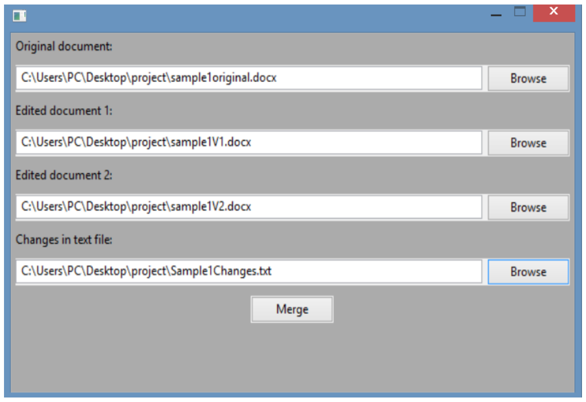
5. **Affordance**

This term refers to the appearance of interface objects that give the user a clue on how to use them. For example, buttons should be designed in way that affords clicking.

In the following sections, we will point to each of these principles while we give further descriptions of our proposed interface.

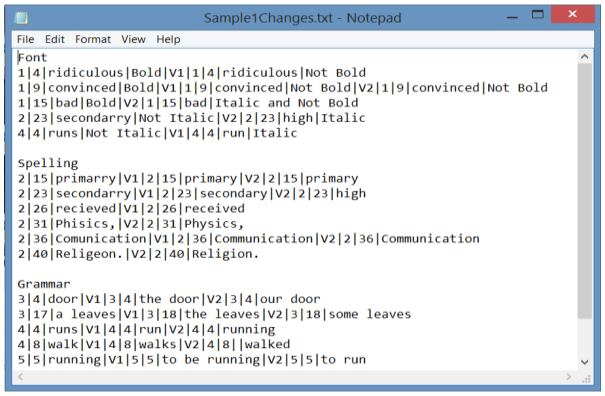
4.3.2 Input data

Four files should be served as inputs for our prototype (figure 4-1): three MS Word documents, where one of them is an original document written by an original author and the others are revised versions from different revisers, and a fourth file that is a text file that serves as the Oracle system. In this fourth file, we grouped the corrections that were made in both edited versions into four categories: font, grammar, spelling, and editing. Any font changes such as bold and italic were classified under font changes. Any grammar or spelling corrections were located under the grammar and spelling categories, respectively. Other corrections made in one or both revised documents that don't fall under any previous categories were included under a general editing category. Under each classification, we listed the changes starting by paragraph number, word number, and the word string in the original document, followed by the same information for versions 1 and 2. Each line contains one change, and the information related to this change was separated by the '|' symbol (figure 4-2).



Original document:
C:\Users\PC\Desktop\project\sample1original.docx
Edited document 1:
C:\Users\PC\Desktop\project\sample1V1.docx
Edited document 2:
C:\Users\PC\Desktop\project\sample1V2.docx
Changes in text file:
C:\Users\PC\Desktop\project\Sample1Changes.txt

Figure 4-1 Input screen of the proposed interface

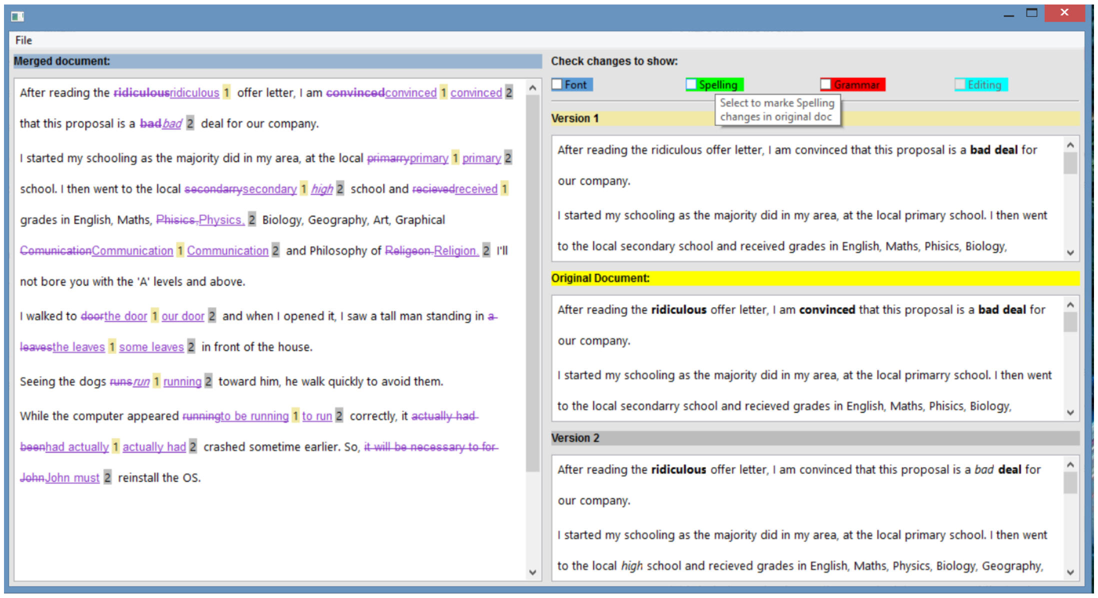


```
Font
1|4|ridiculous|Bold|V1|1|4|ridiculous|Not Bold
1|9|convinced|Bold|V1|1|9|convinced|Not Bold|V2|1|9|convinced|Not Bold
1|15|bad|Bold|V2|1|15|bad|Italic and Not Bold
2|23|secondarry|Not Italic|V2|2|23|high|Italic
4|4|runs|Not Italic|V1|4|4|run|Italic

Spelling
2|15|primarry|V1|2|15|primary|V2|2|15|primary
2|23|secondarry|V1|2|23|secondary|V2|2|23|high
2|26|recieved|V1|2|26|received
2|31|Phisics,|V2|2|31|Physics,
2|36|Comunication|V1|2|36|Communication|V2|2|36|Communication
2|40|Religeon.|V2|2|40|Religion.

Grammar
3|4|door|V1|3|4|the door|V2|3|4|our door
3|17|a leaves|V1|3|18|the leaves|V2|3|18|some leaves
4|4|runs|V1|4|4|run|V2|4|4|running
4|8|walk|V1|4|8|walks|V2|4|8||walked
5|5|running|V1|5|5|to be running|V2|5|5|to run
```

Figure 4-2Tthe text file that contains the editing changes

4.3.3 The main screen

The main screen is divided vertically into two parts (figure 4-3). The left section displays the merged document. On the top of the right part, the changes filters panel appeared. Beneath that, the three versions of a document are displayed where the original text is showed in between the two versions. Colored title headers differentiate each version. We used bright yellow to highlight the original version.



