*CS3354 Software Engineering Final Project*

*Deliverable 2*

**Readary**

(An App for Book Lovers)

**The Group Members**

Aisha Ashfaque

Dante Moreno

Emily Wojciechowski

Joel Francis

Junaid Hashmi

Priscilla Adomako

**Date**

March 15, 2019



# 

# Feedback on Project Draft comments:

Our proposal was approved without a recommendation for any change. However, our plan will be adjusted based on the fact that one of our team members, Michael Elizalde, has joined another project team and is no longer part of our team.

Based on the comments we received, we surveyed and found several similar apps already in use. The most surprising discovery was to find an app named *Bookish*, our proposed app name. Consequently, we have renamed our product to *Readary.* Other than reshuffling of responsibilities due to loss of a team member, this is the biggest change we incorporated.

Since we found similar products already in use, we will do a detailed comparison as part of our final project report. For now, we would like to mention an additional feature to be included in our project in order to distinguish it from other similar products in the market.

* 1. Addition to previously stated purpose

We will incorporate a MUTE function in the app to allow users to silence notifications as follows:

* + 1. Audio notification may be turned on/off in application setup
    2. Notification messages may be muted (turned on/off in application setup)
    3. Notification messages may be muted until a future date provided by user

***NOTE:*** URL for our project GitHub is: <https://github.com/CS3354-Readary>

# Task Delegation:

|  |  |
| --- | --- |
| Name | Task Delegation |
| Dante Moreno | Team Proposal, Create GitHub Repository and Add Team Members, Software Requirements, Addressing Feedback, Presentation |
| Aisha Ashfaque | Team Proposal, Software Requirements, Cost Effort and Pricing Estimation, Software Test Plan, Presentation |
| Junaid Hashmi | Commit README, Architectural Design, Project scheduling, Citing references, Presentation |
| Joel Francis | Team Proposal, Class diagram, Project Scheduling, Software Requirements, Software Test Plan, Presentation |
| Priscilla Adomako | Team Proposal, Sequence Diagram, Cost Effort and Pricing Estimation, Comparison and Conclusion, Presentation |
| Emily Wojciechowski | Project Scope file, Use Case Diagram, Citing References, Comparison and Conclusion, Presentation |

# Software Process Model:

The software process model that will be employed with this project is the spiral model. The spiral model contains development cycles, in which risk management is considered before employing an addition to a prototype. Since Readary’s architecture features multiple systems interacting with a database, it would make sense any risks are considered before implementing to ensure that no additional feature negatively affects the database.

# Software Requirements:

## Functional Requirements

1. A user holding an author account shall be able to submit book titles along with one summary and cover of that book to be added to the repository.
2. A user shall be able to search for books solely on genre, author, reviews and ratings, or other tags or a combination of the four.
3. A user shall be able to add and remove genres, authors, or other tags and interests from their book preferences.
4. The system shall generate each day, for each user, a new list of books based on each user’s preferences and the books each user has tagged.
5. Each user shall be uniquely identified by their member ID.
6. A user shall be able mute notifications until a specified date or indefinitely.

