Software Quality Assurance

Smart Undo Capability

(Added to a java text editor)

COMP 5541

Tools and Techniques for Software Engineering

Assignment 3

Due date: April 1, 2021

Team members

First name, Last name	Student ID	Contribution
Eric Ting-Kuei, CHOU	40070424	GUI coding, code testing, report editing
Adnan, ALI	40181614	Technical review, report review
Seyed Homamedin, HOSSEINI	40137467	Coding, testing, report review
Mohammed, ZAFAR	40152488	Component level and Integration tests

Contents

Technical Review

Introduction

The primary goal of this project is to develop an extension to a text editor, which manages edits in a systematic order. Using this program, users can perform and manage their edits per document using the Smart Undo function. However, prior to moving forward with the project implementation, a clear expectation needs to be set up to encourage efficient and collaborative team dynamics within Team #3. This required a strategy to distribute the workload while being conscious of each member's skill set. Since the technology was already decided by our stakeholders, Babita Bashal and R. Jayakumar, an efficient method to test the code was required in order to approach the implementation of the Smart Editor successfully.

Current Iteration

For this iteration, our main focus has been mainly about the implementation of the Smart Text Editor extension. In other words, we could say this is the engineering stage while the prior stages we were focused on preparation and gathering our data to back up our reasoning of why we should implement something the way we decided to.

This new iteration involves some important steps worth mentioning. One important aspect we focused on was making sense of the original source code and beginning to make clear on how to apply it to our project. Also, during the process where each function had to be implemented, Team #3 had to find different methodologies to actually make the smart undo function work. For example, there could have been different ways

to allow the user to choose each previous edit made on the basic text editor, but we had to find one that would meet the design requirements while also allow the user to implement it naturally and execute it successfully. To do this we had to identify what were the possible solutions or concepts we wanted to implement, weight out the pros and cons of each, choose one, and then implement it. The process has been documented in this report.

In addition to the extensions of the Smart Text Editor, the UI was another main focus throughout this iteration. As we learned from our stakeholder meetings with R. Jayakumar, the UI is a very important aspect that allows the user to understand how the software works and it facilitates the process of understanding how to use the software. As the GUI was being made, our group actively provided feedback to improve our GUI experience while saving us time before reaching a final product/concept.

Lastly, aside from the text editing functions, we also realized we needed to dedicate time to create a validator and parser to successfully handle the different objectives included in the operations and functions we created. This was one of the most tedious but one of the most important things to focus on throughout this iteration.

Other aspects that were not as time consuming as the tasks previously mentioned, but still important, were the implementation of error messages, continuing to review each other's work in form of Smoke-Testing or Regression-Testing for efficient synergy between components, resolving any remaining edge cases and checking with our stakeholders to review our project requirement and design to further reduce the cost of defect.

Goals

To summarize the main goals, we set out to complete in this iteration:

- Parser creation
- Validator creation
- Implementation selection of our functions
- GUI implementation
- Quality Assurance (Smoke/Regression Testing)
- Error message creation
- Resolving edge cases

Team organization and collaboration patterns

Expectations

The first order of business of our team discussion was to get everyone's expectations regarding team dynamics. This allowed us to highlight our strengths and compensate for each other's weaknesses. The expectations agreed upon were:

- The team would be a supportive place where members can develop i.e., if someone does not know something then the team would be able to teach them. This would be done through active communication and would be a perfect fit for our Waterfall-process Model.
- All tasks must be completed before the given deadline.
- If a particular module reaches a task-artifact cycle, then it MUST be communicated in a timely manner so other team members can provide support and readjust the strategy to allow for a timely delivery.
- There would be at least a follow-up session after each team meeting to allow for team cohesion.
- Workload will be distributed fairly, evenly, and considering each member's skill level.

Meeting Schedule

The team agreed to meet at least once a week for 2 hours. We were also in constant contact through Google Meets and we kept all our recent pushes on Github and Google Doc to stay up to date with advancement of code base and accompanying documents. Details of meeting minutes were shared on Google docs after every meeting by the designated secretary for the meeting. Finally, the draft documentation for each iteration was updated via Google Doc to create our final report with the correct formatting.

Strategy

In order to develop a working Smart Text editing prototype in the upcoming weeks, the team has come up with a strategy to be prepared for the challenges that we might face. This strategy involves documenting each feature implemented in the software, the technologies selected to develop the text editor, ideas and algorithms used to develop the features and selected GUI, and the tasks to be allocated to each team member based on their strengths. With this well-advised plan of action, under the mentorship of Prof. Jayakumar, we are more likely to increase the team dynamic and collaborate efficiently to meet the project deadline.

