Final Report

Version 1

April 15, 2021

Smart Undo/Redo for a Text Editor

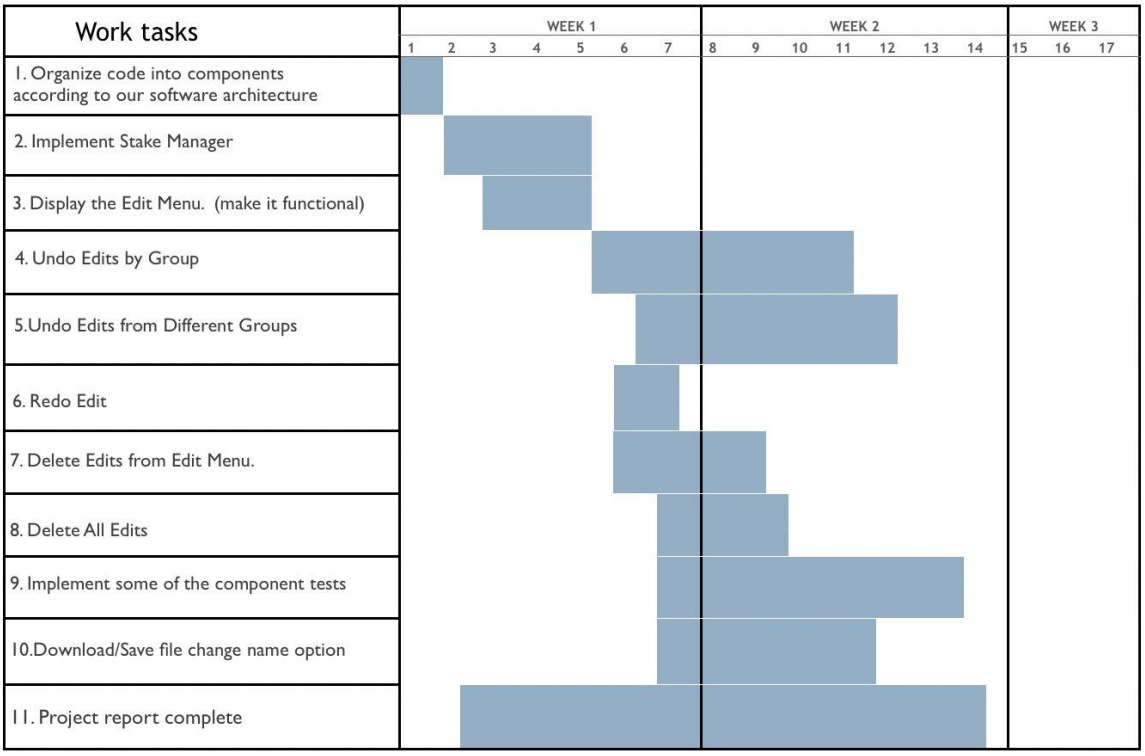
CLS - Team 6
Jacob Lazda 40041526
Jiawei Zeng 40079344
Qichen Liu 40055916
Hualin Bai 40053833
Yifan He 40068616
Lingli Lin 40065176
Jiawei Yao 27197829

Submitted in partial fulfillment for the requirements of COMP354:
Intro to Software Engineering

Schedule for project implementation

Complete Period

Period: 2021.4.1- 2021.4.15



Tasks:

Task 1:

Organize code into components according to our software architecture

Description:

Reorganize the existing code into components and contribute the missing components that are in our software architecture.

Team assignment:

Jacob Lazda

Planned time:

April 1st - April 2nd

Task 2:

Implement Stack Manager

Description:

Based on the component part of design, implement the missing component--stack manager component

Team assignment:

Jiawei Zeng

Planned time:

April 2nd - April 5th

Task 3:

Display the Edit Menu

Description:

Set up a place for showing the user what he did.

Team assignment:

Hualin Bai

Planned time:

April 3rd - April 5th

Task 4:

Undo Edits by Group

Description:

Implement a functionality that allows the user select or choose a specific word, find all the same words and do the undo operation.

