Software Development Planning & Execution

Scribr Application

Prepared by:

Dunya Oguz (40181540)

Muhammad Azman Akhter (40154969)

Hugo Joncour (40139230)

Sarabraj Singh (29473858)

John Purcell (27217439)

Tom Lamouret (26885446)

Table of Contents

Introduction

The Scribr is intended for use during the publication process of a textbook. Therefore, its intended users are peer-reviewers and editors hired to edit a text document and make appropriate changes before final publication.

The goal of the product is to help users save resources during the editing process. The main objective used to measure this is time saved using Scribr versus other text editors like NotePad++ and Atom. With Scribr, users are able to keep track of the last five edits made within a given document. The user can subsequently undo individual edits, or a custom assortment. Additionally, Scribr allows the user to select and delete a subset or all of those recent edits so that they can no longer be undone.

The initial rollout of the product will be directed towards individual users (mainy, contracted editors and reviewers). Only the basic undo functionality will be implemented in the first version. Subsequent releases will address any future bugs and include additional functionality as need arises. The following document outlines the risk analysis and the associated development plan for the Scribr software.

Development Schedule

After an in-depth consultation with the development team, Scribr's development schedule was created. One of the team member, Muhammad Azman Akhtrer, was assigned the role of a project manager. Using a rudimentary project scheduling software called Trello, Azman was able to assign work, link dependent tasks and allocate resources in real-time. Additionally, the whole development is required to meet weekly or bi-weekly to provide an update and demo new features implemented since the last meeting. The meeting minutes were recorded and are provided for documentation purposes.

Meeting Minutes

Meeting for Task Scheduling

Project: Initial Development Plan		
Date: March 25th	**Time:** 5PM EST	**Location:** Virtual - Zoom Meeting
Attendees: Dunya, Azman, Hugo, Sarabraj, John, Tom		
Meeting Objectives		
The purpose of this meeting is to coordinate the coding efforts for the implementation of the Scribr Application. By the end of the meeting, the development team must identify all		

the development tools required, split the tasks and plan a schedule for accomplishing each task.

Discussion Points

- After some discussion, the team agreed to use a version control tool Git and GitHub to coordinate any development efforts. Since some development team members were not familiar with this tool, a brief tutorial and other resources were provided by senior developers Dunya and Sarabraj.
- Since Scribr required a GUI for user-input and display application-related content, Java Swing was selected as a framework for implementing the UX design.
- Additionally, Apache Netbeans IDE was selected thanks to its Java Swing support, making it a powerful tool to develop the GUI.
- Component tests should be implemented concurrently so that developers can verify the proper functionality of each component.
- The team agreed that an initial prototype of the software should be up and running by April 15th.

Task Distribution

Team Member	Task	Estimated Time needed
Dunya	Database Implementation using SQLite	1 week
Dunya & Sarabraj	Scribr Logic Implementation, primarily Undo functionality	2 weeks
Azman & John	Implementation of UX design using Swing Framework	2 Weeks
Tom & Hugo	Component Testing and Quality Assurance including implementation of JUnit tests	2 Weeks

Project Status Update Meeting

Project: Initial Development Plan		
Date: April 9th	**Time:** 11 AM EST	**Location:** Virtual - Zoom Meeting
Attendees: Dunya, Azman, Hugo, Sarabraj, John, Tom		

Meeting Objectives

The purpose of this meeting is to evaluate the progress made by each team member since the previous meeting. More specifically, each member is required to give a ten minute stand-up addressing each of the following points:

- What did the team member work on in the last two weeks?
- What are they working on right now?
- What issues are blocking them from delivering the assigned task on time?

Additionally, the team members are required to demo any new functionality implemented since the last meeting.

Discussion Points

- Dunya presented her implementation of Scribe-related constructs in the SQLite database. However, she was having difficulties in generating appropriate queries using JDBC and linking the Java Classes with their associated tables in the SQLite database.
- Azman and John provided a brief demo for the Scribr GUI which was implemented using the Swing framework. However, both developers failed to implement any functionality associated with the MenuBar or the SideMenu.
- Sarabraj did an informal code-review to present how he implemented Undo functionality for Word construct. However, he struggled to effectively handle Undo functionality for all other Scribr related constructs such as sentences, sections and chapters.
- Hugo and Tom provided an update upon their efforts to implement the jUnit tests for validating each component. Although they made some progress, a lot of work remained to be finished.

