CS 30700: Sprint Planning Document

Team 37
Aneesh Pendyala, Sergio Alvarez, Zhenyao (Michael) Yang, Kye Jocham, Brennan Frank

Sprint Overview:

For sprint one, more than anything else, we need to achieve our product's core functionality. We need a GUI so that we can achieve the very core functionality of one replacement per template field. The Minimum Viable Prototype is what we're getting here

Scrum Master: Brennan Frank

Meeting Plans: Tuesday/Thursday at 8:15am and Saturday at 12pm

Risk and Challenges:

- Many of the risks and challenges of this sprint are associated with the difficulty of parsing DOCX files and the underlying XML files that make up those DOCX files. Since file structures can consist of many folders, our implementation may need to be recursive to make sure we cover all files. Recursive functions are difficult to implement, but we have full faith that our team will figure out how to do this.

- Another potentially difficult portion of this sprint is making sure our solution runs below the targeted runtime of 500ms. We may run into issues with our recursive algorithm being slower than needed, but we will resolve these as they come.

- The final potentially difficult portion of this sprint discussed here is setting-up the GUI since none of us have familiarity with Qt, our C++ graphical library of choice. Again, by looking through documentation, we believe we'll be fine on this though. Since Qt is very popular for C++ developers, we have no doubt there'll be lots of documentation and resources to guide us.

Sprint Detail

User Story #1

As a professional, I would like to save and reuse templates.

#	Description	Estimated Time	Owner
1	Recently used templates pop up in the GUI	8 hours	Aneesh
2	Open templates in last saved state	3 hours	Aneesh
3	Write unit test(s) for this.	5 hours	Sergio

Acceptance Criteria:

- Given the UI is set up correctly, when a user opens the application, then it should display the saved templates in order of most recently used.
- Given the save is set up correctly, when a user clicks the save button, then it should save the template correctly (with your fields half-replaced, for example).
- When reopening the template, it should load your half-edited field data if you pressed save correctly.
- Given the unit test frameworks are functioning correctly, when the unit tests are run, then they test this appropriately, and the tests pass.

User Story #2

As a client, I would like to export the documents created in a DOCX format.

#	Description	Estimated Time	Owner
1	Do the file operations within C++ to save the processed file (received from parser) to the correct folder.	7 hours	Kye
2	Ask the user if they want to overwrite the template if the file doesn't exist, and intuitively create another file if the user chooses to not overwrite.	7 hours	Kye
3	Write the unit test(s) to check the file got saved to the correct location with the correct properties.	4 hours	Sergio

Acceptance Criteria:

- Given the client's computer's file processing is set up correctly, when they export a document,then only one file is created (for this sprint).
- Given the client's computer's file processing is set up correctly, when they export a document, then the document shows up as-expected in the default exports folder (in later sprints, will be able to change this folder in the settings).
- Given the client's computer's file processing is set up correctly, when they export a document, then the document is uncorrupted and has the right file protections (read, write, execute are set appropriately).
- Given the GUI is set up correctly, when a user is doing an operation that would replace a file, then the user should be asked graphically if they are willing to replace the file, and if not, it should write the file with the common (1), (2), etc. download notation.
- Given the unit test frameworks are functioning correctly, when the unit tests are run, then they test this appropriately, and the tests pass.

User Story #3

As a client, I'd like to open a template and fill out the template placeholders to get an output.

#	Description	Estimated Time	Owner
1	Create text boxes for each placeholder	5 hours	Aneesh
2	Replace each placeholder in the document when finished (call on the processing module)	5 hours	Aneesh
3	Create GUI unit tests	5 hours	Sergio

Acceptance Criteria:

- Given the placeholder text fields work, when I open a template there should be text boxes where I can input information for the document.
- Given the GUI is set up correctly, when viewing this field replacement screen, then there should be one textbox per placeholder.
- Given the GUI is set up correctly, when viewing this field replacement screen, then the text boxes should be in order of the placeholder's order in the document.
- Given the basics of the parsing module is set up, when the program calls the
- Given the unit test frameworks are functioning correctly, when the unit tests are run, then they test this appropriately, and the tests pass.