Choice of technology

The source code for a generic text editor was already provided to the team by our stakeholder Babita Bashal to meet our software needs. Therefore, Java was chosen as our main programming language and allowed each team member to jump right into coding as that is where the team expertise lied. This was also advantageous for us

since Java has many libraries that we would be able to use for unit testing (Junit), GUI development (JavaFX) and code documentation (Javadoc) that would be a crucial part of our project. Additionally, the object-oriented design fit of Java fits perfectly with our Model-View-Controller design pattern approach to the Smart Text Editor.

In order to coordinate with each other outside of regular meeting hours, Team #3 decided to use Google Meets as our primary communication tool. As it can be accessed on almost any device and is a reliable and easy to use software. It was easily synced with our Google Calendars and would constantly update us regarding meeting time and agenda to help us prepare for our discussions and scheduling.

For version control software we used Git as it was simple enough to use and had tons of documentation support for our needs. Furthermore, it was to allow us to incorporate our MVC model build perfectly as we could work on multiple modules independently and keep track of changes made to each one for our Regression-Testing.

Task assignment

Ever since our second team meeting, we started going over all the deliverables and made a list of all the things that needed to be done. We quickly saw that this project had a high dependency between tasks and was basically divided into 3 categories.

1. Information gathering
2. Consolidation reports/documentation
3. Coding and Prototyping

A cursory evaluation done by our stakeholders of the workload vs deadline showed that we would not be able to deliver all three tasks at the same time. Therefore, a priority

matrix was assigned to each of our tasks with a simple rule that higher priority tasks would be completed before lower priority ones, which aligned perfectly with our Waterfall-process model.

We reasoned that each task would be divided into a deliverable and priority would be assigned to the upcoming developer-stakeholder meeting. Even though we all had a good idea of what our Smart Text editor would be like, we also knew better than to make a product only for us developers. Therefore, we required information from stakeholders and potential clients (users) as soon as possible so we could align our efforts based on the requirement-report. This step was extremely critical and urgent that we put all 4 members of Team #3 on it with highest priority.

We reasoned that each task would be divided into a deliverable and priority would be assigned to the upcoming developer-stakeholder meeting. Even though we all had a good idea of what our Smart Text editor would be like, we also knew better than to make a product only for us developers. Therefore, we required information from stakeholders and potential clients (users) as soon as possible so we could align our efforts based on the requirement-report. This step was extremely critical and urgent that we put all 4 members of Team #3 on it with highest priority.

We then summarized and consolidated the data gathered from our stakeholders and potential clients and extracted the key features each user wanted. Understanding the key features from each client based on their expected usage (GUI use, grouping convenience) was the goal of our requirement gathering report and our approximation of what the market wants from our Smart Text Editor.

Once requirement gathering was finalized, one team member was tasked with coming up with our testing strategy, while the rest were assigned to researching algorithms needed for implementation of basic text editing functions and trivial Smart Undo functions. Since some functions had multiple algorithms that varied in complexity (usage of lists or vectors for edit instructions) we decided that simplicity would be the key criteria for our second iteration in the choice of algorithms since the deadline was tight. The prototype exhibited proper usage and met all design requirements, we decided not to reevaluate this strategy for the current iteration or for the final deliverable. This now leads us to our last priority of implementation of the Smart Undo Function and coding the basic prototype into our final product. The roles divided among our team members are as follows:

Table 1: Take Priority Matrix

Name	**Priority 1**	**Priority 2**	**Priority 3**
Adnan	SE Artifacts	Macro Architect	Hash tables
Eric	GUI	Micro Architect	Binary Search Trees
Ershad	Validator	Binary Search Trees	Micro Architect
Hoomam	Parser	Micro Architect	Macro Architect

Source Code Review Results

For testing purposes, Team #3 decided to write-up as many test cases as possible for various scenarios in order to cover all potential issues (See section 2). These test cases have been ongoing from the early stages of project-design and were implemented in a

Smoke-Testing implementation to reduce the risk of having insurmountable bugs later on. Taking this into consideration, Team 3#‘s Test-Driven-Development strategy has always been to write unit tests before the functions were complete and imposed a rule that prior to the push made to our master branch on Github, all valid tests must be passed by our validator. This would ensure that whenever someone pulled a prototype from the master branch, they would not be delayed by fixing existing bugs prior to their own task.