Task Distribution

Team Member	**Unfinished Tasks**	**Extension Requested**	**New deadline**
Dunya	Complete SQLite database implementation & provide functional database connectivity	3 Days	April 12th
Sarabraj	Implement Undo Functionality for all Scribe-related constructs including Chapter, Paragraph & Sections	1.5 Week	April 16th
Azman & John	Add proper functionality to each GUI related components so that users can interact with the Scribr application	1.5 Week	April 16th
Hugo & Tom	Finish implementation of all JUnit tests for all components of the Scribr Application as per Software Quality control document	1.5 Week	April 16th

Final Meeting & Software Demo

Project: Initial Development Plan		
Date: April 16th	**Time:** 3 PM EST	**Location:** Virtual - Zoom Meeting
Attendees: Dunya, Azman, Hugo, Sarabraj, John, Tom		
Meeting Objectives		
The objective of this meeting is to demo the finished product. This will enable the developers to validate the software against the requirement documentation and identify any missing features or inappropriate functionality within Scribr application. At the end of this meeting, all team members will vote whether the software is ready for shipment or the project manager should get an extension to address any major issues. Since any delays in software shipment will be extremely costly, any such decisions will be made after great consideration.		
Discussion Points		
● Dunya was successful in generating appropriate queries using JDBC, thus provided functional database connectivity for the Scribr Application ● Azman and John implement all functionality associated with the MenuBar including File, Help and Insert Submenu. However, some aspects of the SideMenu have a few minor bugs. ● Sarabraj succeeded in implementing Undo functionality for Scribe-related constructs. As a result, users can track their last five edits made within a given document and subsequently undo individual edits, or a custom assortment. Additionally, users can select and delete a subset or all of those recent edits so that they can no longer be undone. ● Hugo and Tom were successful in implementing 90% of the jUnit tests proposed in the Software Quality document. All components of the Scribr application were able to successfully execute their associated tests.		
Final Decision		
Although the final Scribr application had a few minor bugs, a group decision was made to ship the software on time without any modifications. This is because Scribr application was mostly functional and any shipment delays would have very dire consequences on customer relations. Additionally, future releases will aim to patch these minor bugs and implement additional functionality if required.		

Scheduling Diagrams

Figure 1: Gantt chart provides an illustration of the scheduled tasks associated with Scribr Application development over time. Such diagrams can show the start and end dates of a project in one simple chart.