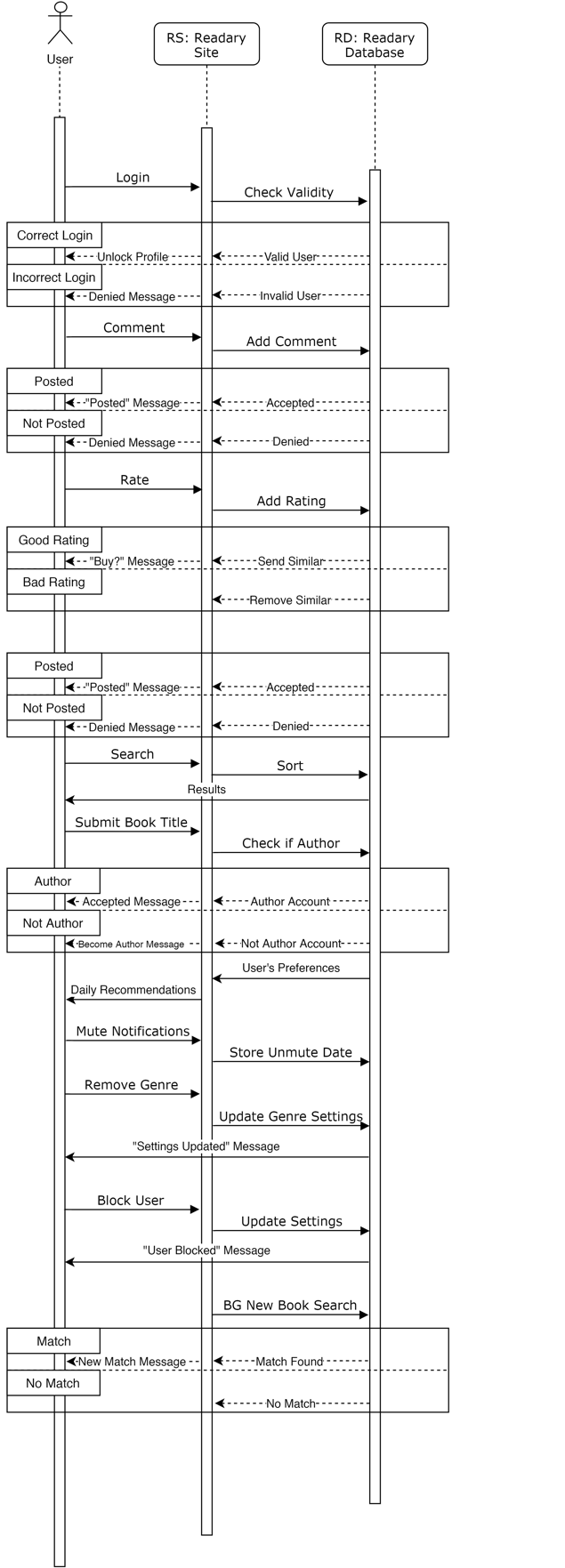
## Non-functional

1. **Product Requirements**
   1. **Usability**
      1. Users shall be able to access all functions from the main menu in less than 3 clicks.
      2. 95% of users will be able to search for a book and add a new interest tag by the third attempt without requiring assistance.
      3. The system shall be accessible to users with vision needs, specifically, users shall be able to increase the font size for the entire user interface.
   2. **Efficiency**
      1. Performance
         1. Responses to actions shall take no longer than 5 seconds to load on to the screen.
         2. The system shall be able to process a single notification in less than 1 second.
      2. Space
         1. The system shall not require more than 1 GB of space when downloaded on the users device.
   3. **Dependability**
      1. In the case of system failure, less than 0.1% of data shall be lost.
      2. The system shall be operational 95% of the time
      3. In case of system failure, down time shall be 2 hours or less.
   4. **Security**
      1. User account passwords must be at least 8 characters long.
2. **Organizational Requirements**
   1. **Environmental**
      1. The system shall be compatible with both iOS and Android.
   2. **Development**
      1. The system shall be developed using the programming language Java.
3. **External Requirements**
   1. **Legislative**
      1. The system shall implement and abide by copyright laws for each book that is requested to be submitted to the repository.