Merged document:

After reading the ~~**ridiculous**~~ridiculous 1 offer letter, I am ~~**convinced**~~convinced 1 convinced 2 that this proposal is a ~~**bad**~~*bad* 2 deal for our company.

I started my schooling as the majority did in my area, at the local ~~primarry~~primary 1 primary 2 school. I then went to the local ~~secondarry~~secondary 1 *high* 2 school and ~~recieved~~received 1 grades in English, Maths, ~~Phisics,~~Physics, 2 Biology, Geography, Art, Graphical ~~Comunication~~Communication 1 Communication 2 and Philosophy of ~~Religeon.~~Religion. 2 I'll not bore you with the 'A' levels and above.

I walked to ~~door~~the door 1 our door 2 and when I opened it, I saw a tall man standing in ~~a leaves~~the leaves 1 some leaves 2 in front of the house.

Seeing the dogs ~~runs~~*run* 1 running 2 toward him, he walk quickly to avoid them.

While the computer appeared ~~running~~to be running 1 to run 2 correctly, it ~~actually had been~~had actually 1 actually had 2 crashed sometime earlier. So, ~~it will be necessary to for John~~John must 2 reinstall the OS.

Check changes to show:

Version 1

After reading the ridiculous offer letter, I am convinced that this proposal is a **bad deal** for our company.

I started my schooling as the majority did in my area, at the local primary school. I then went to the local secondary school and received grades in English, Maths, Phisics, Biology,

Original Document:

After reading the **ridiculous** offer letter, I am **convinced** that this proposal is a **bad deal** for our company.

I started my schooling as the majority did in my area, at the local primarry school. I then went to the local secondarry school and recieved grades in English, Maths, Phisics, Biology,

Version 2

After reading the **ridiculous** offer letter, I am convinced that this proposal is a *bad* **deal** for our company.

I started my schooling as the majority did in my area, at the local primary school. I then went to the local *high* school and recieved grades in English, Maths, Physics, Biology, Geography,

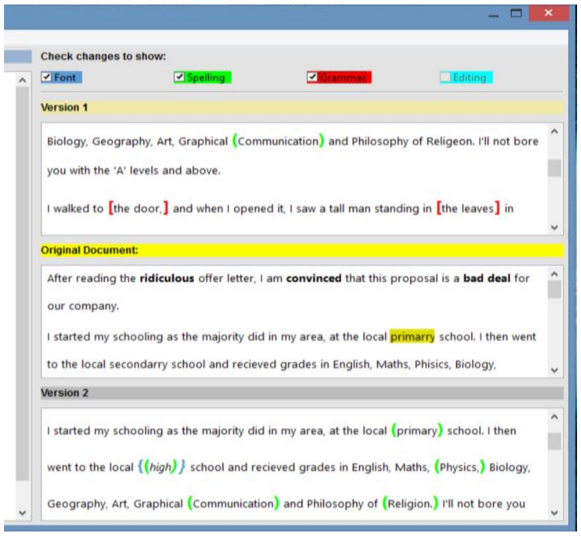
Figure 4-3 The main screen of the proposed interface

4.3.3.1 Filtering and identifying changes

To give the user the freedom to select the classification of the modifications to display on the screen, we used check boxes. If there are no changes that belong to one of our change classifications, the checkbox that represents that category will be deactivated – this is the constraint principle reflected in our design. We applied color coding with different parenthesis shapes to highlight the changes that were made in both revisions. When the user selects the font check box with blue background, the font changes will be highlighted by including them between blue squiggly brackets. The green highlighted spelling checkbox will draw green round brackets around spelling corrections. Also, when the user checks the grammar checkbox with red background, grammar corrections will be included in red square brackets. If the changes don't fall under these three classifications, they appear inside turquoise chevron brackets when the turquoise highlighted editing checkbox is selected.

At the beginning of our interface implementation, we applied our modifications differentiations model on both widgets that contain revised versions. To meet our proposed interface requirement of binding the changes that were made in either revised version with the corresponding place in the original version, we provided a function of highlighting a target text of the original document when the user hovers over the text inside the parenthesis that appears in either widgets of the edited versions (figure 4-4).

To protect the user from losing focus while controlling two widgets that display the edited documents, we created an alternative implementation. We applied our modifications differentiations model on the rich text box that displays the original document and used our highlighting function to highlight the corresponding text in both rich text boxes that show the two revised versions as well as a corresponding text in the merged document when the user moves the mouse over the text inside parenthesis in the original version (figure 4-5). Highlighting the text and drawing the parenthesis represent an application of the feedback design principle. Also, we used tooltips to guide the users to direct their attention to the original document where we applied our modifications differentiations model on the text that has been corrected in either edited version (figure 4-6).



Biology, Geography, Art, Graphical (Communication) and Philosophy of Religeon. I'll not bore you with the 'A' levels and above.