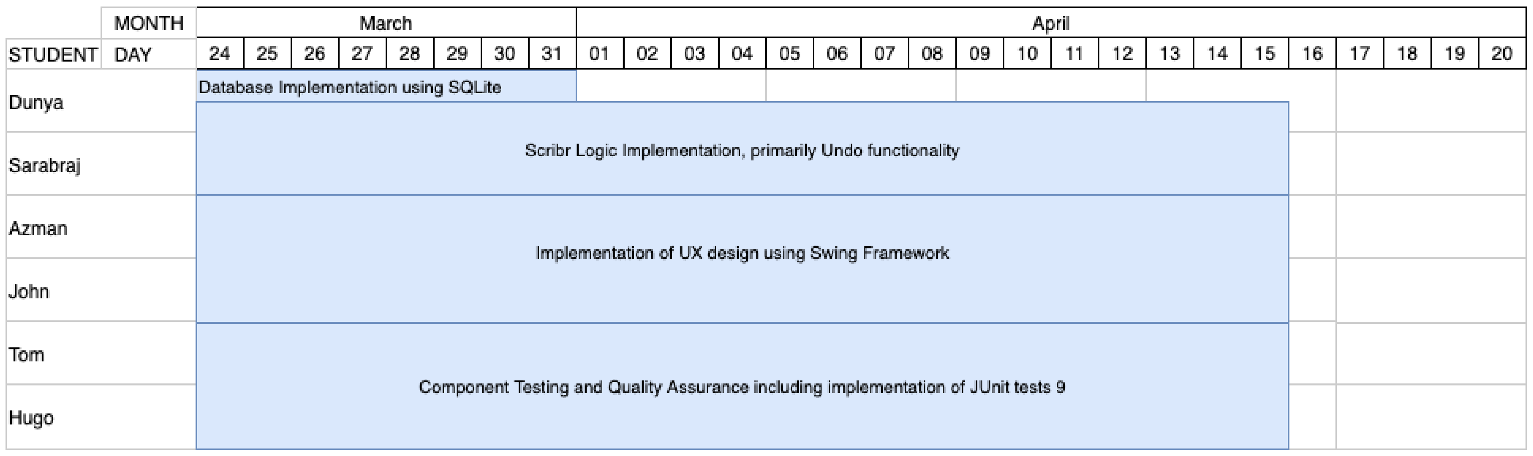
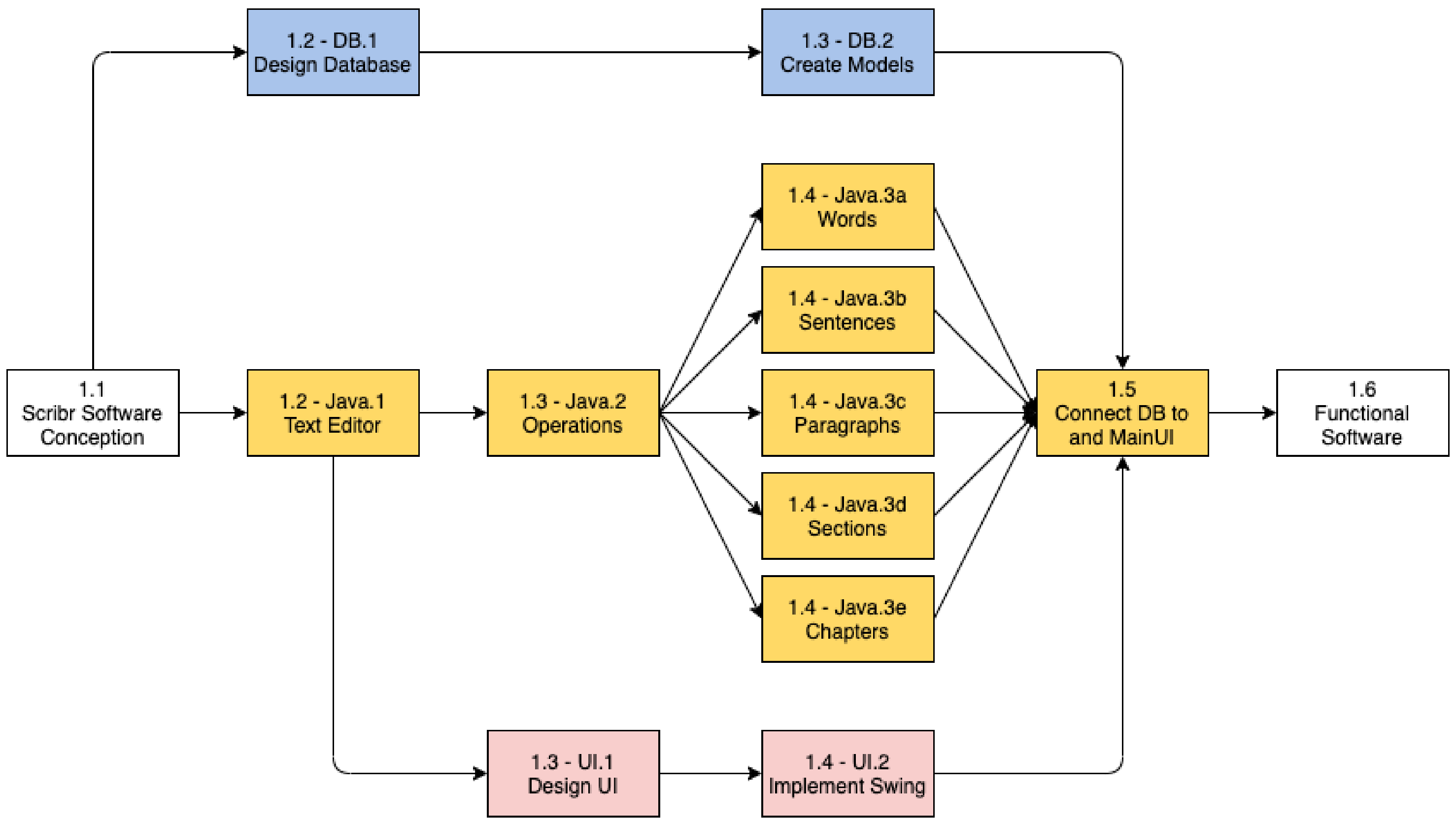