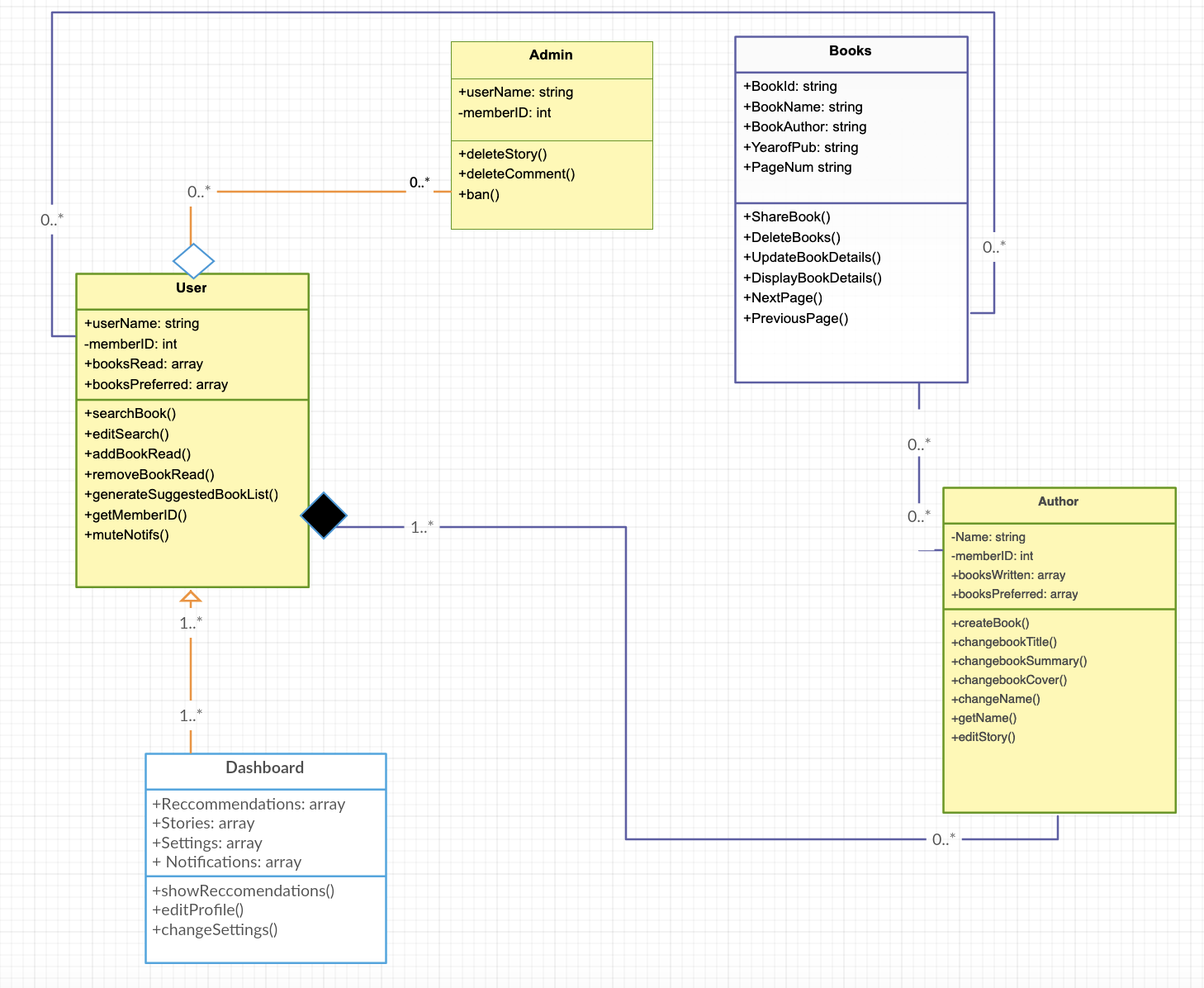
# Use Case Diagram:



# Sequence Diagram:



# Class Diagram:



# Architectural Design:

The central part of Readary is a centralized DBMS. Therefore, our architectural choice is a central repository pattern. A high level architecture is shown below:

User

Authentication &

Preferences

REPOSITORY

Analytics

 Author/Publisher

Books/Summary

Likes/Dislikes

Reviews/Critiques

Apps Interface

Web Interface

Admin

Web Services and 3rd party interfaces

# Task Delegation Update:

|  |  |
| --- | --- |
| Name | Task Delegation |
| Dante Moreno | Team Proposal, Create GitHub Repository and Add Team Members, Software Requirements, Addressing Feedback, Test Plan, Presentation |
| Aisha Ashfaque | Team Proposal, Software Requirements, Cost Effort and Pricing Estimation, Presentation |
| Junaid Hashmi | Commit README, Architectural Design, Program Management, Project Scheduling (Plan), Citing References, Competitive Analysis, Report Reviews, Presentation |
| Joel Francis | Team Proposal, Class Diagram, Software Requirements, Software Test Plan, Presentation |
| Priscilla Adomako | Team Proposal, Sequence Diagram, Cost Effort and Pricing Estimation, Power Point Preparation, Presentation |
| Emily Wojciechowski | Project Scope File, Use Case Diagram, Citing References, Conclusion, Presentation |