Even with a self-validation setup imposed on the team, the team decided to test each other’s code as a paramount importance to the project. This would not only help in tracking down existing bugs but would also ensure that there were no conflicts of interest in maintaining a testing-caliber across team members. The source code was peer reviewed by at least 2 team members and the associated comments were provided to the reviewee.

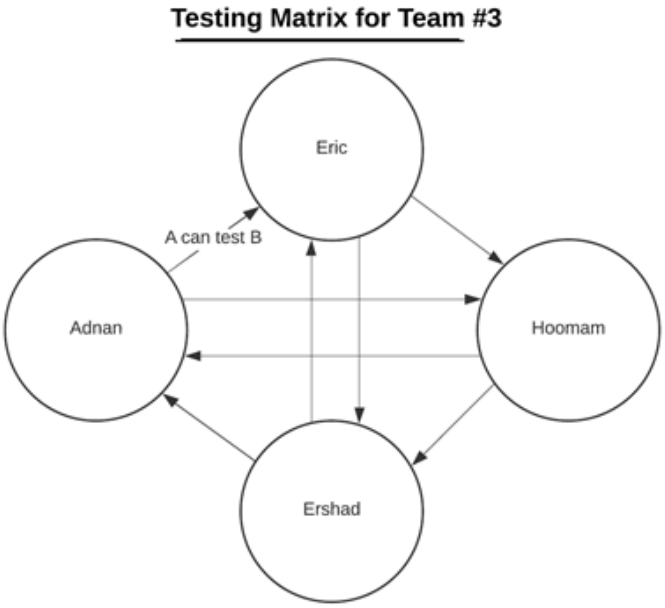

Table 2: Summary of code reviews and comments provided

Reviewer	**Reviewee**	**Comments**
Adnan	Eric	● Use K & R formatting. ● Try to make functions shorter.
Eric	Hoomam	● Use detailed comments. ● Use better variable names.
Hoomam	Ershad	● Do not leave commented code. ● Comment on what a function does rather than what it is.
Ershad	Adnan	● Some functions break for edge cases. ● Try to shorten comments while maintaining clarity.

Testing

Due to this being such a large and and a more complex project than any of us had done before, we implemented Smoke and Regression Testing prior to any uploads in our repository. This was done via the debugger tool in our IntelliJ IDEs (version 2020.3) to keep consistency in our codes. This was extremely important to have a better understanding of what was going on in our code during the initial prototype, while keeping the downstream functionality stable and removing bugs prior to any further testing. The debugger was often used to check the flow of Parser and ExpressionValidator since they were the most error prone parts of the code. This was greatly appreciated in checking for edge cases as the number of inputs were immense and loops were causing out-of-bound errors.

Debugger was also very useful in examining group creations for our Smart Undo Functions, where we wanted to create different Array lists based on sudden changes in an existing group. Since we used hash tables, our inputs sometimes gave us poor

precision when introducing new edits/newGroup() within an existing set. This was isolated by the debugger at a very early stage and allowed us to improve our algorithm for these edge cases.

Unit Testing

Once we had implemented our Undo Functions, we needed a way to reliably and quickly test a large range of inputs against them to validate our expected outcomes. In order to accomplish that, each member of the team tried to use the code from a user perspective and came up with several unit tests spanning ranges of input both inside and outside of each keystroke generated. We also had to determine whether the original source code provided by our stakeholders met the universal acceptable criteria of a “generic text editor” to allow for the best user experience. This was prioritized by incorporating expected results from our Smart Functions into the text frame incorporated in the text editor to have a sufficient and realistic accuracy of what a user might experience. This was made possible by the Junit library, which was integrated into our IDEs to provide a quick way to write, compile and visualize results effectively.

A brief overview of our unit testing examples is listed below. We wrote each function with various keystrokes in mind and a combination of outliers like sticky keys and touch screen. These unit tests proved to be useful in isolating individual function outside of the overall application’s context to validate whether or not the edits created, or group would not only be recorded but provide an acceptable outcome to the user. This also allowed us to diagnose issues or inaccuracies in our code editor which would stall particular code blocks or require extreme processing by making inefficient iteration cycles.