Figure 2: Task Network illustrates the dependency between actions required for the proper implementation of Scribr Application. Since each component has high cohesion and low coupling, they can be developed concurrently



Risk Analysis

Risk Identification & Risk Management Strategy

The acronyms of risk categories include, *BU: Business Risk, ST: Staffing and Personnel, PS: Product size risk.* The Impact values assigned to each risk are categorized as follows: *1- Catastrophic, 2 - Critical, 3 - Marginal, 4 - Negligible.*
Finally, the bold line in the table represents the *cutoff line*. The cut off line is a measurement for those risks that require a concrete risk mitigation, monitoring and management plan (RMMM). The measurement is based on both impact, and probability. Only those risks above the cutoff line will be issued an RMMM plan.

Risk	Category	Probability	Impact	RMMM
Inaccurate estimation of schedule: When the time required to develop and implement a feature exceeds the expectations. It may force the team to change the schedule of the software, or to adapt the software to the new schedule.	*PS*	*80%*	*1*	**Mitigation/Monitoring:** ● Create a detailed timeline of component ship dates. ● Monitor whether a ship date was made on schedule by maintaining a log of those commits made on schedule and those not. **Management/Contingency Plan:** ● Should two ship dates be logged as late or delayed, have a meeting to assess the feasibility of the entire product, and assess if non-necessary features should be removed to maintain schedule.
Knowledge gaps among developers: This software development process is utilizing many different software/workflow products (Netbeans IDE, Java/Swing, Git, Github). Each of these products has many inherent complexities, and a fairly long onboarding process. If any of the developers are unfamiliar with any of these	*ST*	*70%*	*1*	**Mitigation/Monitoring:** ● Have each team member give themselves a ranking of beginner, intermediate, or expert, for each tool being used. Should someone fall in beginner, have them commit to a focused onboarding, and have the most experienced help train those beginners. ● Monitor if someone is

tools, this would greatly reduce the ability of this member to contribute to that segment of the project.				miscategorized as intermediate when they are really a beginner by keeping track of someones errors made due to lack of understanding. If two errors are made, encourage them to seek further onboarding. ● Attempt to divide tasks so as to best highlight the skills of each developer. **Management/Contingency Plan:** ● If the onboarding does not equate to adequate understanding of the tools, have a meeting to discuss reallocation of tasks so that each developer can be best utilized based on their skillset.
Local errors: Errors that occur because of local conflicts on the machine: they may be linked to the software installation, paths, dependencies. They may not appear on the developer's computer, or during local tests, but may show up on the client's computer.	*ST*	*40%*	*2*	**Mitigation/Monitoring:** ● At each iteration of the software, attempt to run the application on a variety of machines, and Operating Systems. ● Monitor this risk by creating a log of every local error. At each iteration, these errors should be deemed closed, and their reason for occurrence should be recorded. **Management/Contingency Plan:** ● Should a local error occur on a client's machine, a task-force will be delegated to fix the error for the user, and investigate the cause. This task-force will be decided beforehand, and when such a delegation occurs, it should be their

				primary priority.
Coordination difficulties due to remote team: The development team for Scribr is not only working remotely, but is also disbursed globally. This is an added level of friction when it comes to organizing meetings with some, or all of the team due to time differences.	*ST*	*30%*	*2*	**Mitigation/Monitoring:** ● Attempt to minimize the necessity of many live in-person meetings. ● Form project sub-groups that share similar time zones if possible. ● Attempt to design an asynchronous workflow for the team ● To monitor the overall coordination of the group, a summary message will be posted each week in the group message board covering status reports, and conflicts. If any member has an issue, they can respond to the message and start a new thread. **Management/Contingency Plan:** ● Should the coordination of the team due to time changes become so burdensome that it affects the chances of shipping the product on time, more live meetings will be scheduled, and each member will rotate having meetings at an inconvenient time.
Merging conflicts: A merging conflict occurs when two or more developers push conflicting modifications on the same file. If not managed properly, a branch will overwrite the other and it will result in errors.	*ST*	*80%*	*3*	*N/A*
Need to revise the design: When because of a schedule	*PS*	*60%*	*3*	*N/A*

modification, a technical dead end, or a change in the requirements from the stakeholders, the team has to adapt the software design.				
High end user churn rate: If users of the Scribr software encounter multiple issues, or the onboarding process of users has too much friction, the churn rate of the software may increase. Churn rate being the rate of which users of Scribr stop using the product altogether.	*BU*	*20%*	*3*	***N/A***
No initial buy in from End Users: If the marketing efforts and business development efforts of the Scribr application are not sufficient, the development team will not be able to get feedback on the product to produce iterations and fix bugs so as to improve the software. It will also mean that the product launch itself has failed.	*BU*	*10%*	*3*	***N/A***

Implementation

Risk Materialization & Mitigation

Catastrophic Risks

Inaccurate estimation of schedule: The software was initially expected to be operational two weeks before the release, but due to a myriad of factors, including, underestimation of the time required to implement all the essential system features, and knowledge gaps among team members, the software development was behind schedule. This risk is catastrophic. If the intended launch date for Scribr is delayed, the company will lose its funding, and the product will never reach the market.

Before materialization, this risk was mitigated by detailing a timeline of component ship dates, and maintaining a log that monitors whether code commits were made on schedule. Should two product shipment dates be logged as late, the team would meet and assess what features could be removed to ship the final product on schedule.

The process of managing the risk began when two shipment dates were missed, and it became apparent that the risk of an underestimate of schedule had materialized. The team held a meeting, and discussed what features/details should be left out.

Knowledge gaps among developers: The software development process utilized the Netbeans IDE, Java/Swing, Git, and GitHub tools. Each of these products has many inherent complexities, and a fairly long onboarding process. After the start of the project, it became apparent that almost all team members were unfamiliar with at least one of the tools being used, and some team members were beginners at 2-3 of the tools being used.

This risk is catastrophic. With only limited resources (time, and team members), members not being able to contribute at a sufficient rate will ultimately slow the project to the point of missing the deadline.

Before materialization, the risk was mitigated by having each team member give themselves a ranking of beginner, intermediate, or expert, for each tool being used. An attempt would then be made to get all beginners up to intermediate through onboarding and coaching. As well, those most skilled at a particular tool would be delegated most of the workload associated with that tool.

This risk materialized due to the fact that there were errors in the assumptions being made on the time it takes to be onboarded on one of these development tools. Consequently, there were less members than desired who were adept at the tools, and due to the demands of the project, they would be unable to delegate time to coaching and onboarding other members. This was ultimately managed during a meeting wherein responsibilities and tasks were reallocated based on current skill set, not on expected skill set after onboarding.

Critical Risks

Local Errors: Errors that occur because of local conflicts on the machine: they may be linked to the software installation, paths, dependencies. They may not appear on the developer's computer, or during local tests, but may show up on the client's computer.

This risk is critical. Should an end user experience any such issue, they may be inclined to churn. If this happens enough time, the churn rate of the software will grow, and the company will inevitably fail.

This risk was mitigated by attempting to run the application on a variety of machines, and Operating Systems at each iteration of the prototype. The risk would be monitored by creating a log of every local error. At each iteration, these errors should be deemed closed, and their reason for occurrence would be recorded.

The risk partially materialized due to a path mistake in the *nbactions.xml* file. It would work properly on one member's computer, but would crash the compilation of the program on anyone else's computer. It required every team member to locally rewrite a section of the program in order to run it. It was fixed by not synchronizing the file between computers, and writing a function that automatically wrote that section of the file, locally.

Coordination difficulties due to the remoteness of the team: Due to the remote, globally dispersed nature of the team, there is an added level of friction when it comes to organizing live meetings.

This risk is critical. Should communication begin to break down due to lack of successful asynchronous workflow, then there will inevitably be resulting errors and mistakes. Added errors and mistakes could contribute to the biggest risk of the product not shipping on time.

To mitigate this risk, the two main strategies were to structure the workflow to facilitate asynchronous contributions from team members, and sub group the team into smaller cohorts.

This risk partially materialized. There were two occasions specifically where the overall state of the project seemed uncertain, and a series of group meetings were required. So that the meetings could take place, some group members had to stay up substantially later than others on these occasions. Through these meetings, the issues were able to be resolved, and the asynchronous workflow was able to resume.

Component Implementation & Testing

Component Integration

Driver Class Unit Tests (Component 1)

#	Method	Input	Description	Expected
1	testMain()	null	Test that we can instantiate an instance of the application	An instance of the application
2	testInsertUpdate()	DocumentEvent	Test that we can capture an event fired off in the application, and that the program will response to the event	The program will trigger downstream events based on the stimulus
3	testDatabaseConnection()	ConnectionURL (String)	Test that we can instantiate an instance of a SQLITE3 database	A connection object that dereferences the location of the database

Database Class Unit Tests (Component 2)

#	Method	Input	Description
1	testGetConnection()	null	Test that we can instantiate an instance of the application
2	testTableCreation()	null	Test that we can create a table in the database
3	testInsertRow()	null	Test that we can successfully input a row into a table
4	testRetrieveRow()	null	Test that we can successfully get values from a database table

Word Class Unit Tests (All Custom Model Objects)

** All Models in the ca.myconcordia.comp5541.scribr.models package will follow the convention below for unit testing

#	Method	Input	Description
1	testGetIndex()	null	Test that we can get the index integer of a Word object
2	testSetSentenceId()	null	Test that we can get the sentenceId of a Word object after it is set and not -1
3	testGetSentenceId()	null	Test that we can get a sentenceId from a Word object after it has been set
4	testGetWord()	null	Test that we can get the String word that we instantiate the object with
5	testHashCode()	null	Test that two objects with the same index and same String variable produce the same hashCode
6	testEquals()	null	Test that two objects are equal by equating the String values they hold in each Word object

Custom Stack Unit Tests (Component 4)

#	Method	Input	Description
1	testStack()	Object	Test that we can instantiate an instance of our CustomStack object with a given sizeand given object type
2	testPush()	Object	Test that we can push a given Object type to the stack
3	testPop()	null	Test that we can instantiate an instance of a SQLITE3 database
4	testGetSize()	null	Test that when we instantiate an instance of a CustomStack, that the size parameter is correct

Requirement and Design Changes

Changes in Requirements

- The ‘Undo Subsection’ function was judged unnecessary, and redundant, so it was removed from the final software.
- The ‘Quick Undo’ function was judged unnecessary, and redundant with regard to the other undo functions of Scribr.
- The requirement to display a widget when a user double clicks on an edit from the sidebar was removed from scope as the feature would have been too complicated to implement, and since it is not a core functionality of the program.

Changes in Design

- The undo button is not located in the menu bar, but on a lateral menu instead.
- Because the Smart Undo is the main purpose of this software, it was decided to change the interface, from a widget to a permanent lateral menu instead.
- The ‘Edit History’ option was removed from the vertical toolbar, to be integrated in the permanent lateral menu, allowing to visualize the current edits.
- The ‘Custom Undo’ function was judged not relevant anymore, as the lateral menu allowed narrowing down by type of edit, with regard to history, what part the author would like to undo.
- Instead of having the user divide the text into sections through the usage of markup language tags, it was decided that having an automatic detection of words, sentences and paragraphs would be a better user experience and that this could be easily implementable through intelligent parsing of dots, spaces and tabs.
- The identification of sections and chapters was changed to be done via highlighting the text that belongs to the respective construct via the top menu.
- Due to the complexity in enabling child-parent relationships between constructs when using a database as the application backend, it was decided to handle certain operations through the usage of stacks, implemented as doubly linked lists.

Changes in Test Cases

- Integration test cases concerning the subsection object were removed, as the construct was deemed unnecessary.
- CustomUI unit tests were removed since the CustomUI class was removed.

Appendix A: Contributions

Sections	Assignment
1. Introduction & Development Schedule	Azman & Hugo
2. Risk Analysis	John
3. Implementation	Raj & Tom
4. Design & Requirement Changes	Dunya