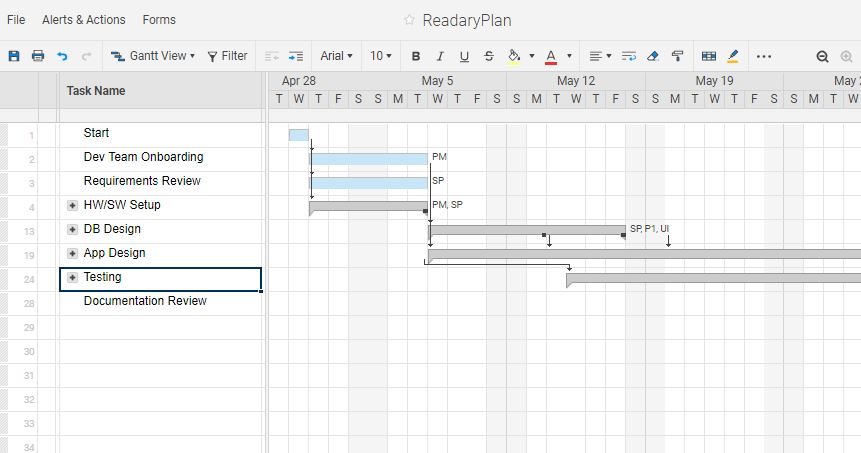
# Deliverable 1:

The entire Project Delieverable1 Report has been included in previous pages.

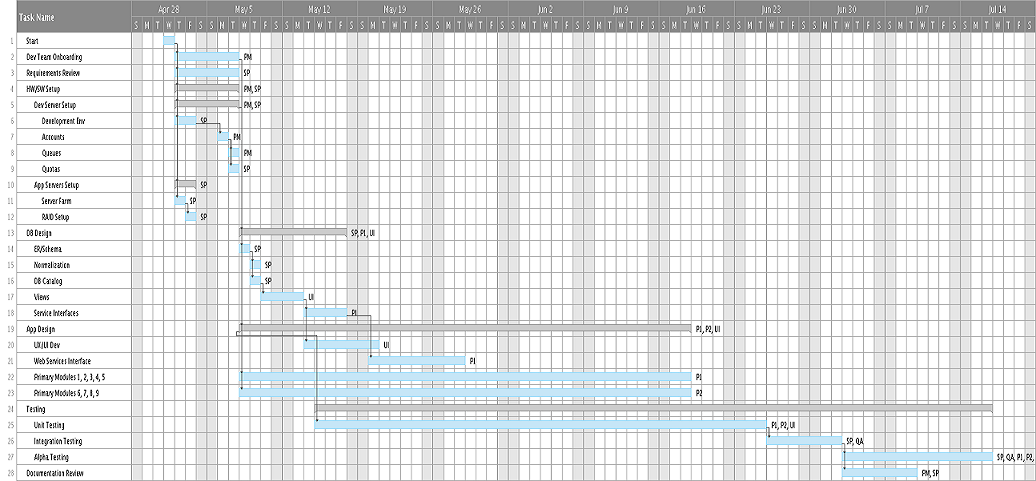
# Project Scheduling, Cost, Effort, and Pricing Estimation:

## 3.1 Project Schedule (Plan)

### Timeline Summary View:



### Timeline Detail View:



### Tasks/Schedule:

|  |  |  |  |
| --- | --- | --- | --- |
| Task Name | Duration(d) | Depend | Assigned To |
| Start | 1d |  |  |
| Dev Team Onboarding | 4d | 1 | PM |
| Requirements Review | 4d | 1 | SP |
| HW/SW Setup | 4d | 1 | PM, SP |
| Dev Server Setup | 4d | 1 | PM, SP |
| Development Env | 2d | 1 | SP |
| Accounts | 1d | 6 | PM |
| Queues | 1d | 7 | PM |
| Quotas | 1d | 7 | SP |
| App Servers Setup | 2d | 1 | SP |
| Server Farm | 1d | 1 | SP |
| RAID Setup | 1d | 11 | SP |
| DB Design | 8d | 2 | SP, P1, UI |
| ER/Schema | 1d | 5 | SP |
| Normalization | 1d | 14 | SP |
| DB Catalog | 1d | 14 | SP |
| Views | 2d | 16 | UI |
| Service Interfaces | 4d | 17 | P1 |
| App Design | 30d | 5 | P1, P2, UI |
| UX/UI Dev | 5d | 17 | UI |
| Web Services Interface | 7d | 18 | P1 |
| Primary Modules 1, 2, 3, 4, 5 | 30d | 5 | P1 |
| Primary Modules 6, 7, 8, 9 | 30d | 5 | P2 |
| Testing | 45d |  |  |
| Unit Testing | 30d | 19SS +5d | P1, P2, UI |
| Integration Testing | 5d | 25 | SP, QA |
| Alpha Testing | 10d | 26 | SP, QA, P1, P2, UI |
| Documentation Review | 5d | 26 | PM, SP |