Test cases for User Interface Module

TC ID	Type	Test Case description	Expected Result
1	**Smart Undo** **Menu List**	1. Open the Text Editor 2. Access the "**Smart Undo"** Menu from the menu bar and check if has following options in the list ● Undo Prev (ctrl +z) ● View/ Undo Edit(s) ● Start a New Group ● Delete Edit(s) ● Delete Group (s) ● Reset Undo	All menu options should be listed in the smart Undo menu.
2	**View/ Undo** **Edit(s)**	1. Open the Text Editor 2. Without entering any text in the editor, access the "**View/ Undo Edit(s)"** item from the "**Smart Undo"** Menu	View/ Undo Edit(s) should have the following columns 1. Group 2. Line 3. Columns 4. Action 5. Character
3	**Delete Edit(s)**	**1.** Open the Text Editor **2.** Without entering any text in the editor, access the **Delete Edit(s)** item from the "**Smart Undo"** Menu	Delete Edit view should have the following columns 1. Group 2. Line 3. Columns 4. Action 5. Character
4	**Delete Group**	**1.** Open the Text Editor **2.** Without entering any text in the editor, access the **Delete Group** item from the "**Smart Undo"** Menu.	Delete Group view should have the following columns 1. Group 2. Start 3. End 4. nEdits
5	**Start a New** **Group**	1. Open the Text Editor ● Without entering any text in the editor, access the "**Start a New Group"** item from "**Smart Undo"** Menu.	Clicking "Start a New Group" should bring about a message and decision box popup screen displaying the following message and action buttons "New Group will start on the next line. Are you sure ?" Three action buttons should be shown in the message box window 1. Action 2. No 3. Cancel
6	**Reset Undo**	1. Open the Text Editor 2. Without entering any text in the editor, access the "**Reset Undo"** item from "**Smart Undo"** Menu.	Clicking "Reset Undo" should bring about a message and decision box popup screen displaying the following message and action buttons "Are you sure you want to clear all Edits ?" Three action buttons should be shown in the message box window 1. Action 2. No 3. Cancel

Test cases for the Module that registers / records the Edits

Test Case ID	Test Type	Test Case Description	Expected Result
1	**Functionality test** ***This test will ensure edit actions are recorded by the Smart Undo Capability***	1. Open the Text Editor 2. Navigate to the "Smart Undo" Menu and select the option "Start a New Group" 3. Check if the user is prompted with a message cum decision box screen Hit "Yes" in the message box.	Hitting "Yes" button in the Message / Decision box screen, should take the user back to the text editor, with cursor on the Line 1.
		In the Text editor, type the following text from Line 1 column 1 onwards **Group 0 - 12345, !@#$%,** **Next line**	User should be able to add text in the editor without any errors or issues
		Access "Smart Undo" Menu and select the option "View /Undo Edits"	This view should list all edits in "Bottom to top order" (starting with the First edit at bottom of the list and so on.)
		1. Navigate to the "Smart Undo" Menu select the option "Start a New Group" 2. Check if the user is prompted with a message cum decision box screen Hit "Yes" in the message box and the user should be taken to a new line in the text editor.	Hitting "Yes" in the message box should take the user back to the text editor, on a new line. The functionality should always start the New Group should from a new line, even if the user is already on a new line with no text.
		In the new line (Line 3), from column 1 onwards type the following text **Group 1 - 12345, !@#$%,** **Next line**	User should be able to enter the text in the editor without any errors / issues
		Navigate to the "Smart Undo" Menu select the option "View /Undo Edits"	This view should list all edits of both group 0 and group 1 in "Bottom to top order" (starting with the First edit at bottom of the list and so on.)
2	**Data Structure Test**	1. Open the Text Editor 2. Type a few alphabetical characters in the editor 3. Check if the View / Undo Edits screen is showing each alphabetical character as a separate edit action.	The smart undo functionality should be able to record each alphabetical character as an edit action.
3		1. Type a few Numeric characters in the editor 2. Check if the View / Undo Edits screen is showing each character as a separate edit action.	The smart undo functionality should be able to record each numerical character as an edit action.

