Software Requirements Specification

Smart Undo Capability

(Added to a java text editor)

COMP 5541

Tools and Techniques for Software Engineering

Assignment 2

Due date: March 19, 2021

Team members

First name, Last name	Student ID
Eric Ting-Kuei, CHOU	40070424
Adnan, ALI	40181614
Seyed Homamedin, HOSSEINI	40137467
Mohammed Ershad, ZAFAR	40152488

Revision History

Version	Description of the change	Date completed	Reason for changes
0.1	First Draft	6th March, 2021	-
0.2	Changes to the System Architecture	9th March 2021	Review meetings
0.3	Added Design Considerations section and Limitations	18th March	Review with Lab instructor
0.4	Added limitations	19th March	Final review

Contents

1. Introduction and Purpose

The main goal of this document is to define the design for the Smart-Undo-Text-Editor application on Windows. The design follows the decisions taken in the Requirements document, and therefore elaborates on the proposed solution previously provided. The project details are outlined through an Architectural Design (**AD**), a Detailed Design (**DD**) and Dynamic Design Scenarios (**DDS**) in this document. The AD focuses on high-level project decomposition, the DD describes the overarching system design (which includes the UML divided into multiple subsections, and the DDS which displays interaction between the different subsystems to produce a system-level service. Since this document serves as a precise and stable reference throughout development, it can be used to plan, coordinate, and guide the development of the software, estimate and allocate the necessary resources, and implement it into the system.

1.1. Scope

This document contains everything to do with the developmental decisions and design of the system, all of which are derived from the requirements and require clarification. However, this document does not include any testing of the system, which indicates that these requirements are met. It is merely a blueprint for the system that should, in theory, successfully pass any tests that would be done in correspondence with the requirements.

1.2. Definitions and Acronyms

#	Acronym / Glossary	Definition / Full form
1	AD	Architectural Design
2	DD	Detailed Design
3	DDS	Dynamic Design Scenarios
4	GUI	Graphical User Interface
5	KISS	Keep It Simple Stupid!
6	MVC	Model-View-Controller

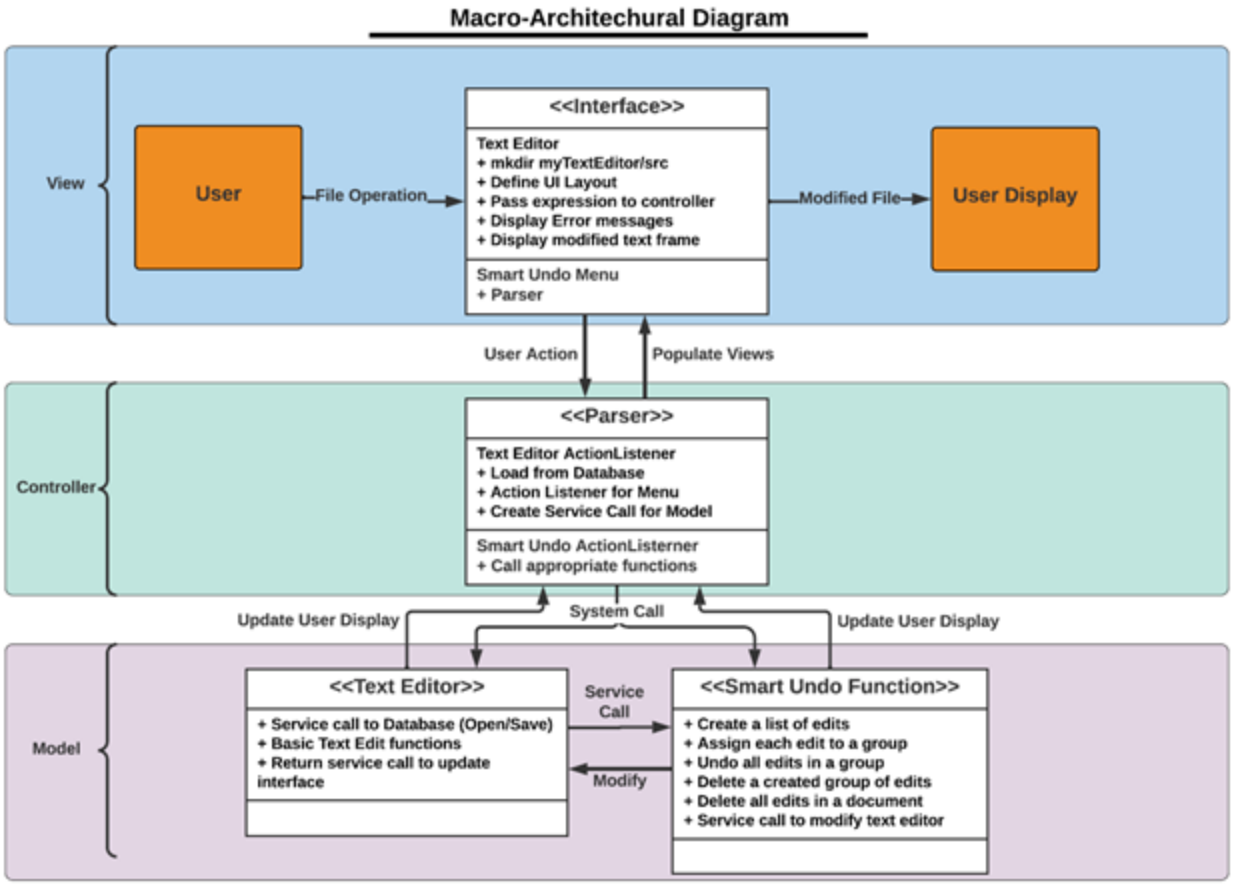
2. System Architecture

2.1. Macro-Architectural Design

The Smart-Undo-Text-Editor uses the Model-View-Controller (**MVC**) architectural pattern, which emphasizes high degree cohesion and low coupling. MVC offers easy adaptability, natural rigidity and maintenance in a module structure geared towards user interface applications, which makes it a fitting choice for this architecture.

Another advantage of MVC model was that it allows for decoupling the coding effort. The main body of our text-editor comes from an open source database at the recommendation of our stakeholders Babita Bashal and R. Jayakumar. This allows us to make minor changes into an already existing GUI, while the software development team works on a specialized Parser, saving significant time as the application would only

require a connection between the two modules.

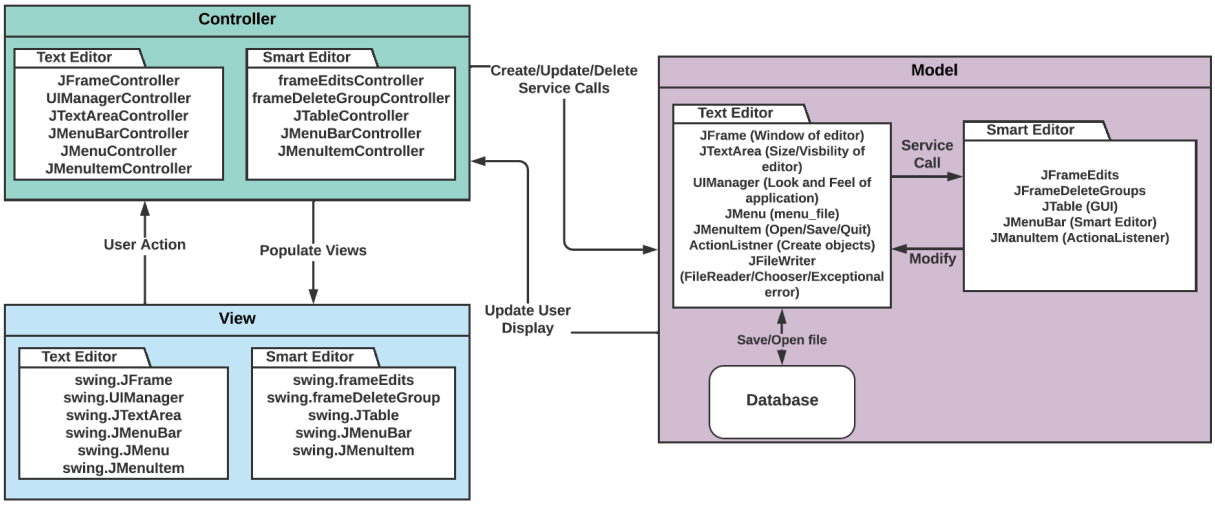


2.2. Micro-Architecture

The View, our graphical user interface is implemented by the JFrame and UIManager front-end libraries. It consists of table views, drop down menus, labels, and other simple GUI components, which are populated by the viewController and our SmartTextApplication with data from the model. The View is the only component that the user interacts with, and is responsible for reporting any user-triggered events (mouse clicks, text entries) to the viewController and renders the updated model received from the Controller. A more in-depth view of the interaction between the user and the view can be seen in section 3 of this document.

The Controller populates the view with the model and translates user input into service calls or behavior requests to manipulate the model. When the model is changed, it updates the view with new data.

Micro Architechural Diagram



The Model consists of two parts: The application logic and data. The application logic manages all basic text editor functions already encoded into the open source software. Each service is updated via the behavior request generated by the Controller, which delegates calls to different parts of the editor and finally updates the user display. The data component of our software consists of a Smart Editor and a local database. The local database can be used to open/save text files and keep a long-term backup. However, the Smart Editor is wiped off once the application is closed. The details of these services and databases are shown in section 3 of this document.

2.3. Subsystem Interface Specifications

View Subsystem

The view subsystem is implemented through the Java Swing framework, which is part of the java foundation classes. It is managed by a collection of simple libraries like swing.JFrame and swing.UIManager with the ActionListener class, that is an extension of JFrame widgets, and responsible for sending an event handler to the Controller for each click on the menu item. The File drop down menu contains pre-existing selections present in the original text editor (New, Open, Save, Exit). The Smart Undo drop down

incorporates the SmartTextApplication choices, which ranges from creating and deleting groups of edits, selecting/deleting specific inputs to Undo, or resetting the Undo features and starting from null values. Functions implemented by the event handlers can be seen in the View Subsystem diagram. Finally, this system also consists of the text area, handled by the swing.JTextArea class which updates on a user keystroke rather than a mouse click.

Controller Subsystem

The Controller can be divided into two categories. The first one, viewController, is a collection of event handler functions which respond upon actions implemented by the user with regards to the menu bar and text area. These functions receive user input and load an existing .txt file, create a new .txt file or populate the view area with domain data, if called upon, with a keystroke. The functions catching these inputs pass their calls to the services in the model-subsystem described below.

The second category involves the SmartTextApplication controllers, which deal specifically with the SmartUndo functions of our application. All these controllers initialize with a null value even while loading an existing .txt file from the local database. They receive input from the view-subsystem at each keystroke and pass these onto the model-subsystem in form of a line position in the .txt file. They are also the ones who pass any errors from the services to the views, alerting the user of any problems that might have occurred.

Model Subsystem

The Model-subsystem contains both our application logic and our database. As mentioned in the previous section, the first part of the application logic consists of basic text editor functions embedded in the original source code. These functions are updated via service calls generated by the Controller and update the user display with the modified .txt file.

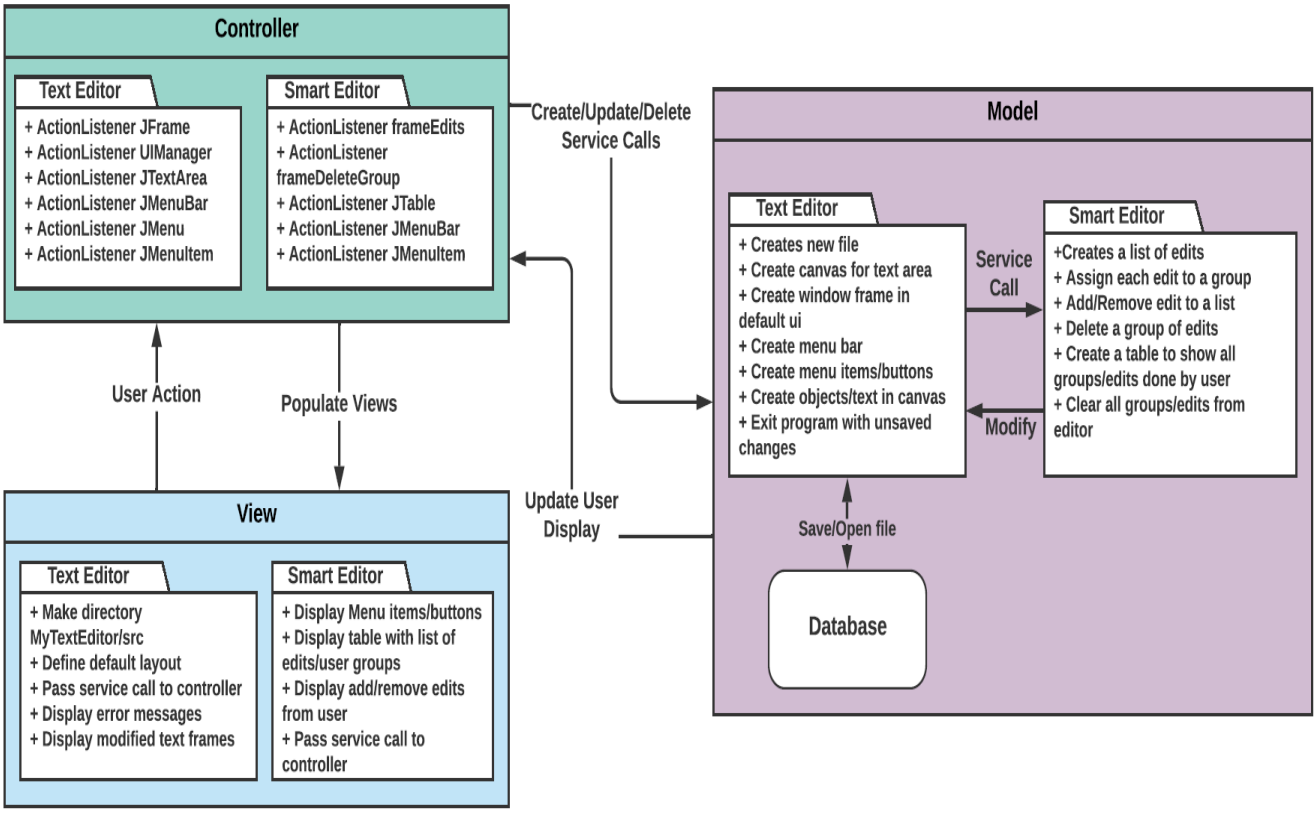
The second part of the application logic deals with our SmartUndo functions. A service call is made for each keystroke recorded via the Controller-subsystem, which is

implemented in the application via the JTextArea or JMenuItem (See Section 5 for functionality).

The Model-subsystem is organized in a layered manner, wherein each layer handles its own services and uses values determined by other layers without worrying about their implementation. As shown in the Architectural Diagram, the source-code implements the basic text-editor services and creates an object in the database for each keystroke. Data validation, processing and integrity are not validated by each layer and assumed that the relevant layer will have handled it. The SmartEditor package, initializes at null value and recalls the position of each object from the database for the relevant function. The SmartUndoController calls services from within this package to update the View-subsystem and generates a user-output.

- The Editor package, embedded into the source code, provides a connection between all underlying layers.
- The object module handles character and edits creation, validation, and verification, before storing it in the Database.
- The Smart Editor package is a lower-level layer and is called upon by the MVC controllers services, generated by the Editor package.
- The SmartUndo functions are an extension of the java.swing toolkit, which makes it the primary reason for choosing the MVC model. This model allows us to decouple the code for our GUI and application logic, allowing one coder to specialize in GUI and another on the Parser. Since each of them requires a different widget, the coder for each package only requires a basic understanding of the implementation, saving us a lot of time and testing. With an easy to organize code, future iterations for each functionality can be revised for further Q&A.

Subsystem Interface Specifications



3. Detailed Design

Design Considerations

We realized after interviewing our clients and potential users that they required an easy to learn and easy to use application that would not require high levels of computer knowledge. Therefore, the principle we followed was **KISS** (Keep It Simple Stupid!), where simplicity was the key goal of our design.

1. Since everyone was familiar with a click and play feature of a basic text editor, we decided to extend our source code into creating additional menu items. This module would naturally mimic the GUI of the original text editor and allow the user to visualize edits/results at a click of a button (See Section 5).

2. The Smart Text Editor application was designed with an existing source code, with pre-built functions and libraries embedded into it. To keep the system architecture simple and easily modifiable for future iterations, we used extensions of previous classes to allow for auto-injection. Each service package has a Module class designed to bind and provide an implementation to an interface. This way classes are never instantiated directly into each other but injected. This pattern is useful because a change in implementation is as simple as creating a new class and changing the module binding. The class that uses it and the Q&A tests that will be performed should in no way be affected. Mocking classes for test purposes is also much easier.

3. We also found that merge conflicts between software developers were much less likely to happen because we can each work on a different part of the system architect without modifying another module.

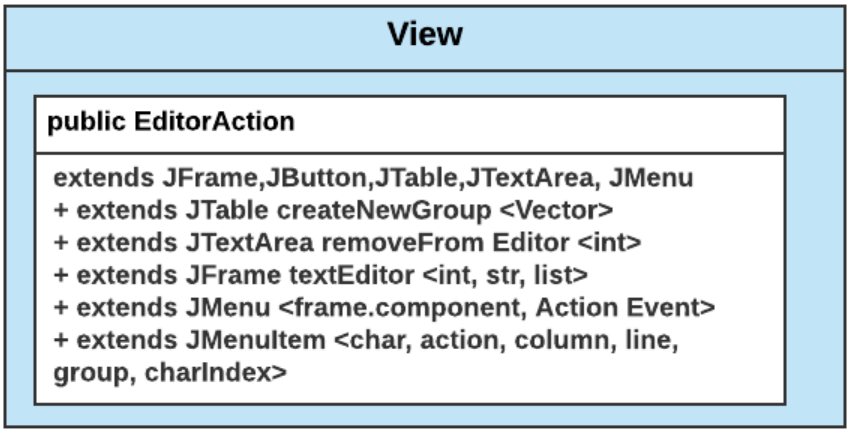
Software Requirements

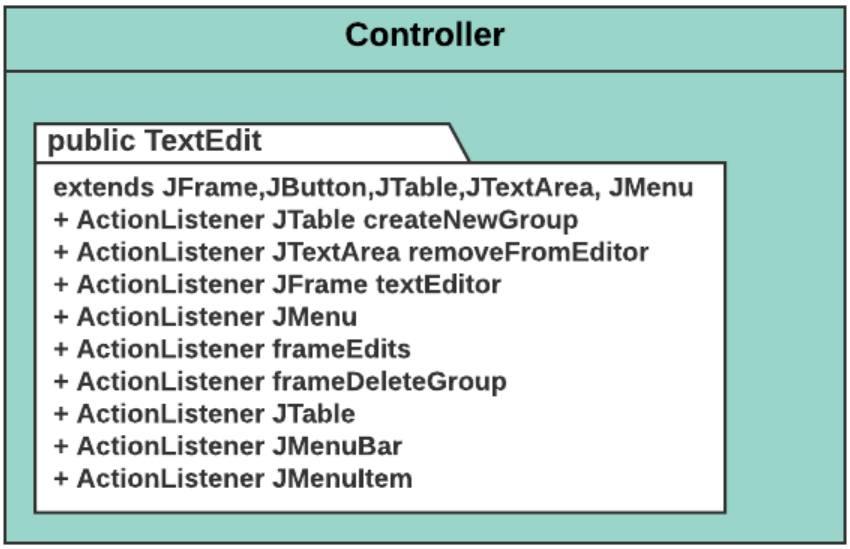
- Operating System : Windows 10 64-bit recommended
- Compiler : Java 8 or lower - Java 16 recommended for important enhancements and improved performance
- IDE : IntelliJ IDEA recommended
- Front End : Java Swing and Java AWT

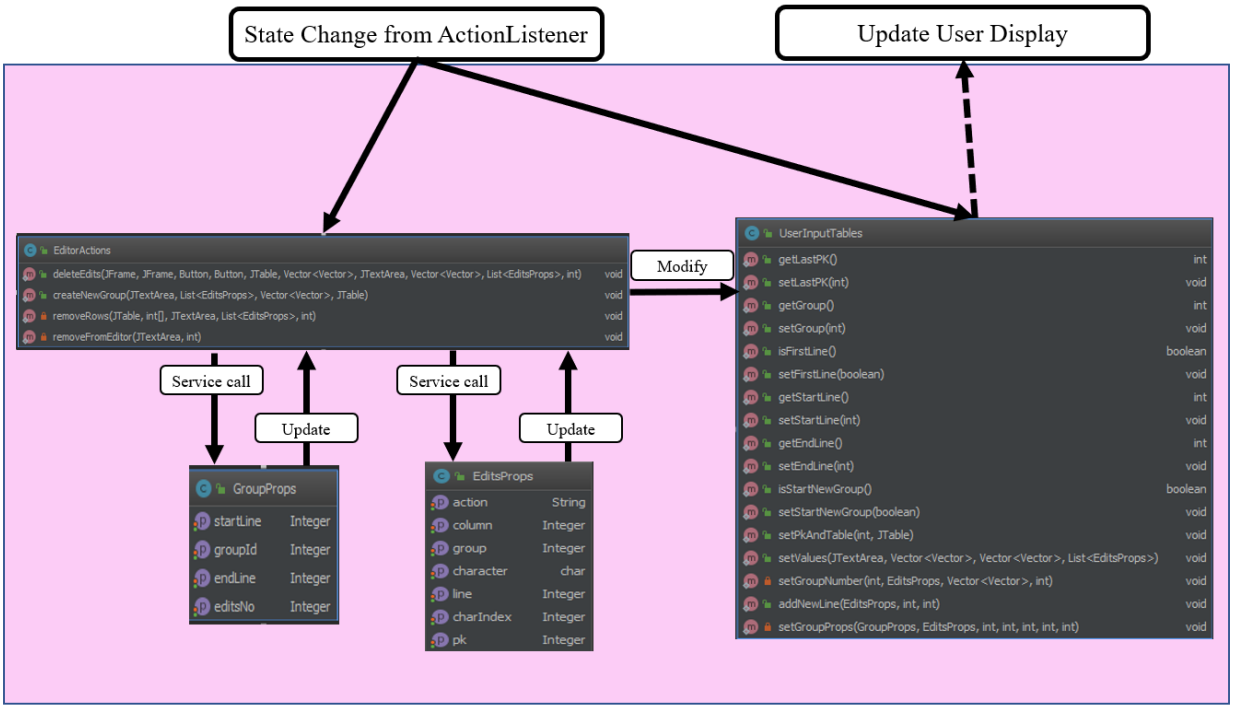
Class Responsibility Collaborator (CRC) Diagram

This section provides an overview of the class diagram of our system, useful for software developers and testers. As we used an open-source text editor and injected our Smart Editor package into the code, our CRC will cover only the relevant information.

As each class is injected into the original text editor, it extends on the relevant java.swing widget and takes an <input> as shown below.




Controller
public TextEdit
extends JFrame,JButton,JTable,JTextArea, JMenu
+ ActionListener JTable createNewGroup
+ ActionListener JTextArea removeFromEditor
+ ActionListener JFrame textEditor
+ ActionListener JMenu
+ ActionListener frameEdits
+ ActionListener frameDeleteGroup
+ ActionListener JTable
+ ActionListener JMenuBar
+ ActionListener JMenuItem


State Change from ActionListener
Update User Display
EditorActions
deleteEdits(JFrame, JFrame, Button, Button, JTable, Vector<Vector>, JTextArea, Vector<Vector>, List<EditsProps>, int) void
createNewGroup(JTextArea, List<EditsProps>, Vector<Vector>, JTable) void
removeRows(JTable, int[], JTextArea, List<EditsProps>, int) void
removeFromEditor(JTextArea, int) void
Modify
UserInputTables
getLastPK() int
setLastPK(int) void
getGroup() int
setGroup(int) void
isFirstLine() boolean
setFirstLine(boolean) void
getStartLine() int
setStartLine(int) void
getEndLine() int
setEndLine(int) void
isStartNewGroup() boolean
setStartNewGroup(boolean) void
setPkAndTable(int, JTable) void
setValues(JTextArea, Vector<Vector>, Vector<Vector>, List<EditsProps>) void
setGroupNumber(int, EditsProps, Vector<Vector>, int) void
addNewLine(EditsProps, int, int) void
setGroupProps(GroupProps, EditsProps, int, int, int, int) void
Service call
Update
Service call
Update
GroupProps
startLine Integer
groupId Integer
endLine Integer
editsNo Integer
EditsProps
action String
column Integer
group Integer
character char
line Integer
charIndex Integer
pk Integer

4. UI Design

4.1. High Level Overview

The text editor must have an interface that is user-friendly.

1. For instance, the standard undo functionality (ctrl+z) that most people are accustomed to must remain the same. This will undo any new edits in a LIFO method (Last In First Out).
2. Users should also be able to easily create a new group.
3. Users must be able to perform undos for specific edits quickly and intuitively.
4. When deleting edits, the process should be similar to the undo except this time, the selected edits get deleted.
5. When deleting one, multiple, or all edits, the text editor will ask the user for a confirmation.

4.2. Interaction Design

There are 3 types of interactions: shortcut key, button, and select list.

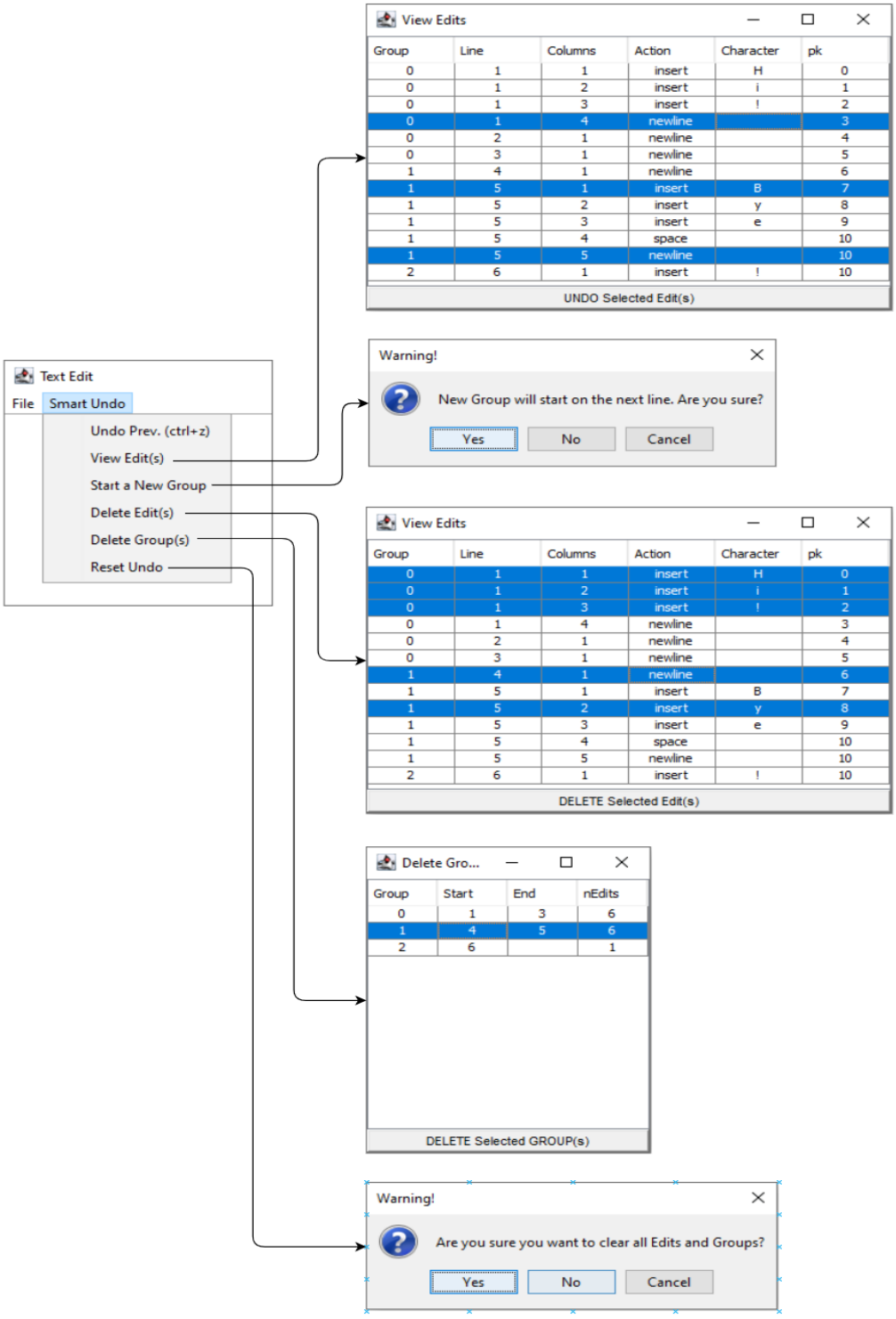
Shortcut: This is used to apply the standard undo functionality (ctrl+z), and to select multiple items in a list (shift + left-click).

Button: This allows the user to carry out an action (e.g. start a new group, undo an edit, and delete an edit).

List: This is to display the list of edits and the list of groups. Users can perform one or multiple selections within the list.

4.3. Visual Design

The graphical user interface GUI must be clear and intuitive. The smart undo i s clearly indicated on the menu bar. Each functionality is clearly labelled.



4.4. User Experience Design

The design process is an iterative one, where we first design the GUI with the ideas we have in mind, then we iterate the design around the constraints. We implement and build the GUI then we validate the design through testing.

It is important to factor personal bias out of the equation. This is because as we design and code the text editor, we are becoming familiar with the text editor therefore it is hard to put oneself as a first time user and provide a just assessment free from a priori knowledge.

As seen in the previous section “Visual Design”, we have simply added a “Smart undo” menu without overcrowding the menu bar at first glance. The user is presented with a list of options, with 2 new windows (the View Edits, and the Delete Groups). Aside from understanding how new Groups are created, the rest is very intuitive.

We minimize the learning curve that users need to use the Smart Undo functionality, while making it nearly graphically indistinguishable from the default application.

4.5. Accessibility

The font size is adapted to the system OS font size.

‘The choice of color is left at system default:

- White background
- Black text
- Grey text editor boundary
- Blue for selections.

4.6. Prototypes

No change made to original design so far.

5. Limitations

In this section, we will describe some of the technical limitations that users may encounter while using our smart undo text editor. These technical limitations can not only serve as a guidance to users (e.g. what not to do), but can also serve as recommendations for improvements on upcoming versions.

1) Users can't create a group edit within a group edit.
2) Once an edit is undo, you can't redo. Hence there is no redo function.
3) Edits can't be saved for later usage (e.g. user closed the document, all the records for edits are gone).
4) After a while of entering text, there could be too many edits that it becomes hard for users to trace the correct edit(s) to undo.
5) There is a lack of visual guidance as in which group of edits the user is currently performing edits on. Perhaps in the future, a feature that would highlight lines of the current group the mouse cursor is on would be a nice addition.
6) Due to the complexity of the feature, there is no shortcut or hot keys to perform smart undo.
7) No refresh option in View Tables. Users must close and re-open the tables to see new edits or groups.

6. Contributions

First name, Last name	Student ID	Contribution
Eric Ting-Kuei, CHOU	40070424	UI Design section, limitation, UI coding
Adnan, ALI	40181614	System Architect, Class Diagram
Seyed Homamedin, HOSSEINI	40137467	Preliminary Coding Implementation, Document review
Mohammed Ershad, ZAFAR	40152488	Document review and Formatting

7. References

1. Software Requirements Specifications
2. Lecture Notes from the following weeks
3. Stakeholder inputs
4. Brainstorming sessions and meetings within the team