## 3.2 Cost, Effort, and Pricing Estimates:

For our cost, effort, and pricing analysis, we used Function Point method. This method analyzes the system’s complexity and productivity to estimate what the project will cost and the duration of the project. This method uses function points to determine the cost and duration of the project. [1]

Then the Processing Complexity adjustment (PCA) is determined by:

*PCA = 0.65 + 0.01 (PC1 + PC2 + ... + PC14)*

The Function Points (FP) for the project is calculated using the formula:

*FP = GFP \* PCA.*

Table below shows the count and the complexity associated with each category and are used to compute the GFP. Following this, another table provides the 14 questions and our estimated weights assigned to each.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Category** | | **Count** | | **Complexity** | | | | | |
| Simple | | Average | | Complex | |
| Number of User input | | 28 | | 3 | | 4 | | 6 | |
| Number of user output | | 15 | | 4 | | 5 | | 7 | |
| Number of user queries | | 32 | | 3 | | 4 | | 6 | |
| Number of data files and relational tables | | 18 | | 7 | | 10 | | 15 | |
| Number of External Interfaces | | 24 | | 5 | | 7 | | 10 | |
| **PC** | **No Influence** | | **Incidental** | | **Moderate** | | **Average** | | **Significant** | | **Essential** |
| Does the system require reliable backup and recovery? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Are data communications required? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Are there distributed processing functions? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Is performance critical? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Will the system run in an existing, heavily utilized operational environment? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Does the system require online data entry? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Does the online data entry require the input transaction to be built over multiple screens or operations? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Are the master files updated online? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Are the inputs, outputs, files, or inquiries complex? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Is the internal processing complex? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Is the code designed to be reusable? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Are conversion and installation included in the design? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Is the system designed for multiple installations in different organizations? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |
| Is the application designed to facilitate change and ease of use by the user? | 0 | | 1 | | 2 | | 3 | | 4 | | 5 |

### Effort Calculations:

*GFP* ***=*** *28x4 + 15x5 + 32x3 + 18x15 + 24x10* ***= 793 FP***

*PCA**= 0.65 + 0.01 (7x3 + 2x5 + 3x2 + 2x2)* ***= 1.06 PCA***

*FP = 793 x 1.06 =* ***840.58 FP***

### Estimated effort (E)

Assumed productivity of 15 function-points per person-week.

*E = FP/ productivity*

*840.58 / 15 = 56.04* ***= 57 person-weeks***

### Project Duration (D)

*D= E / team-size = 57 / 6* ***= 10 weeks***

### Summary of calculations:

The estimated function point for our project is about 841FP. Assuming productivity of 15 function-points per person-week, we determined that the estimated effort is 57 person-weeks. With a team size of 6 for the project, the project duration is estimated to be about 10 weeks. Please note, these estimates do not include the time required to acquire and setup hardware, software, and/or arrange any required training for one or more members of the development team.