Team assignment:

Hualin Bai

Planned time:

April 6th - April 11th

Task 5:

Undo Edits from Different Groups

Description:

Implement a functionality that allows the user select or choose some specific words(from different groups, eg. not the same word), and do the undo operation.

Team assignment:

Hualin Bai

Planned time:

April 6th - April 12th

Task 6:

Redo Edit

Description:

Implement a functionality that allows the user to redo the operations that they undoed.

Team assignment:

Jiawei Zeng

Planned time:
April 6th - April 7th

Task 7:

Delete Edits from Edit Menu

Description:

Implement a functionality that can make the delete the operation from the edit menu that we mentioned above.

Team assignment:

Jiawei Yao

Planned time:
April 6th - April 9th

Task 8:

Delete All Edits

Description:

Implement a functionality that can make the delete all the operations in the edit menu.

Team assignment:

Jiawei Yao

Planned time:
April 7th - April 10th

Task 9:

Implement some of the component tests

Description:

Testing whether the other components are working properly and integrate.

Team assignment:

Qichen Liu, Lingli Lin, Yifan He

Planned time:
April 7th - April 14th

Task 10:

Download/Save file change name option

Description:

Implement a feature that allows users to change the filename as they like when they are saving the files.

Team assignment:

Jacob Lazda

Planned time:

April 7th - April 11th

Task 11:

Project report complete

Description:

Finishing the project report

Team assignment:

Lingli Lin, Yifan He

Planned time:

April 2nd - April 14th

Testing:

Test1:

component: editor

Description: using toBeTruthy() or toEqual() method to check whether the editor component has been created and initialized or not(return true if it is there, otherwise). For testing the integration, we were using the Thread-based testing to validate the output, based on the input.

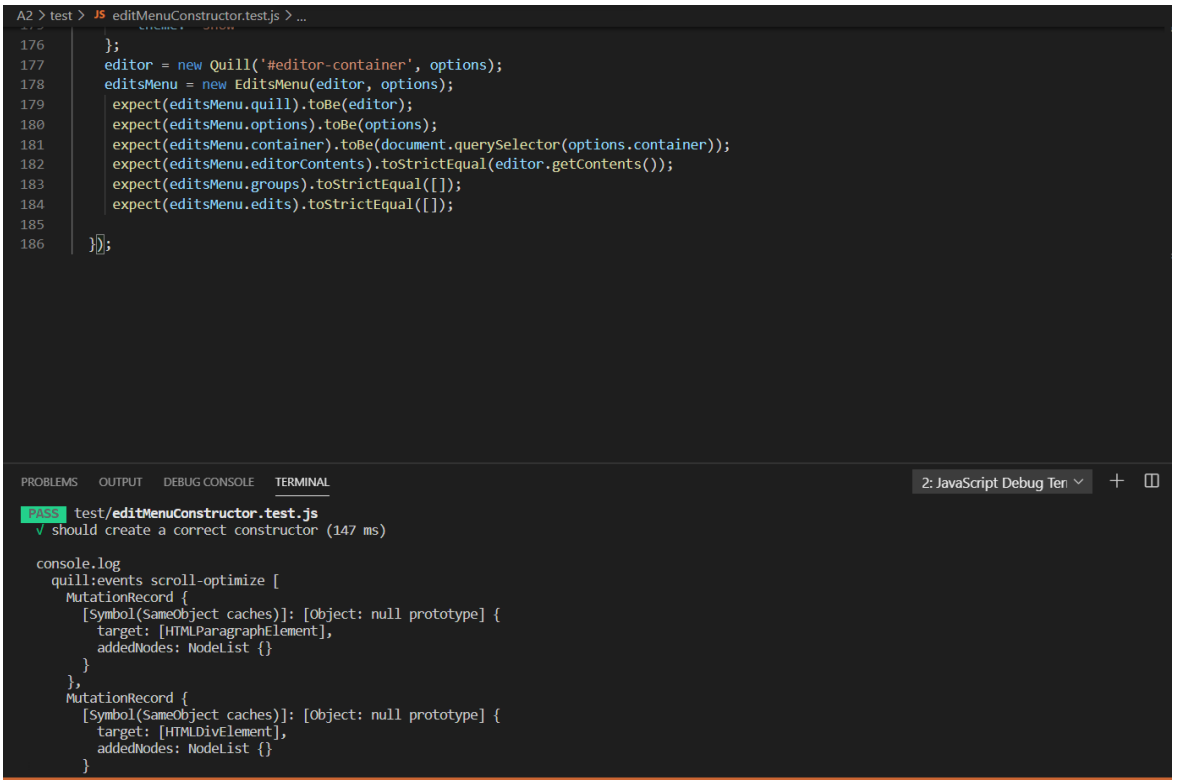
Test2:

component: editorMenu

Description: using toBeTruthy() or toEqual()method to check whether the editorMenu component has been created and initialized inside the editor component or not(return true and pass if it is there, otherwise return false and fail). For testing the integration, we were using the Thread-based testing to validate the output, based on the input. The main components include addToGroups(), editSelected(),updateGroups(),editMenuConstructor(),deleteEdits().

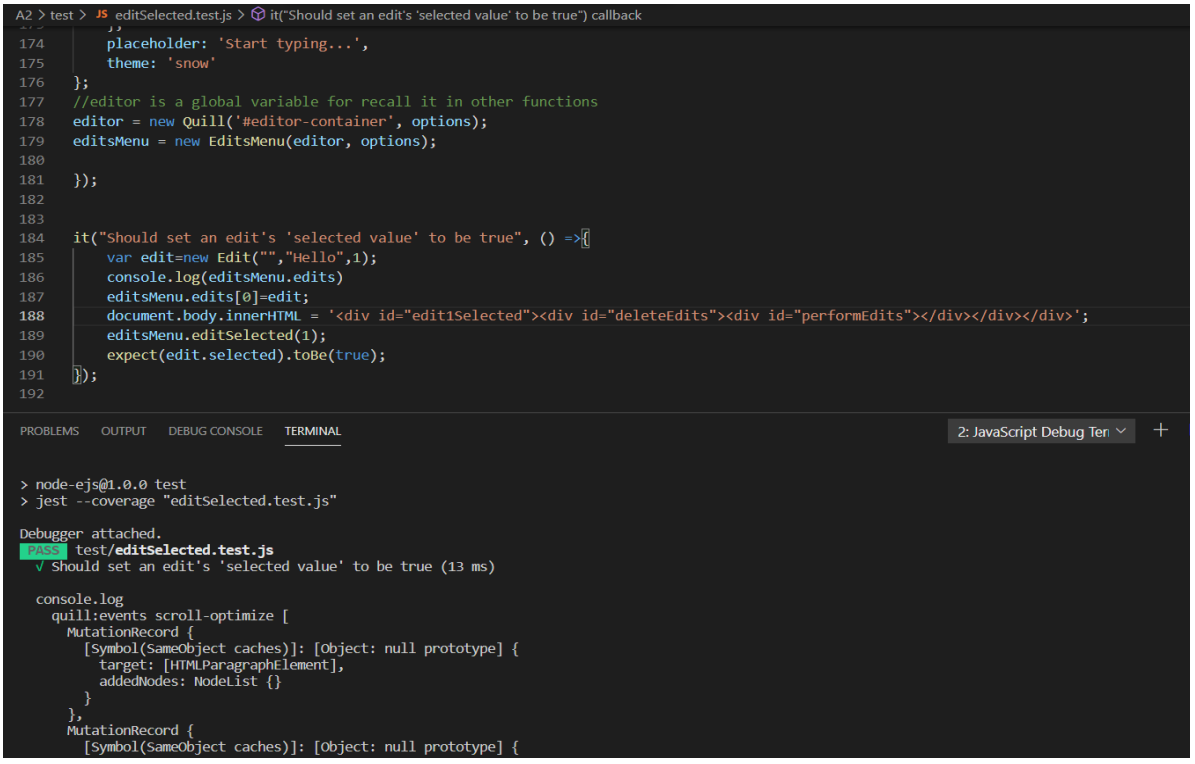
1. editMenuConstructor():

Check whether parameters are passed into the instance by comparing parameters and those corresponding instance variables.