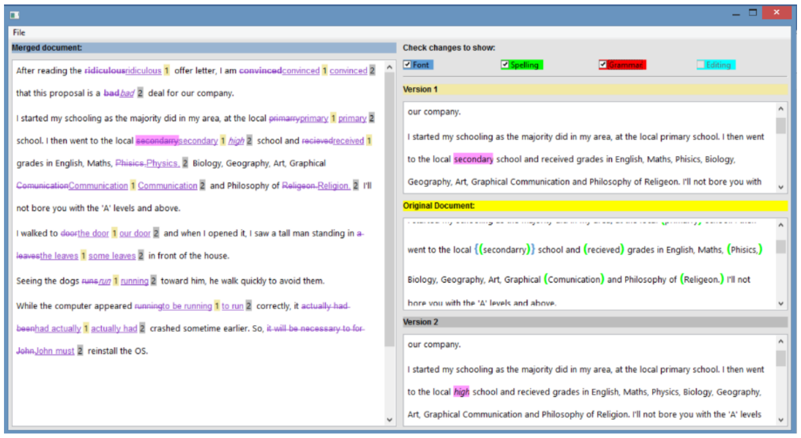
I walked to [the door,] and when I opened it, I saw a tall man standing in [the leaves] in

After reading the **ridiculous** offer letter, I am **convinced** that this proposal is a **bad deal** for our company.

I started my schooling as the majority did in my area, at the local primarry school. I then went to the local secondarry school and recieved grades in English, Maths, Phisics, Biology,

I started my schooling as the majority did in my area, at the local (primary) school. I then went to the local {(*high*)} school and recieved grades in English, Maths, (Physics,) Biology, Geography, Art, Graphical (Communication) and Philosophy of (Religion.) I'll not bore you

Figure 4-4 Initial implementation of identifying changes



After reading the ~~**ridiculous**~~ridiculous 1 offer letter, I am ~~**convinced**~~convinced 1 convinced 2 that this proposal is a ~~**bad**~~*bad* 2 deal for our company.

I started my schooling as the majority did in my area, at the local ~~primarry~~primary 1 primary 2 school. I then went to the local ~~secondarry~~secondary 1 *high* 2 school and ~~recieved~~received 1 grades in English, Maths, ~~Phisics,~~Physics, 2 Biology, Geography, Art, Graphical ~~Comunication~~Communication 1 Communication 2 and Philosophy of ~~Religeon.~~Religion. 2 I'll not bore you with the 'A' levels and above.

I walked to ~~door~~the door 1 our door 2 and when I opened it, I saw a tall man standing in ~~a leaves~~the leaves 1 some leaves 2 in front of the house.

Seeing the dogs ~~runs~~*run* 1 running 2 toward him, he walk quickly to avoid them.

While the computer appeared ~~running~~to be running 1 to run 2 correctly, it ~~actually had been~~had actually 1 actually had 2 crashed sometime earlier. So, ~~it will be necessary to for John~~John must 2 reinstall the OS.

our company.

I started my schooling as the majority did in my area, at the local primary school. I then went to the local secondary school and received grades in English, Maths, Phisics, Biology, Geography, Art, Graphical Communication and Philosophy of Religeon. I'll not bore you with

went to the local {(secondarry)} school and (recieved) grades in English, Maths, (Phisics,) Biology, Geography, Art, Graphical (Comunication) and Philosophy of (Religeon.) I'll not bore you with the 'A' levels and above.

our company.

I started my schooling as the majority did in my area, at the local primary school. I then went to the local *high* school and recieved grades in English, Maths, Physics, Biology, Geography, Art, Graphical Communication and Philosophy of Religion. I'll not bore you with the 'A' levels

Figure 4-5 Alternative implementation for identifying changes

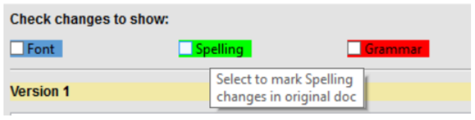


Figure 4-6 Tooltip to direct user attention to the original document where the changes will be identified

4.3.3.2 Merged document

The merged result will be shown on the left side of the interface. Changes will be represented to the users by different text color where the original text has strikethrough and the corresponding corrections that were made in either revision will be underlined followed by either number 1 or 2 (number 1 indicates changes made in revised document 1, and number 2 indicates the second revision). In addition, each one of these numbers has a background that matches the colored title header of the widgets that contain the two edited documents (figure 4-7).

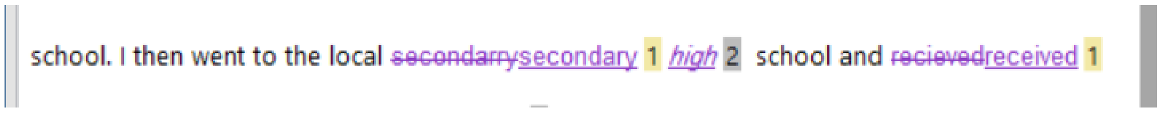
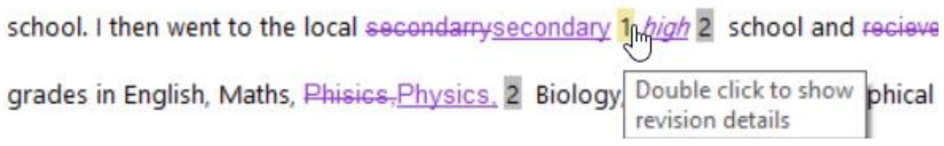

school. I then went to the local ~~secondarry~~secondary 1 *high* 2 school and ~~recieved~~received 1

Figure 4-7 Changes representation in merged document

4.3.3.3 Revision details

A well-designed system provides a decent amount of information to the users upon their request. When the user moves the mouse over the number that represents the version's number, the curser will change to a finger pointer that offers clicking and a tooltip will appear to direct the user to get more information about that modification by double-clicking on the number (figure 4-8). By clicking on the number, a popup window will display more details about the type of the change based on our change classifications model using our coloring code approach. This popup window will disappear by clicking anywhere on the screen (figure 4-9).