## 3.3 Estimated Cost of Hardware:

If we plan on using AWS or another cloud-based server option, on average, the cost comes out to around $1,475 per month [9]. Since we plan to support this product for a long time, we chose to establish in-house development and support infrastructure.

|  |  |
| --- | --- |
| Hardware | Cost |
| Development Server with required terminals for the team | $ 23,600 |
| DB Server | $ 18,750 |
| RAID-6 | $ 12,000 |
| Subtotal: | $ 54,350 |
| Maintenance Service Agreement (annual) | $ 1,000 |
| *Total:* | *$ 55,350* |

## 3.4 Estimated Cost of Software:

The estimate below pertains to setting up the development environment only. It does not include annual service fees for publishing the App on various platforms [11], annual fees paid for some required Web Services, or software maintenance cost [10], except for the first year.

|  |  |
| --- | --- |
| Software | Cost |
| Development Server & IDE + Utilities | $ 2,600 |
| DB Server/Client and Integration Libraries | $ 4,550 |
| System and Network Security | $ 1,500 |
| Subtotal: | $ 8,650 |
| Maintenance/Support/Subscriptions (annual) | $ 1,320 |
| *Total:* | *$ 9,970* |

## 3.4 Personnel and Training Cost:

After researching average salaries and hourly wages in the United States, an hourly wage was determined. The varying hourly wages reflect the amount of experience each team member has and is based on the average wages in the United States found during research. [2] [3] [5] [6]

Team members (6) and estimated hourly rates:

|  |  |
| --- | --- |
| Role/Purpose | Rate |
| Project Manager (PM) | $ 160/Hr |
| Senior Programmer and DBA (SP) | $ 120/Hr |
| Programmer 1 Web Services (P1) | $ 100/Hr |
| Programmer 2 (P2) | $ 100/Hr |
| QA/Test Engineer (QA) | $ 75/Hr |
| UI/UX Designer (UI) | $ 140/Hr |
| Training (one-time) | $ 6,000 |

Assuming 40 hour work-week and taking the estimated project duration to be 10 weeks from the earlier calculations, the cost of personnel while developing the project would amount to:

*Personnel Cost = $695/hr x 40hr/wk x 10wk = $278K*

*Total Cost = Personnel Cost + Training = $278K + $6K =* ***$284K***

# Test Plan:

The method *userInput()* asks and takes user input and stores it into three variables (there can be more, but for the sake of testing purposes we will be using just three). For the plan, we are using the equivalence partitioning method that is described in *black box* testing. The variable *userGenre* is the user’s input for their favorite genre, *userAuth* stores the user’s favorite author, and *userBook* is the user’s favorite book.

All these variables will undergo a type of validation to see if the user left these entries empty (***userGenre.isEmpty()****,* ***userAuth.isEmpty()*** and ***userBook.isEmpty()***), or if the user inputs numbers where they should not (***userGenre.matches(“[0-9]+”*)** and ***userAuth.matches(“[0-9]+”)***), or both.

Once the *userInput()* method is done, the program should have the user’s favorite book, author, and genre for the program to use for matchmaking. JUnit was used to test the unit. The unit came back with no errors in setup time of 0.000 seconds and runtime of .094 seconds. The output and JUnit results are pictured below. A copy of the code is located in the zip file.

### Cases:

1. **userGenre.matches(“[0-9]+”)** 
   1. **Partition 1:** If the user input has a number, repeat until the user does not.
   2. **Partition 2:** If the user does not input a number, then continue.
2. **userGenre.isEmpty()** 
   1. **Partition 1:** If the user does not input anything, repeat until the user does.
   2. **Partition 2:** If the user does input something, then continue.
3. **userGenre.matches(“[0-9]+”)|| userGenre.isEmpty()**
   1. **Partition 1:** If the user does not input anything, repeat until the user does.
   2. **Partition 2:** If the user does input something, but it’s a number, repeat until the user does.
   3. **Partition 3:** If the user does input something, and it does not have a number, then continue.
4. **userAuth.matches(“[0-9]+”)**
   1. **Partition 1:** If the user input has a number, repeat until the user does not.
   2. **Partition 2:** If the user does not input a number, then continue.
5. **userAuth.isEmpty()**
   1. **Partition 1:** If the user does not input anything, repeat until the user does.
   2. **Partition 2:** If the user does input something, then continue.
6. **userAuth.matches(“[0-9]+”) || userAuth.isEmpty()**
   1. **Partition 1:** If the user does not input anything, repeat until the user does.
   2. **Partition 2:** If the user does input something, but it’s a number, repeat until the user does.
   3. **Partition 3:** If the user does input something, and it does not have a number, then continue.

# 

# 

# Product Comparison:

Following is an overview of comparison between similar products found:

|  |  |  |  |
| --- | --- | --- | --- |
| FEATURES | Readary | Goodreads[7] | Wattpad[8] |
| Social App features and fan-base |  |  |  |
| Rate and comment |  |  |  |
| Preferences, including genres, ratings, etc. |  |  |  |
| Smart auto-suggest list |  |  |  |
| Notification mute controls | **** | **** | **** |
| Select library integration | **** | **** | **** |
| Direct-connect with authors | **** | **** | **** |
| Publication, sampling, promotion |  |  |  |
| List of sellers, price, availability |  |  |  |

# Conclusion:

Learning process involves having to go through some challenges and difficulties. Our team found no exception to that rule. We encountered a number of hurdles along the way and learned something important with each experience.

One of the first challenges was to figure out how Git handles word documents. Initially there were concerns that Git might not track changes to Word documents, so we created a naming convention to keep track of updated versions. The use of a decentralized version control system was a fairly simple process, once we understood it.

Another interesting episode was a result of market research that led to discovery of existing products with features very similar to our initial proposal. The team came together quickly to discuss the overlaps and a single brainstorming session resulted in a solution to distinguish our proposed product from competition – reinforcing the value of collaboration.

There were some confusions during Deliverable1 exercise that resulted in duplication of effort at times between members. Deliverable2 process went much more smoothly as a result.

The team would like to express their gratitude to an Dr. Ebru Cankaya for her excellence in planning, delivery, and remarkable dedication to learning – Thank You!!

# References:

[1] R. Southard, *FPARP488*, 13-Nov-2003. [Online]. Available:

https://www.umsl.edu/~sauterv/analysis/function\_point/FPARP488.html. [Accessed: 19-Apr-2019].

[2] “Average Application Developer Salary,” *PayScale*. [Online]. Available: https://www.payscale.com/research/US/Job=Application\_Developer/Salary. [Accessed: 19-Apr-2019].

[3] “Salaries,” *Jobs*. [Online]. Available: https://www.indeed.com/salaries/Application-Developer-Salaries. [Accessed: 19-Apr-2019].

[4] “Average Mid-Career User Experience Designer Salary,” *PayScale*. [Online]. Available: https://www.payscale.com/research/US/Job=User\_Experience\_Designer/Salary/acba21da/Mid-Career. [Accessed: 19-Apr-2019].

[5] “Average Experienced Project Manager, Software Development Salary,” *PayScale*. [Online]. Available: https://www.payscale.com/research/US/Job=Project\_Manager,\_Software\_Development/Salary/4189b7d4/Experienced. [Accessed: 19-Apr-2019].

[6] “Average Entry-Level Software Quality Assurance (SQA) Tester Salary,” *PayScale*. [Online]. Available: https://www.payscale.com/research/US/Job=Software\_Quality\_Assurance\_(SQA)\_Tester/Salary/9995b579/Entry-Level. [Accessed: 19-Apr-2019].

[7] Goodreads. (2018). *Meet your next favorite book.* [Online]. Available: https://www.goodreads.com/. [Accessed: 17-Apr-2019].

[8] Wattpad. (2019). *The world’s most-loved social storytelling platform.* [Online] Available at: https://www.wattpad.com/. [Accessed: 15-Apr-2019].

[9] “Total cost of ownership of servers,” *SherWeb*, 11-Jan-2019. [Online]. Available: https://www.sherweb.com/blog/cloud-server/total-cost-of-ownership-of-servers-iaas-vs-on-premise/. [Accessed: 17-Apr-2019].

[10] M. D. Jun, M. Alleven, and M. Dano, “Maintaining an app is critical to its overall success,” *FierceWireless*, 25-May-2012. [Online]. Available: https://www.fiercewireless.com/developer/maintaining-app-critical-to-its-overall-success. [Accessed: 17-Apr-2019].

[11] T. Mackenzie, “App store fees, percentages, and payouts: What developers need to know,” *TechRepublic*. [Online]. Available: https://www.techrepublic.com/blog/software-engineer/app-store-fees-percentages-and-payouts-what-developers-need-to-know/. [Accessed: 17-Apr-2019].