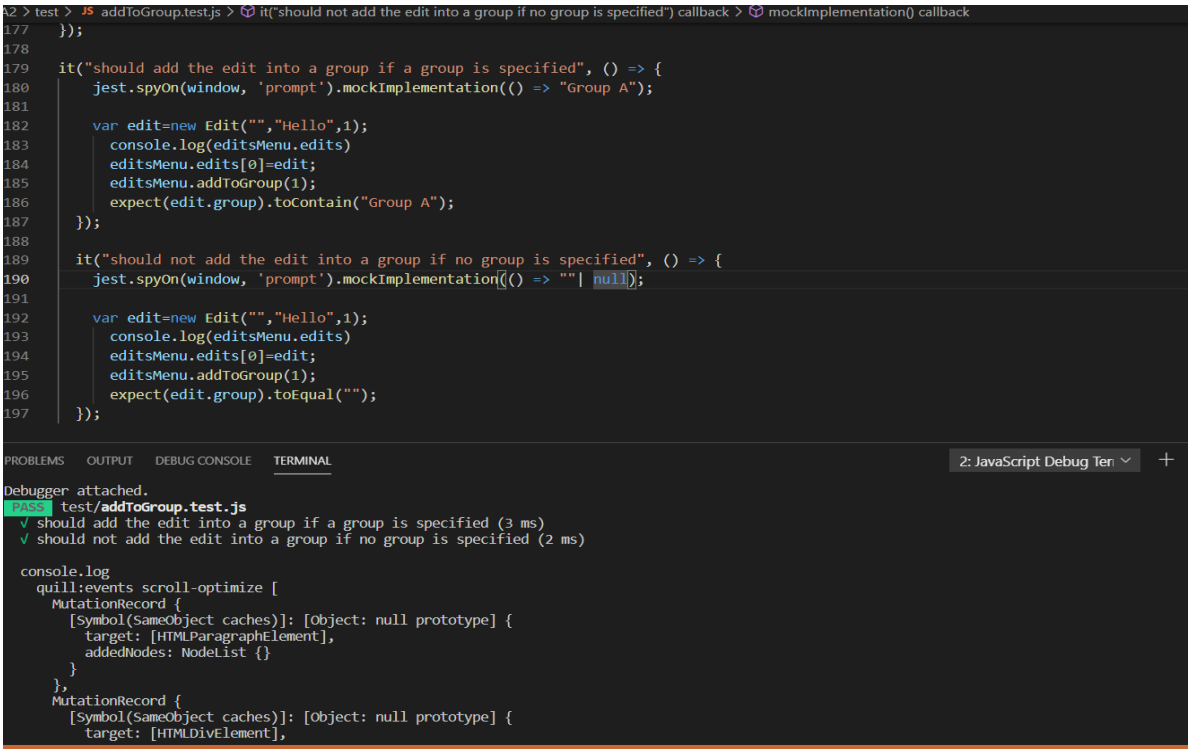
2. editSelected():

Check whether the boolean value “selected” of an edit object is set to be true by toBe().