school. I then went to the local ~~secondarry~~secondary 1 *high* 2 school and ~~recieve~~

grades in English, Maths, ~~Phisics,~~Physics, 2 Biology, phical

Double click to show revision details

Figure 4-8 Tooltip to direct users to get more revision details

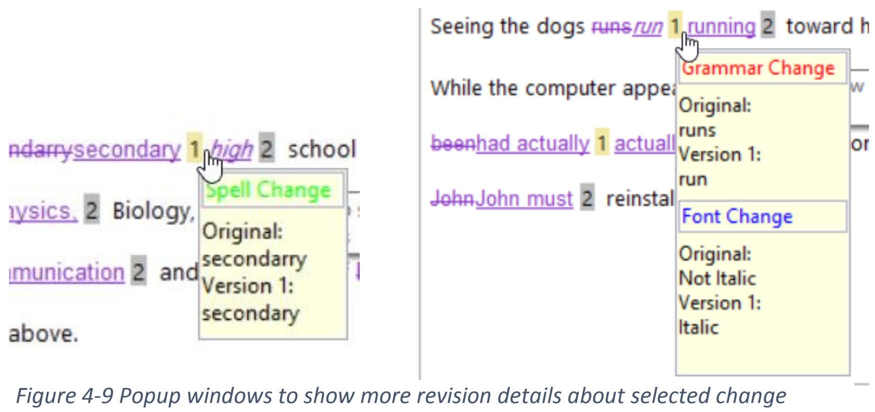


Figure 4-9 Popup windows to show more revision details about selected change

4.3.3.4 Accept/Reject changes

It is not easy to create a really handy control widget that enables users to choose the desired corrections to get the best merged document. As a result, we developed three alternatives controls to enable the user to select a desired correction either by keeping the original text or accepting one of the revisions.

Our initial changes control used the menu bar located above the merged document widget. This menu bar contains two menus: one to deal with changes that were made in either version, and the other to reject the corrections by keeping the original (figure 4-10). We found that this control did not meet our objective to help the user better understand the revision process that was done on his/her writing and choose the best fit. More specifically, this menu did not consider the combination of the change classifications if changes occurred in the same place. Also, in order to accept or reject a specific change, users had to go back and forth between the changed text in the merged result widgets and the menu bar, which consumed time and effort.

To overcome the problem with our initial control change implementation, we produced a second control change panel. Clicking on a strikethrough text lead to a popup menu that enabled the user to keep the original text, while doing the same action over an underline text lead to a popup menu that enabled the user to accept that text (figure 4-11). Although this approach was more flexible than the original menu bar, it was still not informative enough to enable the user to select the best corrections.

Our final control change implementation solved the issues related with the two previous implementations. When the user hovers over any changes, a tool tip showing the control panel will appear by double-clicking (figure 4-12). A popup window appears under the text by double-clicking on any corrected text. This window informs the users about the type of the changes that were made in the current position and give users the option to choose a change. This window disappears by clicking anywhere on the window or clicking on the "Apply" button after choosing the desired correction to applied (figure 4-13).

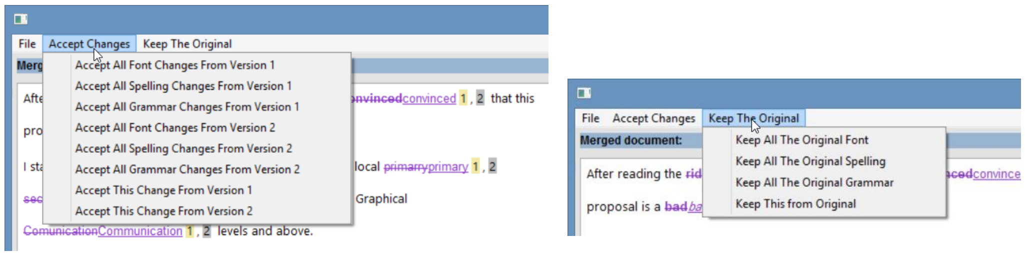

File
Accept Changes
Keep The Original
Accept All Font Changes From Version 1
Accept All Spelling Changes From Version 1
Accept All Grammar Changes From Version 1
Accept All Font Changes From Version 2
Accept All Spelling Changes From Version 2
Accept All Grammar Changes From Version 2
Accept This Change From Version 1
Accept This Change From Version 2
Merged document:
Keep All The Original Font
Keep All The Original Spelling
Keep All The Original Grammar
Keep This from Original

Figure 4-10 Menu bar to accept/reject changes

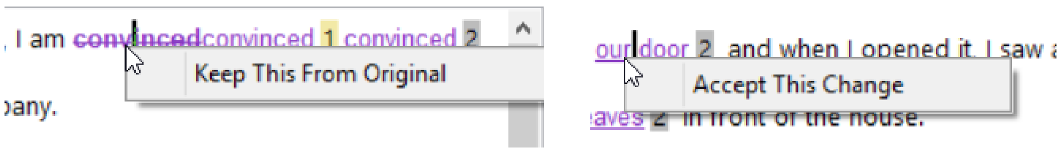


Figure 4-11 popup menus to accept/reject changes

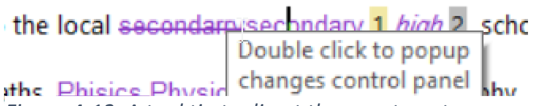
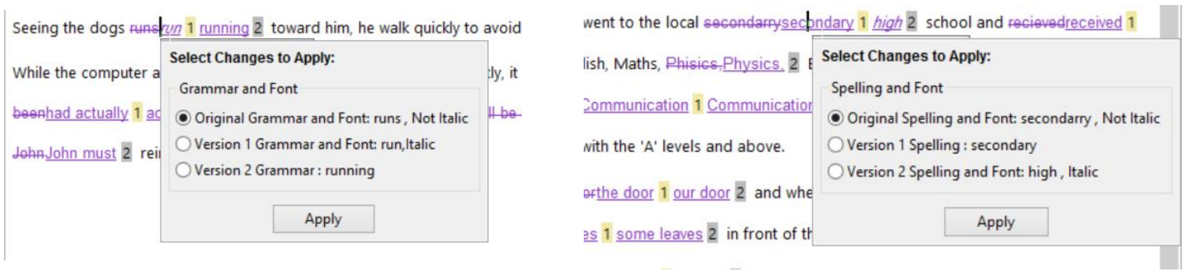


