

ENSE 375 – Software Testing and Validation

Online Banking Simulator

Quinn Maloney (200431628)

Brooklyn Coulson (200375222)

Shahzil Siddiqui (200448986)

Table of Contents

[1 Introduction 5](#_Toc43885122)

[2 Design Problem 6](#_Toc43885123)

[2.1 Problem Definition 6](#_Toc43885124)

[2.2 Design Requirements 6](#_Toc43885125)

[2.2.1 Functions 6](#_Toc43885126)

[2.2.2 Objectives 6](#_Toc43885127)

[2.2.3 Constraints 6](#_Toc43885128)

[3 Solution 7](#_Toc43885129)

[3.1 Solution 1 7](#_Toc43885130)

[3.2 Solution 2 7](#_Toc43885131)

[3.3 Final Solution 7](#_Toc43885132)

[3.3.1 Components 7](#_Toc43885133)

[3.3.2 Features 7](#_Toc43885134)

[3.3.3 Environmental, Societal, Safety, and Economic Considerations 7](#_Toc43885135)

[3.3.5 Test cases and results 7](#_Toc43885136)

[3.3.4 Limitations 7](#_Toc43885136)

[4 Team Work 8](#_Toc43885137)

[4.1 Meeting 1 8](#_Toc43885138)

[4.2 Meeting 2 8](#_Toc43885139)

[4.3 Meeting 3 8](#_Toc43885140)

[4.4 Meeting 4 8](#_Toc43885141)

[5 Project Management 9](#_Toc43885142)

[6 Conclusion and Future Work 10](#_Toc43885143)

[7 References 11](#_Toc43885144)

[8 Appendix 12](#_Toc43885145)

* Proof read the text for typing and grammar mistakes.
* Follow the IEEE Bibliography style for the references by selecting "References/ Citations & Bibliography/ Style".

List of Figures

List of Tables

# Introduction

* Give a brief description of the design and a summary of the relevant background information related to the topic. Give a rationale about what is needed and why.
* Give the reader an overview of what is in the next sections.
* Do not put any detailed results of your work here.

# Design Problem

This section has the following two subsections:

## Problem Definition

We strive to create an online banking simulator to assist those who are new to using online banking and would like to have a better understanding of how to use this new technology effectively and efficiently. This simulator will be using preset accounts for the user to test online banking with to better understand how they may use it for their own banking needs without the worry of damaging their financials in the process. The motivation for this project stems from the observation that many growing individuals lack education on how one should properly operate and interact with a banking system. Acknowledging the fundamental role banking plays in our daily lives, we find it motivating to develop such a system. Some restrictions may include: deposit restrictions (can’t be below 0, can’t be above X), withdraw restrictions (can’t be 0 or lower, can’t be more than you have), daily withdrawal limits (cannot withdraw more than X in a day determined by account type), login and signup restrictions (length and/or required characters).

## Design Requirements

### Functions

* Withdraw: Takes money out of select account
* New Login: Create login credentials for a user
* Login: Login a user
* Deposit: Adds money to select account
* Etransfer: Send money from your account to someone else within your contact list or to a new contact
  + Add New Contact: Add a new recipient to your contact list
  + Delete Contact: Remove a recipient from your contact list
* Transfer between accounts: Takes funds from one account to add to another
* Pay Bill: select a payee to send money to
  + Add Payee: Add account information for a new payee
  + Remove Payee: Remove payee
* Update Account Info:
  + Update Name: allows to change the username
  + Update Address allows to change address
  + Update Phone Number: allows to change phone number
* View Account Balance: Check the balance of money within a specified account associated with the logged in user’s banking plan

### Objectives

* Be able to simulate creating login credentials (general bank account) [Primary]
* Be able to edit limited account settings (login credentials, address) [Primary]
* Be able to login with preset account (different bank account types for them to be able to play around with) [Primary]
* Withdraw and deposit funds on preset accounts [Primary]
* Transfer money from one account to another [Secondary]
* Simulate e-transfers from one preset account holder to another [Secondary]
* Add a payee and demonstrate how to pay bills electronically [Secondary]
* Demonstrate restrictions when moving monetary funds according to preset account types [Primary]
* View balances within existing accounts within a logged in user [Primary]

### Constraints

* Economic factors
  + Cost
  + Software available
* Regulatory compliance (Security and Access)
  + No real data from any account holders with the organization will be used
  + Ensure login details are correct before granting access to the (dummy) account information
* Reliability
  + It accurately simulates an interaction one would have with a real banking application
  + The software will be able to handle all incorrect input properly
* Ethics
  + No info based on real actors
  + Does not mimic existing banking institutions' design that could become a culprit of fraud.

