Mission: Focus/Concentration Reading Application

DID: Software Development Plan (SDP)

PREPARED by: Group 4

Ayrton Massamba,
Sean Huber,
Joey Maggitti,
Ben Rieland,
Peter Stein,
and Leon White

DATE: 9/28/2020

1. Scope.

1.1 Identification.

This software is a focus/concentration software. Our plan is to find ways to stimulate reading habits. Time management in this exercise is therefore a central piece, as the goal is to isolate small chunks of time during one's day to enable constructive reading. Since we have observed that the lack of focus or attention during reading can lead to poor reading habits, a tool to improve focus would be very helpful. In conclusion, the software at its basic core is very simple, but the different features that can be added to it could be very beneficial.

Potential names of the software/Titles: Reading Space, Reading Room, TBR (short for To-Be-Read), my TBR list. Reading helper, reading guide

Identification number: 2.0

Current version of software: Reading Space 2.csproj

Release number: 0

1.2 System overview.

The purpose of the system/software is to promote active reading. The system works as a book database search engine, a follow-up checklist, a timer, a statistics report application, and a reader-reward-tool service.

The book database helps with the search of various book titles, authors/writers, and book covers/images (The data then can be added to the follow-up checklist, if a client wish to do so.), the follow-up checklist is to be used to keep track of the diverse books of interest to the users (e.g. To-Be-Read,Completed, etc), the timer is used to organize reading time efficiently throughout the day (cutting a reader's reading objective in chunk of 15-30 minutes based a set schedule or a everyday 30 minutes a day reminder), a statistics report in then formulate to represent the user current and past usage of the application, and based on the use of the reader the reward system offers different badges/avatars (images).

The general idea of the software was first generated by Ayrton Massamba, but later on was polished by the team, which members are: Ayrton Massamba, Sean Huber, Joey Maggitti, Ben Rieland, Peter Stein, and Leon White. Pre-development, the team envisioned to use a

programming platform such as visual studio or Unity. Moreover, the initial programming language selected for the software was C-Sharp on the visual studio platform. Operation for this project was to be done remotely via platforms such as Discord, Google Doc, and Blackboard Collaborate. Moreover, as a means of research/tutorials, various YouTube videos are promoted. The target demographic for this project are people with the desire to read, but who either lack motivation or time. Nonetheless, the team believes that potential other demographics can be essential users.

1.3 Document overview.

- Section 1: Scope
- Section 2: Referenced documents
- Section 3: Overview of required work
- Section 4: Plans for performing general software development activities
- Section 5: Plans for performing detailed software development activities
- Section 6: Schedule and activity network
- Section 7: Project organization and resources

Privacy and security concerns are considered beyond the scope of this project.

1.4 Relationship to other plans.

No other plans for this project other than general outlines have been developed at the time of this document. Research on how to approach creating an app on Unity and a decision to use Unity over C# have been determined.

2. Referenced documents.

Youtube tutorial videos were used for this project and therefore they will be cited as sources/references. The group decided on the videos as a way to accommodate the team on a peculiar coding platform. Moreover, these videos served as a way of communicating ideas on which programming language should be selected. Therefore, the initial and current videos used so far are as follow:

Initial videos (for software identification number: 1.0)**:**

1) **How to Developed Application for Fast Foods Restaurant in C Sharp**

https://www.youtube.com/watch?v=peb7kO3GDuQ&t=1045s

2) Designing a Modern Flat Desktop Application of a Fast Food Restaurant in Visual Basic VB NET

https://www.youtube.com/watch?v=znANTJNf8wI&t=522s

3) Modern Flat UI, Drop-down/Slider Menu, Side Menu, Responsive, Only Form - C#, WinForm

https://www.youtube.com/watch?v=JP5rgXO_5Sk&t=79s

4) C# Tutorial - Circle Progress Bar | FoxLearn

https://www.youtube.com/watch?v=o7MGaf8YW6s

Current videos (for software identification number: 2.0)**:**

1) **How to make UI in UNITY - EASY TUTORIAL**

   https://www.youtube.com/watch?v=_RIsfVOqTaE

2) **Making UI That Looks Good In Unity using Color Palettes, Layout Components and a Blur Panel**