Figure 4-12 A tool tip to direct the user to get a changes control panel


Select Changes to Apply:
Grammar and Font
● Original Grammar and Font: runs , Not Italic
○ Version 1 Grammar and Font: run,Italic
○ Version 2 Grammar : running
Apply

Select Changes to Apply:
Spelling and Font
● Original Spelling and Font: secondarry , Not Italic
○ Version 1 Spelling : secondary
○ Version 2 Spelling and Font: high , Italic
Apply

Figure 4-13 Our final control panel to accept/reject changes

4.3.3.5 Saving the merged result

Our high-fidelity interface has a feature to save the final document after choosing the desired corrections. A save menu located in the menu bar above the combined file widget enables users to save the merged result as an HTML file (figure 4-14). We chose to save the result as HTML for two reasons. First, it is the only file type that supports formatted text in the Wxpython library. Second, most editing software supports HTML, including Microsoft Word.

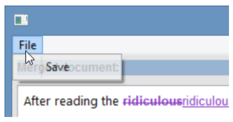


Figure 4-14 Menu bar to save combined result as HTML.

4.4 Limitation

A well-designed system helps the user recover from their mistakes. There are two types of error recovery: backward recovery and forward recovery. In forward recovery, there is no undo action, but there is an alternative way to allow users to recover from their mistake. Backward recovery allows users to revert to the previous state (i.e., before they executed their last interaction) [35]. Our interface lacks this feature; when the user selects a certain correction, he/she cannot undo to reselect from other options of correction.

4.5 Summary

In this document, we discussed the programing language and libraries that we used to implement our interface for merging documents. We provided a brief description of the design principles that we followed in our design. Also, we discussed an illustration of our interface and its functionalities and limitation in further detail.

Chapter 5 Evaluation

We conducted a user study to test the usability of the interface of the document merging tool. Before our study, we hypothesized that the change filtering tool that we used in the interface would help the user to better understand the modifications that had been made in the two revised versions of the original document. We also assumed that the change classification feature we used would guide users to focus their primary attention on the most important corrections that had been made in the two edited versions. We assumed that for any changes involving deletion, addition, and relocation, they should be grouped under the "Editing" category because they could change the meaning of the original content. Moreover, we expected that the following two features would assist the user in choosing the best version among the three versions in each place where a change had been made.

- we applied a highlighting feature in the interface to highlight the corresponding text that had been modified in one or both revisions and the merged result when the user hovering over the original text that was included inside parentheses.
- we made the change control panel a pop-up in the merged result; when users double click on any of the corrections, they are given the option to choose the best changes by either keeping the original or accepting the correction from either the first revision or the second.

We used two sets of Microsoft Word documents as the dataset of the study. Each set included three MS Word versions of the same piece of writing; one of the three was the original version and the other two were the revised versions. We made a variety of mistakes in the original version, such as font errors, deletions, relocations, and spelling and grammar errors. In the two revised versions, we corrected the mistakes that we had made in the original version differently or similarly. In addition to the three MS Word documents, there is a text file that works as the Oracle system. This file lists and classifies the changes that were made in the two revised versions, corresponding to the place/word in the original document. In the first set, each MS Word document contained approximately 75 words, while in the second set, each document contained approximately 125 words. The following sections describe in detail the participants, the procedure of the study, the tasks, and the results of the study.

5.1 Subjects

Ten graduate students from the University of Wisconsin–Milwaukee participated in the study. Eight of them were from the College of Engineering and Applied Science, one was from the School of Information Studies, and one was from the College of Nursing. The participants came from different cultures: five were Arabic, three were South East Asian, and two were domestic students. Each participant had experience with collaborative writing, working with at least one type of versioning control software. In addition, each of them had experience with the "track changes" feature in MS Word. Their prior experience helped us to collect their opinions on how the proposed interface compared with previous experiences with this type of software.

5.2 Procedures

The study took place in our multimedia laboratory using a Windows laptop with an external mouse for greater convenience. We met with one participant at a time, and we provided a five-minute orientation on the interface, where participants were introduced to the interface's features. Following the orientation, two sets of tasks were given to each participant. Each set of tasks was to be completed on one set of the writing samples. No time limit was set for the participants to complete the tasks, because our aim was to observe how the proposed interface would be used in practice. In practice, the time spent by the participants to complete the two sets of tasks ranged from 30 to 45 minutes. Users were encouraged to think aloud while performing the tasks to assist us in understanding their actions. Moreover, we took notes and recorded the users' screens using TinyTake software in order to get more information about the users' interactions with the interface, for further review after the study. Following the completion of the tasks, we further inquired into the users' opinions of the proposed interface with an online, post-study questionnaire, which was administered using UWM Qualtrics. We used a survey with six items, including one multipart question with Likert-type items related to satisfaction, two questions with Likert-type items related to measure levels of agreement/disagreement about the change classifications, and three open-ended questions [see Appendix B]. Participants were asked to complete a survey, either after the tasks completion or at a more convenient time, using email.

5.3 Tasks

Two sets of similar tasks [see Appendix A] corresponding to two different writing samples were given to participants. We created the tasks based on real-world scenarios adapted from collaborative writing in academia. The tasks focused on finding a specific type of modification that had been made to the original document based on our changes classifications and accepting/rejecting the correction.

For example:

- For one of the tasks, we asked participants to find a correction that had been made in either the first or second revision that could change the meaning of the original document, as we assumed that this type of change should be taken care of first.

- Another task asked the participants to find three types of changes that had been made in one place, such as font change, grammar, and spelling corrections.
- In addition, we asked the users to find spelling/grammar corrections that had been corrected differently in each of the revised versions and choose the preferred correction.
- At the end of each set of tasks, the participants were required to take care of one of the categories of changes, for instance, spelling/grammar, and accept the best fit in the merged result.
- After completing all tasks, the users were asked to save the merged result on the desktop.