Test Case ID	Type	Test Case description	Expected Result
4	**Data Structure Test**	1. Type a few special characters in the editor 2. Check if the View / Undo Edits screen is showing each character as a separate edit action.	The smart undo functionality should be able to record each special character as an edit action.
5		1. Type some space character, new line characters along with numbers and alphabets. characters in the editor 2. Check if the View / Undo Edits screen is showing each character as a separate edit action.	The smart undo functionality should be able to record new lines, back spaces, and space character as an edit action.
6		Check if the View/ Undo Edits screen is displaying edits in the Bottom to top order of sequence. (starting with the First edit at bottom of the list and so on).	
7		1. Access the Delete Edits menu option and delete some of the edits. 2. Access the View Delete menu option and check if the edits deleted in the previous step should no longer appear in the list of edits eligible for deletion. 3. Check if the order is still preserved in the view (Bottom to Top order)	
8	**Boundary Conditions test / Error Handling test**	1. Open the Text Editor 2. Start a new group and type in the editor more than 1000 characters. 3. Check if the user is prompted by a message box that says that a new group will be started hereafter. 4. After clicking " Ok", user should be taken back to the editor on a new line	Functionality should not allow the user to record more than 1000 edit actions per group.
9	**Error Handling Test**	1. Open the Text Editor 2. Without entering any text, access the Reset Undo menu item from the Smart Undo Menu. 3. Check if any message box is displayed to the user	A message box with an OK button should be appear on the screen. The message should read " There are no edits to undo". Hitting the "OK" Action button should take the user back to the text editor.

Test cases for the Module that can undo edits, delete edits and delete groups

Test Case ID	Test Type	Test Case Description	Expected Result
1	**Functionality test** ***This test will ensure if the edits can be undone, deleted and the groups can be deleted.***	1. Open the Text Editor 2. Navigate to the "Smart Undo" Menu and select the option "Start a New Group" 3. In the Text editor, type the following text from Line 1 column 1 onwards **Abcde 0 - 12345, !@#$%,** **Next line**	User should be able to add text in the editor without any errors or issues
		Navigate to the "**Smart Undo" Menu select the option** "View /Undo Edits" Undo some edits	Once the user has undone the edit from the " View / Undo Edits screen", the edit should be undone in the text editor. Undone edit should be removed from View /Undo Edits.
		Navigate to the "Smart Undo" Menu and select the option **"Delete Edits"** Delete some edits.	Delete Edits view should be opened listing all the edits User should able to successfully delete the edits. Once deleted, the Edits should no longer appear in the **Delete Edits** view
		Navigate to the "Smart Undo" Menu and select the option **"Delete Group"** Delete some edits.	Deleted group should be removed from the view.
2	**Data Structure Test**	1. Open the Text Editor 2. Type a few alphabetical characters in the editor 3. Check if the **Delete Edits** screen is listing each character as a separate edit action in the Bottom to Top order	
3		Check if the **Delete Group** screen is listing each character as a separate edit action in the Bottom to Top order	

Integration Tests

Integration Testing Strategy

Once we had ironed out most of the issues pertaining to individual functions regarding our Smart Undo requirements, we integrated those in the original text editor with a simple user interface, expression validator and an expression parser to validate that all modules were working together properly. This integration test allowed us to experiment with scenarios where unit tests could not be efficiently written. A list of expected outcomes was compiled prior to testing (shown in table below) which allowed us to explain the expected functionality of the module and how it would affect the entire system. As each module was coded independently, the task for coming up with the list lied with the coder responsible.

As an extension to a text editor, our input relied on converting a valid string, through a correct smart expression, into the expression parser. This was tested through the **Expression Validator** where nonsensical strings, along with integers and char, were used as inputs for testing functionality. The expected exceptions were designed to throw error messages to the user (as shown in table below) and most importantly, were not passed to the parser. If any of these did not catch an error and sent an unexpected string to the parser, we would debug it and re-test until the issue was fixed.

Finally, we tested the actual group creation, editing, deletion and add/remove functions through the **Expression Parser**. We used online open-source MOOC editors to compare results of our expressions to the real answers. This would be done by having long inputs that used multiple functions (which might include error messages too) and document the behavioral patterns in our code vs MOOC editors. Testing our parser with

these pseudo-random patterns allowed us to find several issues that were not caught in our unit tests. This was vital in zeroing in on issues that would not be found unless the whole software was consolidated and each layer would have to be tested individually, requiring a lot of time and manpower which our team lacked. It also allowed us to visualize the system harmony from the users perspective and they would have the final review of our code.

Ultimately a Top-Down approach was identified in the meetings and technical reviews as the most appropriate method for the Integration testing. Modules that are overarching would be tested first followed by other modules that are lower in the hierarchy. At the top is the module that maps over to the View / Undo Edits View from where the data is shared to other modules and allowed to be modified. If the High levels modules are working as expected, only then the testing lower level modules should be performed.