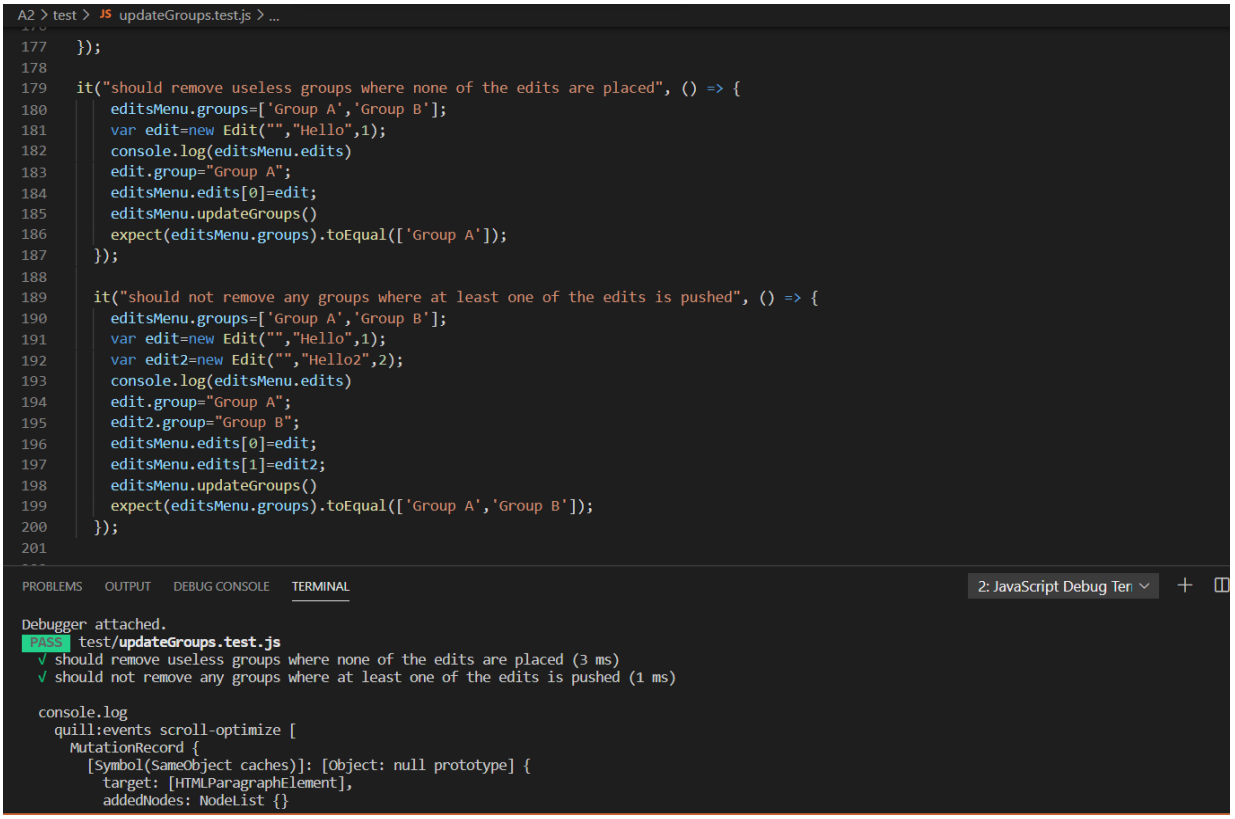
3. addToGroups():

Check whether a tested edit object can be grouped to a name which users define. The String value “group” is supposed to be equal to the name



4. updateGroups():