5.4 Results

Nine out of ten participants completed the surveys; two of the nine filled out the survey immediately after the task completion, and the rest of the surveys were emailed to participants upon request, to be completed at a more convenient time.

We received both positive and negative feedback from participants through observations and questionnaires. On the positive side, participants were able to complete all tasks in a short amount of time. We also drew the following conclusions based on our observations and user feedback:

- The high fidelity interface appeared to be easy to use and learn. Although only five minutes of orientation were provided, participants used the interface design effectively to perform the given tasks. Seven out of nine responses indicated that our interface design was easy to use, and six agreed that it was easy to learn. Two responses neither agreed nor disagreed about the ease of use and learning, and one subject found that the interface was not easy to learn.

- Six participants found that the proposed interface provided a decent amount of information that helped them to understand the modifications that had been made in the two revisions. One participant did not find the interface to be informative, and two responses expressed a neutral opinion on this matter.
- Seven participants considered the change classification tool that we implemented in the interface to be very useful in assisting the author to locate and understand the corrections that had been made to the writing in the two revised versions. One of the participants suggested applying subcategories under the grammar heading to address verb tense, passive/active voice, and singular/plural issues. She also emphasized that this software would be a great tool for teaching writing to high and middle school students, who could learn from their mistakes by using the changes classification feature.
- Eight participants agreed that the change classification feature helped the user to focus first on the most important modifications that had been made in the two revisions.
- The participants admired the color code that we used to identify the different types of modifications, as well as the highlighter that was used to highlight the corresponding text in one or both revised versions, as well as the corresponding text in the merged result when the user moves the mouse over the text inside parentheses in the original version.
- The participants found the change control panel to be very helpful in easily choosing the most suitable corrections.
- One of the participants added this complement about the interface:

> "I sometimes struggle with using the Microsoft Word track changes feature, especially when there are multiple versions of a document, and I am waiting to accept changes until the end of the editing process. This interface helped me easily identify changes by user, which can be another area of difficulty in MS Word."

This positive feedback was encouraging. However, we also noticed some issues related to the interface and the tasks. First, regarding the issues of the interface, we found that:

- All of the participants expected that the change filtering feature, based on our proposed classifications, would be applied on the merged result beside the original document, whereas we only implemented it on the original version.
- Users double-clicked to select the original text that was included inside parentheses instead of hovering over it to highlight the corresponding text that had been modified in one or both revisions and the merged result. The participants also complained about the highlighter going away when the mouse was not hovering over the text that is included inside parentheses in the original document. They expected that the highlighter would stay until they selected other text in the original text to highlight the corresponding modifications.
- The method we used to indicate that a specific modification had already been accepted/rejected was very confusing to the users. In places where the user had already accepted/rejected a modification, we got rid of the highlighting in the corresponding modified text in the merged result and kept the highlighting in both revisions when the user hovered over the original text that is included inside parentheses. One of the participants suggested that for every change that has been accepted/rejected, we remove the surrounding parentheses that appear in the original text.

- The participants faced difficulty when they double-clicked the changes in the merged result to open the change control panel pop-up. From our observation, we noticed that the users tried to right-click in order to open the change control panel.
- The participants expected to see three options on the change control panel, even when the two revisions were the same. However, when the modifications were made in the same way in the two revisions, we only showed two options in the changes control panel. See Figure 5-1:

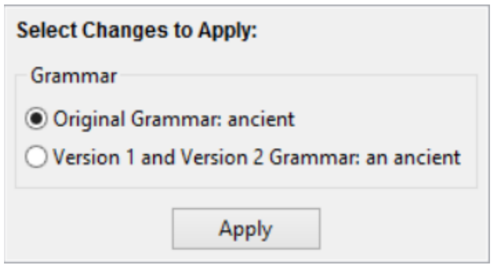

Select Changes to Apply:
Grammar
Original Grammar: ancient
Version 1 and Version 2 Grammar: an ancient
Apply

Figure 5-1 Shows the changes control panel with two options when the modification was made in the same way in both revisions.

- While we considered modifications that include deletion, relocation, and insertion to fall under the “Editing” category, this classification could result in changing the original text’s meaning. Most of the participants were looking for grammar/spelling modifications when they were asked to find a correction that could change the meaning of the original text.
- Most of the participants did not perform one of the tasks correctly, which was when they were asked to find a format-only change and reject it. During this task, they checked the font category only and unchecked the other changes categories. As a result, they rejected changes that had been made to spelling and font or to grammar and font.
- The users were looking for a way to undo the changes that they had accepted/rejected, which is a limitation of the interface.

Second, regarding the deficiencies with respect to the tasks, we found that:

- The participants were confused by the task that asked them to find format changes. They asked questions about it, and some of them thought we were asking about the paragraph or sentence format. Others asked for clarification by asking, “What do you mean by format change?”
- Some participants were confused by the task that asked them to find grammar or spelling changes in instances where there could be multiple categories of changes in the same place.

5.5 Lessons learned from the usability test

In section 5.4, we discussed the results of the usability test, as well as the limitations of our interface. In this section, we will discuss the limitations of the user study.

First, although the number of users was sufficient to find many of the usability problems in our interface design, we believe that more participants would provide even more useful feedback.

Second, the user and the writing were unrelated. Since participants were not the authors of the piece of writing they were working on, and they were not familiar with the given piece of writing, the study set-up did not create a realistic setting. This issue may have created barriers for users in performing the tasks.

Finally, the size of the writing sample used in the study was small. We believe a larger document would create a more advanced testing environment.

5.6 Summary

This chapter provided an evaluation on the proposed interface for a merging documents tool. Details of the procedure and general observations were described. Positive and negative feedback were summarized, providing good direction for further improvement of the interface.