Integration Test Cases

Test Case ID	Test Scenario Description	Test Case	Expected Result
1	**Validate if the edits are being recorded in different user groups correctly by the Smart Undo Capability** This test case will ensure editor is successfully able to : 1. Record edit actions in different groups. Each character added in the text editor should be considered by the Smart Undo Functionality as an Edit Action. 2. User is able view all the edits from the View / Undo Edit screen 3. User is able to undo an edit from the View / Undo Edit screen **Note** - *Space, special characters and even the next line character should be considered as a separate Edit action, besides the alphabetical characters*	1. Open the Text Editor 2. Navigate to the "Smart Undo" Menu and select the option "Start a New Group" 3. Check if the user is prompted with a message cum decision box screen Hit "Yes" in the message box.	Upon hitting the "Yes" button in the Message / Decision box screen, the user should be taken back to the text editor, with the cursor on Line 1.
		In the Text editor, type the following text from Line 1 column 1 onwards **Group 0 - 12345, !@#$%,** **Next line**	User should be able to add text in the editor without any errors or issues
		Navigate to the "Smart Undo" Menu select the option "View /Undo Edits"	This view should list all edits in "Bottom to top order" (starting with the First edit at bottom of the list and so on.)
		1. Navigate to the "Smart Undo" Menu select the option "Start a New Group" 2. Check if the user is prompted with a message cum decision box screen Hit "Yes" in the message box and the user should be taken to a new line in the text editor.	Hitting "Yes" in the message box should take the user back to the text editor, on a new line. The functionality should always start the New group should from a new line, even if the user is already on a new line with no text.
		In the new line (Line 3), from column 1 onwards type the following text **Group 1 - 12345, !@#$%,** **Next line**	User should be able to enter the text in the editor without any errors / issues
		Navigate to the "**Smart Undo" Menu select the option** "View /Undo Edits" Undo an edit from the Group 0 and another from Group 1	Once the user has undone the edit from the " View / Undo Edits screen", the edit should be undone in the text editor. Undone edit should be removed from View /Undo Edits.

Test Case ID	Test Scenario Description	Test Case	Expected Result
2	**Validate the deletion of edits** This test case will ensure if: 1. The edits of different groups are getting deleted 2. The View / Undo Edits view is refreshed after the deletion of edits 3. The Delete edits view is refreshed after the deletion of edits 4. The Delete Group view is refreshed after the deletion of edits	1. Open the Text Editor 2. Navigate to the “Smart Undo” Menu and select the option **“Delete Edits"**	Delete Edits view should be opened listing all the edits
		Delete some edits from Group 0 and also Group 1, as per test data table 3.2	User should able to successfully delete the edits. Once deleted, the Edits should no longer appear in the **Delete Edits** view
		Navigate to the “Smart Undo” Menu and select the option “**View /Undo Edits**”	Deleted edits should be removed from the list. And the order in which the edits are listed should remain intact. Bottom to top order (starting with the First edit at bottom of the list and so on)
		Navigate to the “Smart Undo” Menu and select the option **“Delete Group/s**”	View should display correct information as it relates to the number of edits.
3	**Validate the deletion of group/s** This test case will ensure if: 1. The group is getting deleted 2. The deleted group is then removed from the View / Undo Edits view 3. The deleted group is removed from the **Delete Group** view 4. The deleted group is removed from the **Delete Edits** view	1. Open the Text Editor 2. Navigate to the “Smart Undo” Menu and select the option **“Delete Groups"**	Delete Edits view should be opened listing all the edits.
		Delete Group 0	Deleted group should be removed from the Delete Groups view.
		Navigate to the “Smart Undo” Menu and select the option “**View /Undo Edits**”	Deleted group should be removed from the view.
		Navigate to the “Smart Undo” Menu and select the option **“Delete Group/s**”	Deleted group should be removed from the view.
		1. Open the Text Editor 2. Navigate to the “Smart Undo” Menu and select the option **“Delete Edits"**	Deleted group should be removed from the view

Changes to the Original Design

During the technical review exercise, the team encountered requirement defects that were not foreseen in the requirement gathering phase. Only during the development of the code and its review, deeper insight into some use cases was collected, that allowed the team to come up with solutions that would also then lead to design changes. There were two important design changes that were incorporated in the code as a result of technical review and brainstorming of test cases.