User Story #4

As a user, I would like the application to have a search feature that I can easily find templates and documents within my library

#	Description	Estimated Time	Owner
1	Create a search bar icon in the library interface.	3 hours	Aneesh
2	Connect a search query with a result	15 hours	Kye
3	Write unit test(s) to test the search functionality	4 hours	Sergio

Acceptance Criteria:

- Given there is a search icon, when the user clicks on it, they are able to type into a search bar
- Given there is a search feature in the application, when the user types into the search bar with the name of a document that is saved, that document should pop up
- Given our application's GUI functions correctly, when the user types a part of the name of the document, then the most relevant documents should be at the top possibly alphabetically
- Given the unit test frameworks are functioning correctly, when the unit tests are run, then they test this appropriately, and the tests pass.

User Story #5

As a user, I would like to tag and categorize my documents and templates for easier organization.

#	Description	Estimated Time	Owner
1	Create an underlying document tagging system for our document	5 hours	Michael

	object.		
2	Implement a folder structure for user to organize their template	3 hours	Michael
3	Create an interface for displaying the tag and the folder	5 hours	Michael
4	Create unit tests for the folder and the tag system	3 hours	Michael

Acceptance Criteria:

- Given that the tag system is implemented correctly, when users want to add tags to their template, the tag should be searchable and can be used as filters within the application.
- Given that the folder system is implemented, when users want to organize their templates into folders, they should be able to easily navigate and find templates based on their folder structure.
- Given that there is an interface for tag and folder, when a user wants to search for a specific tag or click on the folder, relevant files should appear.
- Given the unit test frameworks are functioning correctly, when the unit tests are run, then they test this appropriately, and the tests pass.

Developer Story #1

As developers, we want the app to read through DOCX files (this is the parsing and processing modules).

#	Description	Estimated Time	Owner
1	Create a system to open, read,and parse through DOCX files	15 hours	Brennan
2	Parse through the documents to look for placeholders	15 hours	Brennan
3	Create a system to mark the locations of these placeholders for later processing.	8 hours	Michael
4	Create a system to replace these marked locations with provided text.	8 hours	Michael
5	Create unit test(s) to test that the expected and actual output from the parsing and processing modules are equivalent	7 hours	Sergio

Acceptance Criteria:

- Given the app can read local files, we want to read text from docx files for parsing
- GIven we parsed through the files correctly, when we finish parsing we want for the file to create
- Given that the process of replacing placeholders with desired text completed, the format and the structure of the document remain the same as before parsing.
- Given the unit test frameworks are functioning correctly, when the unit tests are run, then they test this appropriately, and the tests pass.

Developer Story #2

As a developer, I want to have a GUI setup for our EasyDraft program.

#	Description	Estimated Time	Owner
1	Create GUI design and include basic functionality	17 hours	Aneesh, Kye
2	Create unit test(s) with QTestLib to check the GUI looks right	6 hours	Sergio

Acceptance Criteria:

- Given we have the UI set up, when the user boots our application up, then they should see a welcome screen with recently-used templates.
- Given that each component of the user interface is complete, when the users view our application, then they should see that our components are fine-tuned to maintain a consistent style.
- Given we have the UI set up, when we add more features, then it'll be easy to tell the style and general look to make sure the new feature's graphical elements has.
- Given the unit test frameworks are functioning correctly, when the unit tests are run, then they test this appropriately, and the tests pass.

User Stories Not Included Thus Far

Functional Requirements

1. As a user, I would like to have a visually guided instruction to teach me how to use the software.
2. As a user, I would like to be able to specify a list of users for personalized bulk output (many PDFs, for example) from a text box within the program itself.
3. As a client, I would like to export the documents created in a PDF format.
4. As a user, I would like to be able to see the top three templates I use most often at the top of a list when selecting a template.
5. As a client, I would like to bold, italicize, or underline the text I input.
6. As a client, I would like to have more customization with the text I input, such as changing font style and size (if time allows).
7. As a user, I would like to save the information for placeholders I've used before to complete other documents.
8. As a user, I would like to preview documents before finalizing to ensure that all information is correctly placed.
9. As a user, I would like to choose from multiple export formats for my documents to meet different submission standards.
10. As a user, I would like the software to support multiple languages for the interface.
11. As a user, I would like to have a feedback button on the software to give developers suggestions or report bugs.
12. As a user, I would like my work to be auto-saved so that I don't lose progress when there is a system or software crash.
13. As a user, I would like to use already created or automated placeholders like dates and time.
14. As a user, I would like to be able to input where in the template should have placeholders.
15. As a user, I would like to select pre-filled inputs from a list that I have created to fill the template fields with.
16. As a user, I would like to send emails to each client when bulk outputting multiple documents. It could be a pdf document attached to an email or in the email itself. (if time allows).
17. As a user, I would like to be able to have a signature input box (if time allows).
18. As a user, I would like the application to be usable with a visual disability like including a high contrast mode.
19. As a user, I would like to create my own template document from scratch in the application without using Microsoft Word (if time allows).
20. As a user, I would like to have a dark mode.
21. As a user, I would like to have access to different language settings in the application.
22. As a user, I would like the option to print the finished document.
23. As a user, I would like the ability to organize and sort my templates into folders.
24. As a user, I would like the ability to see what kind of templates and how many I have created for a specific client.

25. As a user, I would like to be warned if I'm creating a template that is not completely filled.
26. As a user, I would like to receive regular updates for security and feature upgrades.
27. As a user, I should be able to use the application on different screen sizes and resolutions.
28. As a user, I would like to be able to preview a saved template before selecting it.
29. As a user, I would like to be able to connect to pre-existing cloud services to access my template
30. As a user, I would like to delete templates that I no longer need.
31. As a user, I would like to be able to set up my organization/company account.
32. As a user, I would like to be able to login with organization/company credentials (one login per group).
33. As a user, I would like to go through a visually-guided onboarding tutorial when the program first boots up to teach me how to use the program.
34. As a user, I would like to be able to launch the onboarding from some menu at any time so that I can refresh on my EasyDraft skills.
35. As a user, I would like to specify the way the input text for replacement should be split (with newline characters, commas, spaces, or otherwise) when working with the input module.
36. As a user, I would like to have the option to watermark the output documents for added branding and security.
37. As a user, I would like the ability to schedule automated template processing at specific times.
38. As a user, I would like the option to upload a document from Google Drive which is not contained in the commonly-used or recently-used sections of the EasyDraft Google Drive Folder.
39. As a user, I would like to bring a template file into EasyDraft directly from Google Docs through an EasyDraft Google Docs extension.
40. As a user, I would like to track the history of changes made to a template, including who made the changes and when.
41. As a user, I would like to be able to see the character count for each text field to ensure that I stay within specified limits and avoid the document looking improperly formatted.
42. As a user, I would like to be able to set and change the default values for template fields from within the application's GUI.
43. As a user, I would like to have keyboard shortcuts for the application's common actions.
44. As a user, I would like to be able to set default file path and filename upon exporting for each template.
45. As a user, I would like to see analytic information on how often each template is used and other numerical information about the exports for each template.

Main Function

Clients will create will have their own document created in Word with placeholders possibly with special language(eg. $firstName, $Date, $Employee_pay or something similar). They import their docx document in our application, they input information for the placeholders related to each receiver of the document, and then bulk output multiple documents each related to a receiver.

Architecture

A main goal is to make the application cross-platform for Windows, MacOS, and Linux. The plan is to develop the application with C++ as both the frontend and backend language. When developing the application, we will utilize frameworks that are designed for cross-platform development, like Qt.

Performance

The application will be designed with optimized code to ensure high performance in processing large amounts of documents. The response time when filling the template will be under 500 ms for smooth user interaction. The goal is to use less than 300 MB of memory. The software should be able to handle the generation of thousands of documents simultaneously. It is capable of processing and exporting up to 100 pages per document within seconds.

Security

The application will have modern security features and protections, keeping traffic between the cloud storage network and the local computer protected and secure.

Usability

The application will have an easy to navigate UI. Since our application is more experimental and not so similar to other products on the market, the organization should be simple. The application will be faster than using find and replace in a word processor to justify its use. It would support a template up to 100 pages.

Overarching Roadmap

- Sprint 1--Core functionality. GUI, one replacement per template field, do it through the GUI. We should have recently-used templates show up. The Minimum Viable Prototype is what we're getting here

- Sprint 2--Networking (sync with Google Drive), multiple replacements per template field, error handling, replacements from text file, tutorial,

- Sprint 3--Bells and whistles. Finish up anything we didn't get to and work on extra features, like other user settings, light/dark mode, maybe more cloud syncing platforms, etc.