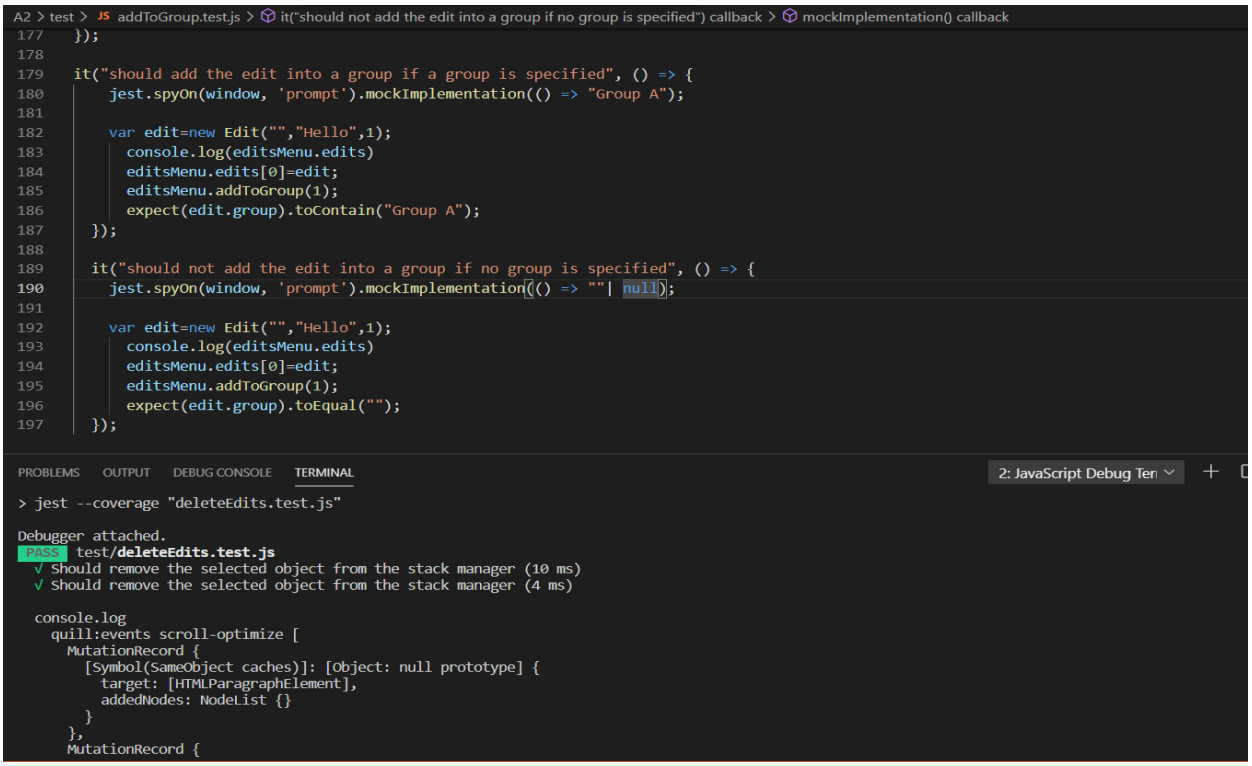
Check whether a group name will be removed if no edit object has the group name, or the group name will be retained if at least one edit object has such a group name.



```
A2 > test > JS updateGroups.test.js > ...
177   });
178
179   it("should remove useless groups where none of the edits are placed", () => {
180       editsMenu.groups=['Group A','Group B'];
181       var edit=new Edit("","Hello",1);
182       console.log(editsMenu.edits)
183       edit.group="Group A";
184       editsMenu.edits[0]=edit;
185       editsMenu.updateGroups()
186       expect(editsMenu.groups).toEqual(['Group A']);
187     });
188
189     it("should not remove any groups where at least one of the edits is pushed", () => {
190       editsMenu.groups=['Group A','Group B'];
191       var edit=new Edit("","Hello",1);
192       var edit2=new Edit("","Hello2",2);
193       console.log(editsMenu.edits)
194       edit.group="Group A";
195       edit2.group="Group B";
196       editsMenu.edits[0]=edit;
197       editsMenu.edits[1]=edit2;
198       editsMenu.updateGroups()
199       expect(editsMenu.groups).toEqual(['Group A','Group B']);
200     });
201

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    2: JavaScript Debug Ter

Debugger attached.
 PASS  test/updateGroups.test.js
  √ should remove useless groups where none of the edits are placed (3 ms)
  √ should not remove any groups where at least one of the edits is pushed (1 ms)

  console.log
    quill:events scroll-optimize [
      MutationRecord {
        [Symbol(SameObject caches)]: [Object: null prototype] {
          target: [HTMLParagraphElement],
          addedNodes: NodeList {}
```

5. deleteEdits():

Check whether a selected edit object is removed from the stack manager. This test is done after we confirm the reliability of editSelected().



```
A2 > test > JS addToGroup.test.js > it("should not add the edit into a group if no group is specified") callback > mockImplementation() callback
177   });
178
179   it("should add the edit into a group if a group is specified", () => {
180       jest.spyOn(window, 'prompt').mockImplementation(() => "Group A");
181
182       var edit=new Edit("","Hello",1);
183         console.log(editsMenu.edits)
184         editsMenu.edits[0]=edit;
185         editsMenu.addToGroup(1);
186         expect(edit.group).toContain("Group A");
187     });
188
189     it("should not add the edit into a group if no group is specified", () => {
190       jest.spyOn(window, 'prompt').mockImplementation(() => ""| null);
191
192       var edit=new Edit("","Hello",1);
193         console.log(editsMenu.edits)
194         editsMenu.edits[0]=edit;
195         editsMenu.addToGroup(1);
196         expect(edit.group).toEqual("");
197     });

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                    2: JavaScript Debug Ter

> jest --coverage "deleteEdits.test.js"

Debugger attached.
 PASS  test/deleteEdits.test.js
  √ Should remove the selected object from the stack manager (10 ms)
  √ Should remove the selected object from the stack manager (4 ms)

  console.log
    quill:events scroll-optimize [
      MutationRecord {
        [Symbol(SameObject caches)]: [Object: null prototype] {
          target: [HTMLParagraphElement],
          addedNodes: NodeList {}
        }
      },
      MutationRecord {
```

Test3:

component: undo functionality

Description: After inserting a word Hello, we undo the first edit which is the word Hello. The first edit should be stored in the undo stack. By using the toBe method, we can check whether the component of the first edit in the undo stack is the word Hello.(pass if they are the same, otherwise fail). For testing the integration, we were using the Thread-based testing to validate the output, based on the input.

Test4:

component: redo functionality

Description: Similar to the undo check, After inserting a word Hello, we undo the first edit which is the word Hello, and then redo it.The Hello edit should be shown in the editsMenu again. By creating the same edit object. we can check whether the first edit in the editsMenu is the object that we want.(pass if they are the same, otherwise fail). For testing the integration, we were using the Thread-based testing to validate the output, based on the input.

Test 5:

component: counter

Description: Tested the calculate method and update method.

1. calculate method: the calculate method can return two different results: if the unit of counter is not equal to ‘word’ then this method will count the string’s length. It is important to note that the initial value is 1 after we load the counter; therefore, if we insert ‘a’ into the editor, the return value of the calculate method will be 2. Another situation is that the unit of counter is equal to ‘word’, which is the case we used in our editor. In this case, the calculate method will count how many words in the editor. For example: insert ”two words” into the editor, the return value of calculate will be 2. We used toEqual method of JEST for both of these situations.

2. update method: this method uses to update the unit of the counter module. If there is no content in the editor, the initial value of the unit is equal to ‘word’. After we insert some edits to the editor, this method will change the unit value to ‘words’ . So, we did two tests by using the toMatch method to check if the update method works properly. In the first test, we did not insert any content to the editor, we expected the value of the unit to not contain ‘s’. For the second one, we insert a random string to the editor. If the value of the unit contains ‘s’, then the test pass.

Test 6:

component: showGroup

Description: tested compareWord() method from showGroup class. compareWord() method: uses to compare the word that the user inserts to the text editor that matches the word that is in the array list group. This test we insert “Brossard, Montreal, Toronto” into the group array list and then use toContain() method to check whether the group array contains “Toronto”.

Risk Analysis:

Undo

1. Risk description:
   The UndoByGroup is created by the Undo editor in this project. The main purpose of UndoByGroup is testing the input of the group. For example, input a group of the cities including Montreal, Toronto, Borrsard, Sherbrook. When choosing the undo manager to remove this cities group to check if all cities have been removed. The first risk is testing undo manager, only the last city is removed and the first cities are still in the editor. This is an uncertain risk that when writing the undo program, programmers don’t realize the group of cities should continuously check the previous word if it’s a city name. If the program checks the first word is a city, then remove the city name as well. The second risk is that one sentence has discontinuous cities. For example, “This

is a test for undo by group in Montreal school Brossard” Montreal and Brossard those two cities are separated by other words. The program should test word by word to remove the city name. The risk should happen that is just removing Brossard, and Montreal is still in the sentence. This risk is just checking the last city and previous word is not the city, then the program stops. In this case, the program should be written to check every word of the sentence. Then when choosing the undo manager, all cities should be removed.

2. Criticality:
   The first risk just removes the last city, and it does not check the previous word if it is a city or not. Therefore, this program has the risk of finding groups of cities, and the undo manager cannot work properly to undo this process. Programmers should plan properly before they start coding undoByGroup.
   The second risk is that when choosing undo manage, there are discontinuous city words. The cities' groups are separated by different other words. When choosing undo management, this sentence’s cities cannot be removed, there are still some cities in this sentence. Therefore, this program still has problems to find the name of the cities.

3. Mitigation Plan:
   To improve the Undo manager we need to write appropriate code to change those risks, and make sure the best algorithm for each Undo manager of improvement.
   For the first risk, we need to write an algorithm to check the last word and previous word whether it is the same group or the same attribute. In this algorithm, we need to write a for loop to check the last word, the previous word, and previous word and so on. Until we check the word is not the same group we need to delete, then choose the Undo manager delete all the same group data. All data that has been deleted will be put into stack.

The second risk should check all words whether they are in the same group that we want to delete. This algorithm is more complex than the first algorithm. First we need to write a for loop to check all words in the sentence, since during checking the word, the word is what we want to delete. We will add the other loop to put those words into the stack. When we finish all words checking, then we can choose Undo manager to delete the same group or same attribute words.

Redo risk:

1. Risk description:
   The Redo manager is based on Undo manager. First the Undo manager deletes the option, then we can choose redo manager to restore the data, which can help users to find the data back easily just in case deleting the wrong options. The risk of the redo is that even if users choose the redo manager, the delete data still cannot find back. The reason is that choosing the Undo manager, the deleting data did not store in the stack. When choosing redo manager, because this is no data, and after choosing redo manager. There is no data to come back.

2. Criticality:
   Stack is important to use in the redo manager, without creating stack we cannot use redo manager. First, we use undo manager to delete the data. We cannot see the data in the editor, but the data is stored in the stack. When we want to restore the data, we can choose redo manager, and the data will pop the data from the stack. The delete data would display back what the user deleted.

3. Mitigation Plan:
   Before we code undo manager, we should create a stack. The undo manager can push all data to the stack, which is temporary

to store the data. After user use undo manager, and choose the redo manager. The stack will pop out all data that uses undo manager to delete, and the editor will display original editing.

Editor risk:

1. Risk description:
   We use the QuillJS text editor to edit the words, which references from the website. However, there are some risks if we use QuillJS from the website. For example, there are risks that the website may not refresh and stop maintenance. Therefore, we cannot use this editor anymore.

2. Criticality:
   QuillJS is open source, which has safety problems and everyone can see our project. People can go to our editor to check what we write, and they also can change our writing. It is very easy to disclose our information. People also can copy our report to them, which can cause academic problems.

3. Mitigation Plan:
   In our project, we can interface the API to encapsulate the QuillJS. We can set this to private and give the permission to the group member to see and edit the document. Other people who want to see and edit the document must apply for permission for the group manager, after getting the permission can edit the files.

Changes to Original Design

Making a better project should test the original design again and again, and discuss it with teammates to find out the improvement of the project. In the Technical Review section of this document there were a few extra changes to the original design implemented in the code that is currently under development.

Firstly, we added a smart Undo manager to make the undo more intensive and more convenient to use when users utilize this editor. We added Undo by group manager, which can recognize the group when inputting the testing. For example, when a user inputs one sentence, the sentence includes the cities' name. When a user chooses Undo by group manager, the editor will rescind the all group of the cities. We also added undo by time manager, which can count the users inputting time. For instance, users are typing a paragraph, and users can set up the time when they want to rescind the words. We can assume 2 second of the text editor. When users want to cancel their typing, they can choose Undo by time to cancel their previous 2 second typing words.

Secondly, we added a voice functionality to improve accessibility. We think some people have difficulty reading the text, so we added a voice functionality. People can type their words to the text editor, and when they click voice bottom. The text editor will read what they typed. The user has the option to click on a button and the text inside the text editor will be read aloud to them. This is important to help people with eyesight problems and those who rely more on hearing than seeing.

Thirdly, we also added dark mode to make our editor look nicer, which idea is based on apple IOS. Now people spend more time on the computer, which has a bad effect on our eyes. Therefore, we added a bottom when users feel their eyes fatigued to change editor color.