Chapter 6 Conclusions

In academia, it is very common to revise documents in a parallel manner, where each editor works independently to make corrections to the same original document. Thus, a document's author often receives multiple revisions to process. As a result, many merging software tools have been developed to merge revised versions with an original document. Although these merging tools provide the user with all of the information about the changes and who made them, the interfaces lack the feature of allowing the user to filter the corrections, based on level of importance, in order to prioritize which revisions to address first. In addition, these software tools do not provide the user with a way to select the best revision in the case of disagreements. We designed and developed a high fidelity interface tool for merging documents to display three versions of a document as well as the merged result of all three versions. The primary goal of creating this interface was filtering the changes made on two revised versions of the original document to help the author better understand the corrections. Also, the interface helps the user take care of the most important changes first. The designed interface provides an easy way for the user to choose the best correction from the three versions.

We also conducted a user study to evaluate the interface of the documents merging tool in terms of user satisfaction, which has rarely been examined in related research. The results of the user study bring new insights to the field of merging documents. The study shows that changes classifications enhance the user understanding of changes made in the two revisions, as well as guide the user to take care of the most important changes first. Users consistently reported that they were satisfied with the interface in terms of ease of use, ease of learning, and effectiveness.

6.1 Future work

Despite extensive attempts to improve English grammar and spelling checkers, this area still needs more consideration and collaboration between computer and linguistic scientists.

There are many issues related to spelling checkers. First, the spelling checker flags some proper names as misspelled, which annoys some users, but some spelling checkers enable the user to add the proper names to their dictionaries to avoid marking them as spelling errors in the future. Second, spelling checkers may mark some words as mistakes that are correct; this confuses users. Third, the essential problem with the spelling checker is ignoring the context surrounding the word, which leads to improper use of homonyms. This results in embarrassing and meaningless writing.

The existing grammar checkers are more limited than spelling checkers. They do not catch grammar errors that can be caught easily by insights. For example, the grammar checker in MS Word does not find any grammar mistakes in the following sentences:

“Marketing are bad for brand big and small. You Know What I am Saying? It is no wondering that advertisings are bad for company in America, Chicago and Germany. McDonald's and Coca Cola are good brand. ... Gates do good marketing job in Microsoft.[36]”

We also believe that there could be benefit to merging tools that look for cut-and-paste errors in revised documents. In long-lived documents like graduate theses, it is common for authors to move content to various locations as they experiment with different narrative structures. It is easy to make errors in this process, such as not integrating copied material correctly, or forgetting to remove redundant copies of the same text. Automated support could ease this problem.

Integrating the document comparison tool with a machine learning classifier to classify the differences among the versions into different categories—such as grammar, spelling, deletion, insertion, and meaning changes—would be a great improvement to the field of merging documents. However, further work is needed to create writing samples in order to train the classifier to get the appropriate result.

Revision could be done by handwriting on soft or hard copy of the original document. Designing software to create an electronic copy that contains the changes that were made by hand, using natural language processing to distinguish between the changes and the comments to avoid including the comments in the context of the writing, would be very helpful to the user using the document comparison tool, especially with documents that have a large number of handwritten corrections.

References

[1] P. B. Lowry, A. Curtis, and M. R. Lowry, "Building a taxonomy and nomenclature of collaborative writing to improve interdisciplinary research and practice," *J. Bus. Commun.*, vol. 41, no. 1, pp. 66–99, 2004.

[2] S. B. Heard, *The Scientist's Guide to Writing: How to Write More Easily and Effectively throughout Your Scientific Career*. Princeton University Press, 2016.

[3] M. Miller, *Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online*. 2008.

[4] A. Kittur, B. Suh, B. A. Pendleton, and E. H. Chi, "He says, she says: conflict and coordination in Wikipedia," *ACM Conf. Hum. Factors Comput. Syst.*, pp. 453 – 462, 2007.

[5] J. Birnholtz and S. Ibara, "Tracking Changes in Collaborative Writing: Edits, Visibility and Group Maintenance," *Proc. ACM 2012 Conf. Comput. Support. Coop. Work*, pp. 809–818, 2012.

[6] S. Otte, "Version Control Systems," *Comput. Syst. Telemat. Inst. Comput. Sci. Freie Univ. Berlin, Ger.*, 2009.

[7] K. Hinsen, K. Läufer, and G. K. Thiruvathukal, "Essential tools: Version control systems," *Comput. Sci. Eng.*, vol. 11, no. 6, pp. 84–91, 2009.

[8] J. A. E. Garc\ia, "Software Development and Collaboration: Version Control Systems and Other Approaches," 2011.

[9] C. Thao and E. V Munson, "Version-aware XML documents," in *Proceedings of the 11th ACM symposium on Document engineering*, 2011, pp. 97–100.

[10] C. Thao and E. V. Munson, "Using versioned trees, change detection and node identity for three-way XML merging," *Computer Science - Research and Development*, 2014.

[11] S. M. Coakley, J. Mischka, and C. Thao, "Version-Aware Word Documents," in *Proceedings of the 2nd International Workshop on (Document) Changes: modeling, detection, storage and visualization*, 2014, p. 2.

[12] M. Pandey and E. V Munson, "Version aware libreoffice documents," in *Proceedings of the 2013 ACM symposium on Document engineering*, 2013, pp. 57–60.

[13] M. Stoerzer, B. G. Ryder, X. Ren, and F. Tip, "Finding Failure-inducing Changes in Java Programs Using Change Classification," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2006, pp. 57–68.

[14] S. Kim, E. J. Whitehead, and Y. Zhang, "Classifying software changes: Clean or buggy?," *IEEE Trans. Softw. Eng.*, vol. 34, no. 2, pp. 181–196, 2008.

[15] B. Fluri and H. C. Gall, "Classifying change types for qualifying change couplings," in *IEEE International Conference on Program Comprehension*, 2006, vol. 2006, pp. 35–45.

[16] Deltaxml.com, "Overview | DeltaXML DITA Merge." 2016.

[17] J. . Daxenberger and I. . b Gurevych, "Automatically classifying edit categories in wikipedia revisions," in *EMNLP 2013 - 2013 Conference on Empirical Methods in Natural Language*

Processing, Proceedings of the Conference, 2013, pp. 578–589.