Creation of a default group "0" if the user starts editing document without starting a group

From the perspective of the usability, creating a default group was identified as an important feature to ensure edits are always recorded regardless of whether the user is neglectful or not. This feature was identified as a control that will make the Undo Capability really smart and intuitive. The team also considered prompting users in a message box to always create a group before editing anything in the document. However, prompting users with message boxes seem to be more annoying than being user friendly, especially when it cannot be predicted if the user is intending to record the edit actions or not. As and when the user wishes to record and maintain edits, the capability will switch from the default to the user created groups.

Switching to next line if a new group is started by the user

The capability will have to switch to a new line if the user is starting a new group, to ensure that groups are primarily representing one or more paragraphs. The User is

therefore also encouraged to maintain only one group per paragraph. However, the requisite for creating a new group is only the commencement of editing in a newline, but the new line does not necessarily have to start in a new paragraph. By automatically switching the line to a new one when a new group is started will intuitively encourage the user to maintain groups by paragraphs but not for a group of lines. Furthermore, groups will be labelled with be numbered in the integer sequence starting from 0. Management of edits will therefore significantly improve because the user will be able to easily identify, position and locate the edits in the Smart Undo Menu windows performing Undo and Delete edit actions.

Intuitive and smart aspects in the above two features will enhance the overall quality and usability of the Smart Undo Capability.

Annexe

Test Data Table 3.1

View / Undo Edits				
Group	**Line**	**Columns**	**Action**	**Character**
0	1	1	Insert	G
0	1	2	Insert	r
0	1	3	Insert	o
0	1	4	Insert	u
0	1	5	Insert	p
0	1	6	Insert	0
0	1	7	Space	
0	1	8	Insert	-
0	1	9	Space	
0	1	15	Insert	,
0	1	16	Space	
0	1	17	Insert	1
0	1	18	Insert	2
0	1	19	Insert	3
0	1	20	Insert	4
0	1	21	Insert	5
0	1	22	Insert	,
0	1	23	Space	
0	1	24	Insert	!
0	1	25	Insert	@
0	1	26	Insert	#
0	1	27	Insert	$
0	1	28	Insert	%
0	1	29	Insert	,
0	2	30	Insert	n
0	2	31	Insert	e
0	2	32	Insert	w
0	2	33	Insert	l
0	2	34	Insert	i
0	2	35	Insert	n
0	2	36	Insert	e

Test Data Table 3.2

View / Undo Edits				
Group	**Line**	**Columns**	**Action**	**Character**
0	1	1	Insert	G
0	1	2	Insert	r
0	1	3	Insert	o
0	1	4	Insert	u
0	1	5	Insert	p
0	1	6	Insert	0
0	1	7	Space	
0	1	8	Insert	-
0	1	9	Space	
0	1	15	Insert	,
0	1	16	Space	
0	1	17	Insert	1
0	1	18	Insert	2
0	1	19	Insert	3
0	1	20	Insert	4
0	1	21	Insert	5
0	1	22	Insert	,
0	1	23	Space	
0	1	24	Insert	!
0	1	25	Insert	@
0	1	26	Insert	#
0	1	27	Insert	$
0	1	28	Insert	%
0	1	29	Insert	,
0	2	30	Insert	n
0	2	31	Insert	e
0	2	32	Insert	w
0	2	33	Insert	l
0	2	34	Insert	i
0	2	35	Insert	n
0	2	36	Insert	e
1	3	1	Insert	G
1	3	2	Insert	r
1	3	3	Insert	o
1	3	4	Insert	u
1	3	5	Insert	p
1	3	6	Insert	1
1	3	7	Space	
1	3	8	Insert	-
1	3	9	Space	
1	3	15	Insert	,
1	3	16	Space	
1	3	17	Insert	1

1	3	18	Insert	2
1	3	19	Insert	3
1	3	20	Insert	4
1	3	21	Insert	5
1	3	22	Insert	,
1	3	23	Space	
1	3	24	Insert	!
1	3	25	Insert	@
1	3	26	Insert	#
1	3	27	Insert	$
1	3	28	Insert	%
1	3	29	Insert	,
1	4	30	newline	
1	4	30	Insert	N
1	4	31	Insert	e
1	4	32	Insert	w
1	4	33	Insert	l
1	4	34	Insert	i
1	4	35	Insert	n
1	4	36	Insert	e