   https://www.youtube.com/watch?v=HwdweCX5aMI

3) **START MENU in Unity**

   https://www.youtube.com/watch?v=zc8ac_qUXQY

3. Overview of required work.

1. Software needs to work with book database(s) and be captivating enough to keep users interested in the product.

2. A series of video tutorials will need to be thoroughly studied to be able to conceptualize the project.
3. The position of the project in the system life cycle currently between system planning and analysis.
4. The selected program/acquisition strategy or any requirements or constraints on it

   The selected program needs to be able to attract about 60% of active readers, while the other 40% maybe passive readers, occasional readers, or even beginners.

5. The team needs to meet at least twice a week during school hours, as for the hours of conceptualization the software teams need to put in at 5 hours of work a week for each assignment given. If team members are not working independently, they must report on their progress every Friday at 4pm on Discord.
6. For privacy and security reasons, Discord conversations are not to be shared with the public, unless authorized by the team/group. The team will also evaluate any security concerns related to user privacy, and will establish appropriate methods and standards. And as for interdependencies in hardware, an outside consultation may be necessary. Software development will be the main responsibility of the team without, of course, neglecting other important aspects.

4. Plans for performing general software development activities.

4.1 Software development process.

Due to the short time for creation of the project, we will follow an agile software development process. This will allow us to handle customer input and programming or feature roadblocks on a week-sized timescale. It also emphasizes unit testing, which meshes well with chunking up the program for development.

The chunking strategy will be applied to feature development to be able to distribute development work with less, more purposeful integration. This can enable specific developers to have more intimate knowledge of their assigned features and increase development speed. However, care has to be taken in planning integration to avoid later restructuring.

4.2 General plans for software development.

4.2.1 Software development methods.

All code will be developed using the team voted development systems. In this case, the team agreed on working on the platform Unity with C# or C++ as based programming languages. Nonetheless, team members can propose editors or tools that will facilitate such development. Issues and bugs may be resolved using the proposed editors or tools. But most encouraging are thorough discussions before jumping into development technical aspects. All features on Unity need to be exploited to the fullest so the team can provide quality.

The team can make use of source codes found on the internet, and add them to original code. Using collaboration platforms such as GitHub is appreciated, but should not be exaggerated. The features that these websites/platforms offer are to be considered complementary or even inspirational.

Here are some GitHub review procedure that may be of interest to the group:

Style Guide: https://www.python.org/dev/peps/pep-0007/Source Version

Standard: https://semver.org/Source

Repository: https://github.com/ddkauffman/FSW_SPR_2020

Moreover, here are some **UMBC coding standards** that the team can use for reference:

https://www.csee.umbc.edu/courses/undergraduate/CMSC201/spring17/docs/CMSC%20201%20-%20Python%20Coding%20Standards.pdf

https://www.csee.umbc.edu/~chang/cs341.f16/projects/coding-standards.shtml

4.2.2 Standards for software products.

1. Tabs shall be used to indent code instead of spaces
2. Header comments should include the name of the file, a timestamp of the latest revision (and revision number), and a brief description that includes any inputs or outputs.
3. Functions will be preceded by a comment explaining what the function does, the importance of the parameters and the return value.
4. Variables shall be written in camelCase and functions in pascal case, i.e. int myNumber; float MyFunction();
5. For naming conventions not already listed, refer to appropriate above PDF documentations.

4.2.3 Reusable software products.

4.2.3.1 Incorporating reusable software products.

To incorporate reusable software, like API's, into the project, developers should follow a simple process. Once reusable software that fits the requirements is found, a quick search should be completed to look for alternatives. Any alternatives found should be compared against the first piece and against writing original code for the purpose. It should be compared over the criteria of: ease of use, permission, features, compatibility, and overhead. Once one strategy is determined to be the best method of implementation, it should be reviewed by at least one other member, then it is free to be implemented.

Currently, the only planned reusable software to be used is Unity, a software engine which may handle much of the background functions of a program. It should save development time by making creating a user interface significantly easier. One drawback is that other API's that may be incompatible with Unity.

4.2.3.2 Developing reusable software products.

There is no plan within this project to develop reusable software. The software developed will likely be application specific. But if a section of the software is determined to be reusable outside of this project, this topic shall be revisited to determine permission or credit of such software.

4.2.4 Handling of critical requirements.

The application does not provide any security assurances. The developers do not intend to collect customer data. Therefore, privacy assurances are not applicable.

4.2.5 Computer hardware resource utilization.

The project will not be intensive on the device's hardware. As long as the device is capable of running Unity, system resources should not be a problem. During testing it will be confirmed that the project will run without using excessive system resources.

4.2.6 Recording rationale.

Any "key decisions" determined by the team shall be recorded for future reference. A "key decision" is a decision that presents a structural change in the format or design of the project, a change in requirements, or a selection of reusable software. The outcome of the decision and the rationale for and against it shall be put in writing in a shared "Decisions" document in the shared drive.

4.2.7 Access for customer review.

The program will be able to be compiled and sent to the customer via github, email, or drive (whichever works best). It should be portable and able to run on their computer. In such a case where it is not, screen-share capabilities will allow customer review remotely.

5. Plans for performing detailed software development activities.

5.1 Project planning and oversight.

Plans will be updated as development progresses and changes will be written in the development documents and in the discord server. Updates will be pushed to different branches on the github repository. Eventually, after approval, the branches will be merged with the main branch. Review may be done by peers, however the team does not have a specific individual who explicitly does review.

Testing will be done inside of the Unity Engine. Testing results will be recorded and put into a shared folder for other developers to see.

As for transitioning, Unity is a cross-platform engine which should make transitioning simple. Any operating system which Unity supports should be able to develop and use the application.

5.2 Establishing a software development environment.

The software will be made and tested within the Unity engine. Github will also be used for source control with the project. Unity also has an entire framework made for unit testing explicitly which will be used to ensure the application will not crash even with unexpected input.

5.3 System requirements analysis.

The application will need to track the user’s cursor so they may select different buttons on the screen. Keyboard input will also need to be tracked when the application is in specific states (asking for the name of the book). User input should be fairly easily done through Unity.

The application should require minimal system resources. Testing will be done to ensure the application is not using up more resources than necessary.

5.4 System design.

The project will be written in Unity which is cross-platform capable as well as mobile device compatible. Developers should be able to work on the application so long as they have a Unity compatible operating system and users should be able to use the application with a Unity compatible operating system.

The software itself when completed will be on the customer's system running. There will not be any need for running our own database/servers. The software may externally search for information on the readers' books (such as the cover image of the book). However, we will not be providing this service using our hosted systems.

5.5 Software requirements analysis.

Our requirements will be very clearly described and will be verified through testing/demonstration. Users will know the minimum requirements are fulfilled when the application is successfully tracking reading time, providing accurate statistics, and keeping an accurate checklist of books.

Should we continue beyond our original requirements, users will know requirements are fulfilled when the application successfully notifies users when it is time to read, awards users badges correctly, and has a functional do not disturb feature.

5.6 Software design.

Different structures for program implementation will be considered after the requirements are finalized. One likely option involves a core program with components for other features.

To determine design details, there are no concrete preferences/rules which apply to the application. Members will consider different designs and debate and survey amongst themselves and the customer to try to determine what the final design will look like.

The application will be written in Unity to promote cross-platform compatibility with the potential of the application being mobile-compatible. Members will do their best to design the application for more seamless platform transitioning.

5.7 Software implementation and unit testing.

To ensure that the application is fully functional and will not crash due to unexpected circumstances, the application will undergo unit testing throughout the development process.

Unity has an entire framework made explicitly for unit testing. Using the unit testing framework, members will be able to ensure that the application will function even with unexpected input.

For more details on unit testing, see 5.8.1 - 5.8.4.

5.8 Unit integration and testing.

5.8.1 Preparing for unit integration and testing

To set up components, specific inputs outputs and context should be described. This will later give developers working on specific components something to test against.

5.8.2 Performing unit integration and testing

The core of the program will be developed first, giving the features to be tested a base from which to run. From there either real or false data can be sent to the component to check for accurate output. If each component accepts the proper inputs and gives the proper outputs, by individually testing, many integration problems can be avoided. As components pass unit testing and are attached to the core program, the entire bundle should be tested to confirm data lines up and to weed out logic errors.

5.8.3 Revision and retesting

If more values are needed or the project structure changes, the changes should be recorded in the applicable documentation, and adjusted components should be retested.

5.8.4 Analyzing and recording unit integration and test results

The tests to be run should be recorded in design and the output from the tests will be recorded.

5.9 CSCI qualification testing.

This project will not be divided into components independent enough or large enough to warrant classification of a CSCI.

5.10 CSCI/HWCI integration and testing.

This project will not be divided into components independent enough or large enough to warrant classification of a CSCI, thus does not warrant CSCI/HWCI integration or testing.

5.11 System qualification testing.

The developers will be testing their own code as they go to ensure bugs are found earlier rather than later. Developers may choose to have another developer test the code for a second opinion/attempt.

Because the application is being made in the Unity engine, testing is not needed for other devices until a working prototype of the application is ready. Once testing on one device is finished, testing on other devices should not be necessary for any purpose other than proper formatting.

Initial testing will begin with preparing multiple Android and desktop devices with the application/source code. Testing will include: basic functionality testing of book/file output, moving to different pages, reader statistics, and potentially other functionality not yet considered.

Testing will result in identifying bugs which will require the original writer of that section to address and fix. The results of testing will be recorded. Potential documentation of test results could be screenshots of test environments, application output, and full error reports on any failures.

5.12 Preparing for software use.

Preparing the executable software should be fairly easy in the Unity environment. There is little manual interaction required to go from source to execution.

Version control will be done through github. The project is not large enough for a user site to be deployed.

Due to the Unity engine, installation should be fairly straightforward. If necessary, a document may be written on how to install the application.

5.13 Preparing for software transition.

Preparing executables should be fairly easy in the Unity environment. There is little manual interaction required for developers.

Source files will be readily available in the github repository and within the Unity project. Github will be used for version control and updating the project.

This project is not incorporating the CSCI design because it is too small. The project is simple and will not require a manual. The customer may contact developers for support. A readme file will be included with the program.

5.14 Software configuration management.

All code for the project will be stored and maintained in a github repository. Github is a great tool for auditing changes to the code, maintaining a backup in the event of a failure, version control, clean documentation, issue tracking, and code approval.

5.15 Software product evaluation.

The project will be evaluated as time progresses by both the team and the customer. All evaluations will be recorded and changes will be made following an evaluation if necessary.

5.16 Software quality assurance.

All quality assurance evaluations will be performed by the development team and recorded. Testers should purposely try bogus/unexpected actions/data in addition to normal user behavior to ensure the application works as designed. Unit testing will also help to provide bogus data (see 5.5).

Developers will record items to be tested as they continue to work on the project. Doing so will allow developers to unit test the application thoroughly and conceptualize some of the base cases that need to be tested to confirm the application is running as expected.

Members of the team may request for other members' of the team to test the portion of the project they designed. This will help maintain independence in quality assurance and potentially allow for more creative testing ideas. However, members may choose to test their section alone.

5.17 Corrective action.

The following items will be recorded should an error occur: project name, originator, problem number, problem name, software element or document affected, origination date, category, priority, description, tester name, date assigned, date completed, analysis time, recommended solution, impacts, problem status, follow-up actions, corrector, correction date, version where corrected, correction time, description of solution implemented

Issues may be created and processed through the github repository. Issue tracking through github will allow developers to track issues in a central location and check the status easily.

5.18 Joint technical and management reviews.

Joint technical reviews will occur in the team space and will cover project status and will revisit future goals. They may also cover any branch progress and potential merges. Developers will work off separate branches within the github. Developers must submit a pull request before the changes are merged with the main branch. Pull requests must be approved by another member. Joint management reviews will mostly be held asynchronously in the form of status reports. When these meetings must be synchronous with the professor, they will be conducted on Blackboard Collaborate.

5.19 Other software development activities.

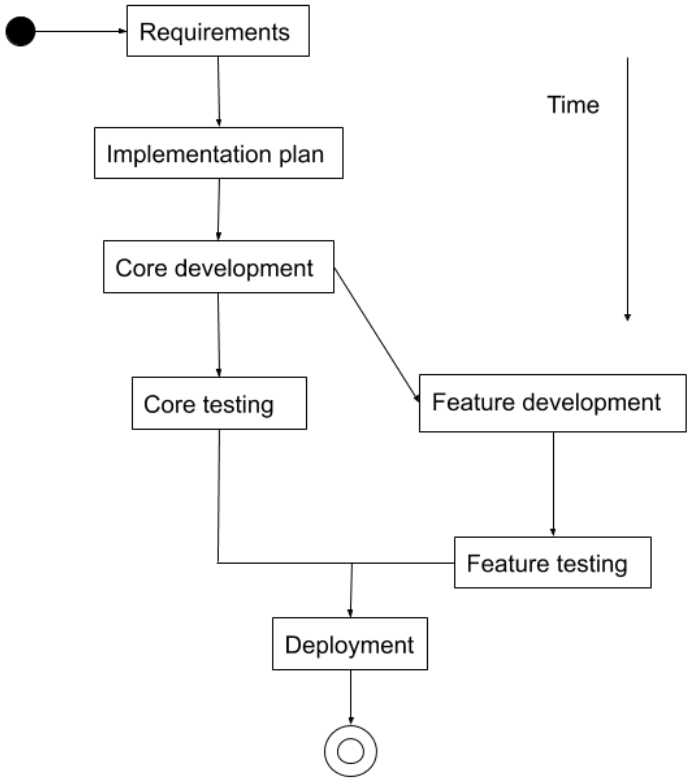
Other sections of this document cover many of the ideas covered in this section. Some ideas that aren't covered include: Subcontractors, which are not involved in this project. Interfacing with verification and validation agents; verification and validation will be conducted on a per-requirement base, without independent agents or software. And improvement of project processes, which when an improvement is decided upon, a decision document will be created, and relevant documents will be updated.

6. Schedules and activity network.

Project Checkpoints:

Date	Task to complete	Task to begin	Phase
29 Sept. 2020	Have SDP Completed	Plan a meeting with customer, arrive with draft of requirements	0
9 Oct. 2020	Finalize requirements, have met with customer		0
13 Oct. 2020	Have SRS Completed	Plan implementation, set up development software and test its functionality	1
20 Oct. 2020	Have an implementation plan complete, begin development		1
27 Oct. 2020	Have SDD Completed	Begin testing core components	2
7 Nov. 2020	Have working prototype at 90%		3
10 Nov. 2020	Have STD Completed	Begin intensive testing for all features	3
20 Nov. 2020	Have customer final approval, potentially have product deployed		4
24 Nov. 2020	Have STR Completed		4

Activity Diagram:



7. Project organization and resources.

7.1 Project organization.

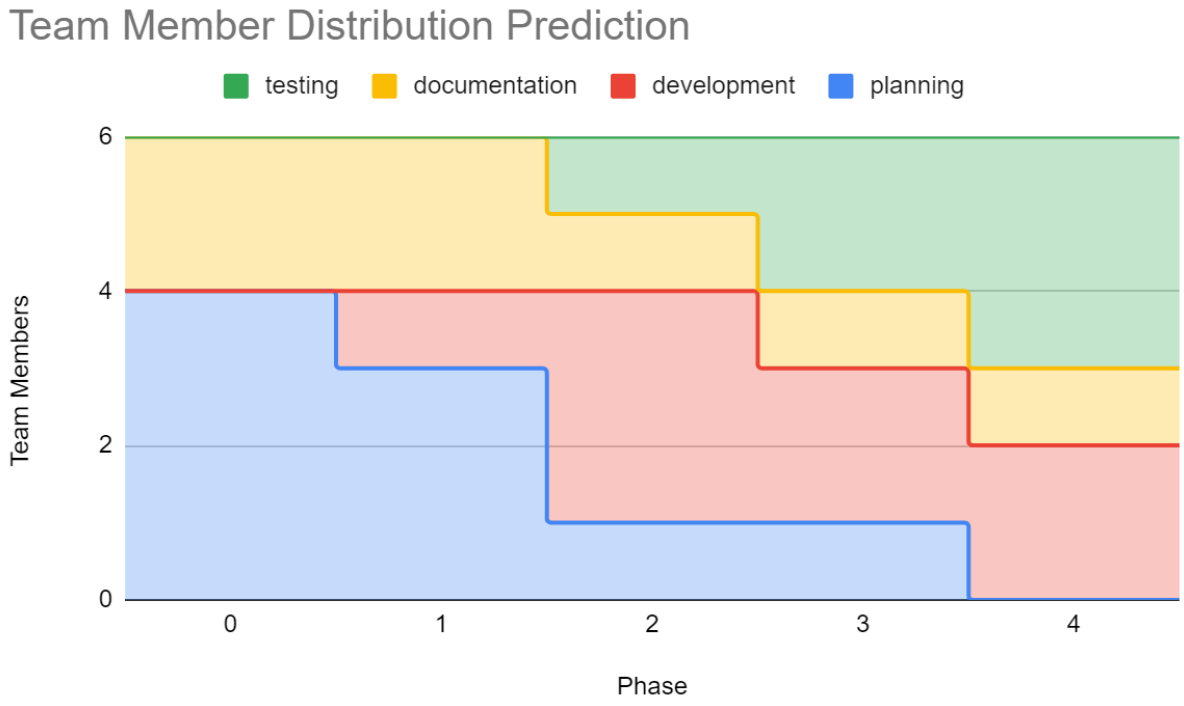
The project is divided up into five phases. Phase 0 is the conceptualization of the project, and meeting the customer to lay out requirements. Phase 1 is planning the implementation for the project, and improvement of the plans of phase 0. Phase 2 is the heart of development, and testing of components will begin concurrently. Phase 3 is when the core of the program is complete. This will be subjected to testing as more features are developed. Phase 4 is when we want to have a program that fulfills the requirements, and we will transition to a heavier focus on testing to find and remove as many bugs as possible.

Work for the project will be divided up on a per-phase basis,with team members roughly assigned to particular goals. During development and testing, roles will be more clear, as testing and development of certain features can be distributed to specific team members.

There will be a customer of the project. They will review requirements at the end of phase 0, then throughout the project they can look at the product and bring up features they would like to see in the product, as well as problems and questions they might have from a consumer perspective.

7.2 Project resources.

The personnel resources for this project consist of 6 group members. This number will be steady through the project, but the workload will not, so members will be distributed according to the load between development and testing. See the table for predicted team member distribution. Due to restrictions on in-person meetings, the team members will meet and work online, irrespective of geographic location.



The technological resources to be used for this project consist of various devices that the team members have access to. Unity is available for download, and should be installed and functional by the time requirements are set. Testing equipment like mobile devices and different PC’s and monitors are owned by the development team and can be used when required. To

share these resources if required when testing, a developer's computer running the program can be screen-shared to other development members.

The last resource to consider is time. Overall project time is limited due to a hard deadline, and developer time is limited due to concurrent projects and external responsibilities. It is important to be time-efficient because of these constraints, so asynchronous work will likely constitute a significant portion of the project.

8. Notes.

Glossary of document acronyms:

- CSCI - Computer Software Configuration Item
- HWCI - Hardware Configuration Item
- SDD - Software Design Description
- SDP - Software Development Plan
- SRS - Software Requirements Specification
- STD - Software Test Description
- STR - Software Test Report