[18] F. Zhang and D. Litman, “Annotation and classification of argumentative writing revisions,” in *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications*, 2015, pp. 133–143.

[19] F. Zhang and D. Litman, “Sentence-level rewriting detection,” *ACL 2014*, p. 149, 2014.

[20] T. P. Ping, K. Verspoor, and T. Miller, “Structural Alignment as the Basis to Improve Significant Change Detection in Versioned Sentences,” in *Australasian Language Technology Association Workshop 2015*, p. 101.

[21] N. Whitaker, “Understanding Changes in n-way Merge: Use-cases and User Interface Demonstrations,” in *Proceedings of the 2nd International Workshop on (Document) Changes: modeling, detection, storage and visualization*, 2014, p. 4.

[22] J. Zhang and H. V Jagadish, “Revision provenance in text documents of asynchronous collaboration,” in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, 2013, pp. 889–900.

[23] Softinterface.com, “Document Comparison and Document Conversion Software Tools from SoftInterface.” 2016.

[24] Scootersoftware.com, “Scooter Software: Home of Beyond Compare.” 2016.

[25] J. Eibl, “KDIFF3 - HOMEPAGE,” *Kdiff3.sourceforge.net*, 2016. [Online]. Available: http://kdiff3.sourceforge.net/.

[26] “WinMerge,” *Winmerge.org*. 2016.

[27] H.-C. E. Kim and K. S. Eklundh, “Reviewing practices in collaborative writing,” *Comput. Support. Coop. Work*, vol. 10, no. 2, pp. 247–259, 2001.

[28] “No Title.” .

[29] M. Lutz, D. Ascher, and F. Willison, *Learning python*, vol. 2. O’Reilly, 1999.

[30] A. Sweigart, *Automate the Boring Stuff with Python: Practical Programming for Total Beginners*. 2015.

[31] “python-docx 0.8.5 documentation,” *Python-docx.readthedocs.org*, 2016. [Online]. Available: http://python-docx.readthedocs.org/en/latest/.

[32] “wxPython,” *Wxpython.org*. 2016.

[33] C. Precord, *wxPython 2.8 Application Development Cookbook*. Packt Publishing Ltd, 2010.

[34] H. Sharp, P. Jenny, and Y. Rogers, “Interaction design:: beyond human-computer interaction,” pp. 25–29, 2007.

[35] D. Stone, C. Jarrett, M. Woodroffe, and S. Minocha, *User interface design and evaluation*. Morgan Kaufmann, 2005.

[36] “THE DANGERS OF RELYING ON SPELL CHECK AND GRAMMAR CHECK,” *ServiceScape*. [Online]. Available: https://www.servicescape.com/article.asp?cid=93325. [Accessed: 15-Jun-2016].

Appendix A

IRB Protocol Number: 16.293
IRB Approval date: March 21, 2016

Tasks for study entitled:

Evaluation of an interface for document merging using a language analysis oracle

Please complete the following tasks as you can. If you feel uncomfortable doing any of the tasks, please tell us and go to the next task. In our record you will be referred as Subject: ___ (using random letter)

When you are done writing your research paper, send it to two of your professors. Each professor will make some corrections on your writing and send it back to you. Then you will have three versions of your paper, one with your original writing and the other two with corrections by your professors. The design interface will show the three versions. By using the features in the interface, perform the following two set of tasks on two different given samples:

Tasks Set 1:

1. Find a correction that made in either version 1 or version 2 that might change the meaning of the original content, and reject that change.
2. Try to find a grammar correction that has been made differently in both revised versions and choose the correction from revised version 2.
3. Find a spelling correction that was made only in revised version 2 and accept it.
4. Try to find a change that only made on format and get more details about it.
5. Try to reject the format change that you found in task 4.
6. Find a location where three kinds of changes were made in the same place, and accept the correction from either version 1 or version 2.
7. For **all spelling modifications**, accept the spelling corrections you think are more suitable.

Tasks Set 2:

1. Try to find a location where three kinds of changes were made in the same place, and accept the correction from either version 1 or version 2.

2. find a correction that made in either version 1 or version 2 that might change the meaning of the original content, and accept that change.

3. find spelling correction only that has been made differently in both revised versions and choose the correction from revised 2.

4. Try to find change that only made on format and get more details about it.

5. Try to reject the format change that you found in task 4.

6. Try to find a grammar correction that has been made only in revised version 2, and accept it.

7. Now, for **all grammar corrections**, accept the changes you think are a good fit.

8. Finally save the merged result on the desktop, and call it “May 9 result”

Appendix B

Post-study questionnaire for study entitled:

Evaluation of an interface for document merging using a language analysis oracle

Based on your experience of using the designed interface of documents merging tool, please answer the following questions as completely as you can. If you feel uncomfortable answering any of the questions, please skip that question. Your name will not be recorded. In our records you will be referred by a random letter

1. Please rate your satisfaction with the following aspects of the interface.

	Very Dissatisfied	Dissatisfied	Neutral	Satisfied	Very Satisfied
a) Overall ease of use	□	□	□	□	□
b) Ease of learning	□	□	□	□	□
c) Intuitiveness	□	□	□	□	□
d) Informativeness (i.e. does the interface provide enough information for the tasks you performed?)	□	□	□	□	□

2. To what extent do you agree or disagree with the following statement: The change classifications (font, grammar, spelling, and editing) that were used in this interface would help you better understand the changes that were made in both revised versions?
    - o Strongly agree
    - o Agree
    - o Somewhat agree
    - o Neither agree nor disagree
    - o Somewhat disagree
    - o Disagree
    - o Strongly disagree

3. To what extent do you agree or disagree with the following statement: The change classifications would help you to give your primary attention to the most important changes, especially in a large document?
    - Strongly agree
    - Agree
    - Somewhat agree
    - Neither agree nor disagree
    - Somewhat disagree
    - Disagree
    - Strongly disagree

4. What features of the interface did you find useful?

5. What features did you find confusing? Do you have any suggestion to improve them?

6. Other comments: