Software Quality Assurance

Smart Undo Capability

(Added to a java text editor)

COMP 5541

Tools and Techniques for Software Engineering

Final Report

Due date: April 20, 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

Development Plan

This report contains the estimated cost and schedule for the project that Team #3 followed, in the form of project schedule and phase plan. During our first team meeting, we decided to document our software milestones in form of a Work Breakdown Structure (WBS). This was to allow Team #3 to create a hierarchical structure, which would break down the work to be executed and align with our Waterfall-process model. Each team member would be able to execute the project objectives and create/update the required deliverables based on the deadline or the envisioned scope of the project. This was done by introducing project milestones and phase delivery dates of protypes and builds along with their completion dates for the team to work on.

Project Timeline Estimates

Based on the original timeline given to us by our stakeholders, we based our project on 91 days (13 weeks). We defined one-day on the WBS equivalent to 8-hours, which can be completed in a day, week, or the whole project. With a smaller team than usual, each member had to pull more weight to meet all our deadlines set in by our stakeholders. Even though one-day is defined as 8-hours, they can be changed based on the complexity and manpower assigned to each task. So sometimes, one-day may be more or less than the actual amount of time estimated to complete the task.

The **Requirement phase (phase 1)** took us approximately 15 one-days to complete and was completed by February 16. This was based on the amount of time all members had to collect input from potential users and stakeholders.

The **Design phase (phase 2)** had a schedule of around 26 one-days for completion. Even though this was only the design phase, Team #3 broke it down into two parts to make further iterations down the line. The first phase, which was completing the first prototype, had an estimated completion time of 13 one-days. Once the prototype was completed, we required extensive feedback from our stakeholders to meet all requirements and start working on the second prototype. The reason we allotted an additional 13 one-days to the second protype was due to specific meet between team members and potential clients due to availability.

The **Implementation phase (phase 3)** was estimated to take 44 one-days to complete, but due to the shortness of manpower and project deadline, we decided to complete it by April 20. The implementation also included our testing and release date as adjustments needed to be made in the text editor based on the success of each build.

Project Gantt Chart

Project Milestones

The first prototype commenced once the first part of the Design phase was over. It contained all the core features introduced in the Requirement phase, without the interface to implement it into our basic text editor. The results showcased were implemented within the JDE and the GUI was pushed back into part two along with minor revisions.

Feedback received from the first build was used in the second prototype, along with an extension of our Smart Editor into the basic text editor Frame. Even though the design phase and second prototypes were being developed in parallel due to time limitation, only the elements defined by the stakeholders were implemented without optimization in each build. Once the second build was completed, we opted to independently test our software in the second part of the design phase. This was also where we decided to optimize our Smart functions by implementing different algorithms, with a backup build ready to be released, if so asked for.

Project Plan

The project plan was segregated into phase plan and project schedule. The phase plan consisted of major and minor milestones, along with a given timeline for the project. Due to a shortage of team members, we decided to dedicate a major task to one member at a time with revisions being discussed in team meetings once a week.

The milestones for the project were also divided among the three phases, and they were not marked as completed until the revisions were discussed and written off in team meetings.

A project schedule was drafted to keep track of when each milestone was completed and a reminder for each project deliverable requested by the stakeholder.

Project Resourcing

Team #3 consisted of 4 members our of an expected 6, which limited our ability to form sub-teams for each phase. Rather than forming subdivisions to explore different technologies, we decided to all work on each phase of the project rather than do future research for upcoming phases. However, since all of us lacked software engineering experience, we decided to rotate team leadership to give each member the relevant exposure.

Implementation Roadblocks

During the implementation of our planned architecture, several roadblocks appeared, which forced us to change the architect to work around those.

The first roadblock we encountered was during the design stage of the Smart Undo Function. The problem was that when the Smart Function was received a sudden burst of input from the user, the record table would stall. After an extensive use of the debugger, we were able to track the problem back to our linked list function. The software would try to update the linked list and the JFrame table at the same time. If the table received the input prior to the append() command, it would include a wrong String in the JFrame and overall, record a wrong edit. Therefore, the insertion performance had to be improved. Upon research, we discovered the usage of “Critical Sections” that was very efficient in record keeping. This would block off all output from the linked list,

till the append() or insert() function went through and the input was converted into a String. Once it was done, the JFrame or JTable would record the output string and keep it as a list of Undo, which ended up solving a problem.

Another major issue was that if we put an Integer after a String and then used undo on the Int value, it would undo anything before and after the Integer. This problem was both a mathematical and conversion method. We decided to input values into our Undo List, one digit at a time (be it an integer or String). Since our output was String only, Java tried to use toString() on its own which would couple a digit before and after the integer value. To solve this, we had to implement our own toString() method, so java would end up recording that as compared to fixing the problem by itself.

After the basic Undo functions were implemented, and work was started on the parser, we realized that if a user gave and improper input by pressing “key + Enter” at the same time, there was a potential to crash the software. For this reason, it was decided that our program had to have a way to verify the user input as well as display the appropriate error message. Hence, an expression verified had to be added to our initial design and implemented later on.

From the problem above, the coders responsible for our ExpressionValidator and the Parser spent a great deal of time manually parsing Strings. In fact, 30% of the effort of those respective classes were spent on String manipulation of all sorts but converting each input to a JTable and looking for edge cases. The team only learned of the Regex API (Pattern and Matcher classes) at the end of the project. Even though it increased our productivity a lot and probably allowed us some leeway in compensating for some

edge cases, most of the work done required individual attention and therefore had to be revised a lot.

Adjustments to the Micro Architecture

As mentioned in previous deliverables, a generic body of a text editor was presented to us by our stakeholders R.Jayakumar and Babita Bashal. However, we did not have a full picture of the microstructure of our Smart Text Editor before we started coding it. This meant that we had to come up with the algorithms and UI design that would not only be executed in a timely manner (See Project Plan) but also provide an excellent user experience.

During the design phase of the project, Team #3 decided to use Gap Buffer Data Structure to edit and store text in our Smart Function prior to UI implementation. However, due to our inadequate knowledge of JavaFx and translating inner conversions to GUI.swing, we decided to switch to an array and focus only on one input a time rather than multiple cursors. This not only allowed us to fulfill the project requirement, but also showcase user inputs in an organized manner for efficient undo function (if needed). However, one crucial element we had not foreseen was that the String expression entered into our Smart Text Editor by the user, would not be “clean”. That meant that we would need to validate it for syntactical correctness and do some light formatting for it to be simplified for the Parser. Even though our validation step was outlined in the planner as 2-3 days, our Smoke-Testing process of the functions took us 2-3 weeks. We initially assigned only one coder to the validator function and within a day of testing, he came up with 10 scenarios that we had not anticipated. We figured that in a normal user

experience, this number would be much higher as each user has a different set/thought process for keyboard inputs. Team #3 ended up spending the majority of our programming portion of the project on methods pertaining to recording user input to memory. This was a significant time sink and we kept coming up with new ways to break the parser. The final consensus of Team #3 was to switch to linked lists for our final user input function for the Parser, even though it required more extensive coding than anticipated. This was, by far, the largest change added to our micro architecture.

Another micro architectural decision, Team #3 had to vote on, to consolidate all Undo Functions into a single class. The initial idea was the implementation of multiple classes, where one would record the input, one class would only extend into the JFrame to update the interface and one would only be relevant for the Smart Undo feature. However, upon implementation, we realized one class into the JavaFx, which ideally would deal with the user input and extend its output into the JFrame. But as mentioned earlier, our toString() conversion and recall feature from the linked list had to be synchronized using critical sections to correctly update the JTable extension into the application. Each class functioned well during module testing but required significant amount of coding to implement it in the whole design. Therefore, it was unanimously concluded that there was a good reason to put them all in one place and require as little syncing as possible.

We also reduced the quantity of Exception classes we used. Initially, we initialized our Smart Function with multiple classes that would check multiple exceptions based on input (it would throw a MathExceptions error if someone would miswrite a function that existed in “The Java Math Package” or give a helpful error if someone divided an

equation by 0). However, we reduced all this to basic SyntaxExceptions due to limited time and a much more manageable code. It also greatly reduced our CRC diagram, which would be visited multiple times prior to implementation.

Outstanding Issues

Even with such a methodological and diligent approach to software development, Team #3 faced some challenges that we were unable to solve. Even though some of them were due to having been too ambitious and not having the programming knowledge to implement them, most of them were due to time limitations. As mentioned before, our biggest limitation while creating the Smart Undo extension was dealing with the user input. Even after changing the Data structure and design of the software, there were instances of Exceptions which had to be worked around in order to meet the project deliverables. Some of the limitations for the changes in the Microarchitecture are listed below:

Limitation	**User Experience**
Undoing last group edit	User has to manually open the Undo-Table and remove the last element of the table to exclude it from the group, even though there is no character in the text area.
Limited edits per page	The highest number of edits that can be incorporated in a group is 9-digits.
Backspace edit limitation	Backspace is not counted as an edit for the Smart Undo Function and therefore cannot be removed from the Undo-Table.

Limitation	User Experience
Key release per edit	The user has to release the key prior to inserting a new character or only the first char will be counted as an edit. It is therefore suggested that the user enter the keystrokes slowly and carefully.
Can’t insert edit within an edit.	Instead of inserting an edit within an edit, the function will just enter a new line without creating a new edit.

Conclusion

It was agreed within Team #3 if the project were to continue and that the time permits, we would like to provide these functions into our Smart Text Editor. We would also like to improve the overall UI experience for the user by implementing key mapping, customizable skins, making it available online and publishing it on various platforms like smartphones. This was a great learning experience for Team #3 and a great example of team dynamics overcoming greater obstacles.