# Solution

In this section, you will provide an account of some solutions your team brainstormed to implement the project. Some solutions might not have all the desired features, some might not satisfy the constraints, or both. These solutions come up in your mind while you brainstorm ways of implementing all the features while meeting the constraints. Towards, the end you select a solution that you think has all the features and satisfies all the constraints. Remember that an engineering design is iterative in nature!

## Solution 1

A full banking application with an embedded database to represent a fully functional application. In this version we envision to deliver the closest experience to our user with respect to using a banking application without the real-life risks that are closely coupled. We envisioned using an SQL database to hold sets of data where the users can create one of their own accounts. Based on the user’s requirements, this includes a child, student, and an everyday account. The primary reason we chose not to pursue with this solution is with respect to the project timeline and delivery constraints. In addition, managing a database requires for resources and time to be further divided, causing a deviation from the project’s primary goal and objectives, which falls beyond the scope of the project.

A graph paper with a drawing

Description automatically generated

Figure 1: Low fidelity paper prototype for main page, solution 1.

A white sheet with blue writing

Description automatically generated with medium confidence

Figure 2: Low fidelity prototype for withdraw page, solution 1.

## Solution 2

Our second solution is to change from a banking application to make a banking simulation. This allows us to have similar functionality without requiring a database. The application will be able to simulate a user logging in with either pre-made or their own credentials, withdraw or deposit money into accounts, demonstrate restrictions and limits when moving money, view balances, and edit or change the accounts settings. Like solution one, we will still include multiple banking plans to represent the different types of accounts and restrictions a user is likely to face in a real banking application. The difference being we will not have to develop and maintain a database to accompany the banking application.

A graph paper with text on it

Description automatically generated

Figure 3: Low fidelity paper prototype for main page, solution 2.

A whiteboard with blue and red text

Description automatically generated

Figure 4: Low fidelity prototype for withdraw page, solution 2.

## Final Solution

This is the final solution. Explain why it is better than other solutions. You may use a table for comparison purposes. After providing the reason for selecting this solution, detail it below.

The development team decided on three total designs for the solutions, where the first two are illustrated in figures 1, 2, 3 and 4, respectively. The final design considers the project’s scope and customer deliverability, ensuring and maximizing product neutrality. As such, the project development team designed the following final solution.

A screenshot of a computer screen

Description automatically generated

Figure 5: Main user interface page for the final solution.

A screenshot of a computer screen

Description automatically generated

Figure 6: Withdraw page for the final solution.

The development team decided to approach selecting the best solution according to both the qualitative and quantitative methodologies. The selection process begun with outlining the pros and cons for all three proposed solutions. The conceptualized findings were recorded in a table illustrated below.

A table with text and images

Description automatically generated with medium confidence

Figure 7: Pros and Cons for each solution.

Due to the fact that the pros and cons chart, alone, is a superficial qualitative approach, the following decisive decision matrix was created with the attempt to quantity the findings from figure 7 where decisive categories were created and assigned weight. The weightage of the categories helped the development team prioritize the most important aspects of the minimal viable product (MVP) from both the development and user perspectives.

A table with numbers and equations

Description automatically generated

Figure 8: quantitatively grading each solution.

This comparative approach grades the solution according to the categories outlined in the left most column of figure 8, where value to customers and ease of use hold the maximum weightage of 5, and the development teams believes them to be the most important factions given the primary purpose of this simulator. While the final solution scores lower in terms of features as a database is not implemented in it, it totals high in value to customer and out scores the other solutions in Environmental factors and ease of use. Even the lack of a database is a security and safety asset in the sense that entered user information will never be stored on a server and is instead destroyed after the session is closed. Furthermore, the final solution design utilizes neutral colours and presents the simplest graphical user interface (comparatively). Especially where the options in the main page are purposefully centered to immediately catch the target audience focus, drawing attention. Whereas in the other solutions, the user interface is convoluted and offers a complex presentation due to the extra features. In addition, a left-to-right design approach was implemented in the withdraw page as most of the targeted audience in Canada scans content from the left. In the final design, first, the amount available in each account are presented and later the input field prompts the user to enter a value. After which a button specific to the account can be pressed to confirm the withdrawal.

### Components

There are nine total components: FactoryPlanController.java, SimulatorGUI.java, PlanFactory.java, Plans.java, Basic.java, Kids.java, Student.java AccountHolderInfo.java, and Account.java. FactoryPlanController.java is where main is located. It initializes SimulatorGUI.java and calls the first page from SimulatorGUI.java. SimulaotrGUI.java is where all of the user interaction and GUI components are managed. PlanFactory.java creates an instance of Plans.java based on what type of plan the user requests either Basic, Kids, or a Student plan. Plans.java is the interface that is the template for the Basic, Kids, and Student plan. Plans.java contains the method names and parameters so they can be overridden and utilized by the individual plan files. Basic.java, Kids.java, and Student.java all implement the Plans.java file. The Basic.java and Student.java plans are similar. They both have a savings and chequing account associated with them but have different daily limits. Kids.java is different since it only has a savings account. This changes some of the functionality with it compared to the Basic or Student plan. All three of the plans contain the methods of the primary functions of the application including checking account balance, withdraw, deposit, or transfer money in the case of the Basic and Student plan. AccountHolderInfo.java hold the user’s personal information including their name, phone number, username, password, etc. Each plan has one AccountHolderInfo association. Account.java is where the balances for a user’s account is located. The Basic.java and Student.java both have two associated accounts. A savings as well as a chequing account. Kids.java does not have a chequing account; only a savings account.

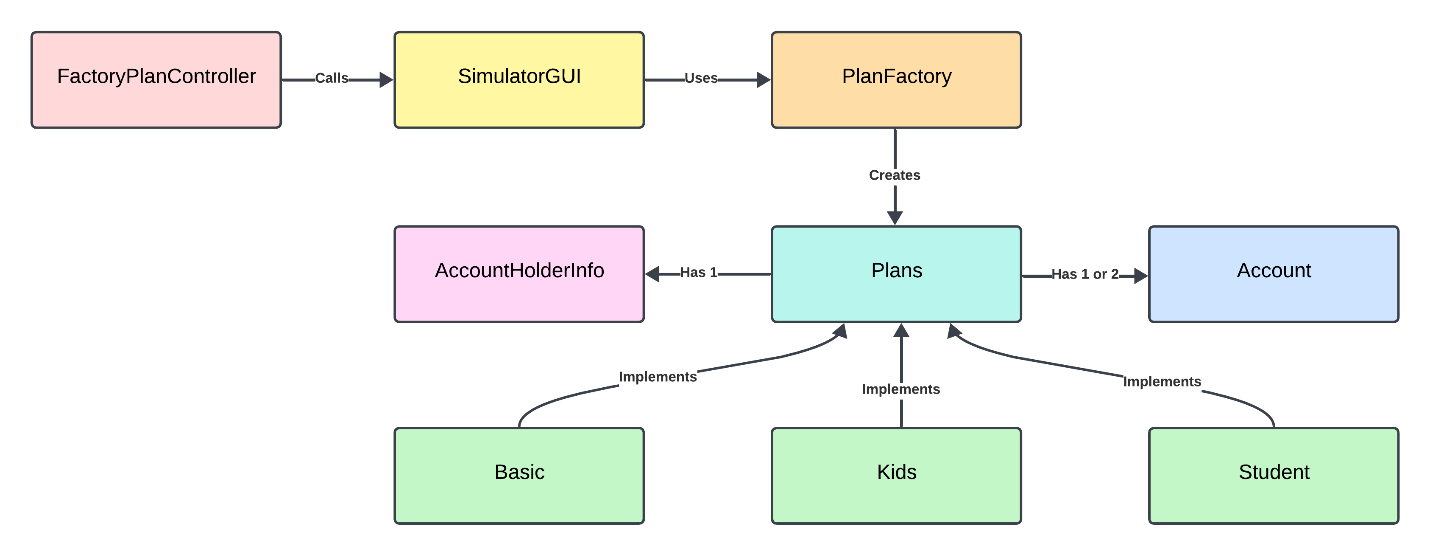


Figure 9: Component Block Diagram

### Features

Online Banking Simulator Features

|  |  |
| --- | --- |
| Feature | Summary |
| Check Account Balance | This functionality allows Basic or Student users to check the balance of their savings and chequing accounts. In the case of a user with a Kids account they can only view a savings account. |
| Deposit Funds | This functionality allows the Basic and Student users to deposit funds into either their savings or chequing account. A user with a Kids account can deposit into their savings account. |
| Withdraw Funds | This functionality allows the Basic and Student users to withdraw funds from either their savings or chequing account. A user with a Kids account can withdraw from their savings account. |
| Transfer Funds | This functionality allows Basic or Student users to transfer funds between their chequing and savings account. This feature is not accessible for users with a Kids account. |
| Daily Transfer Limits | This functionality makes it possible for the user to only withdraw a certain amount of money a day. There is also a limit on the number of transactions that can be made per day. The values of these limits depends on what plan the user has selected. |
| Password Authentication | This functionality is a simple password authentication system that only allows the users to log into their accounts if the provided username and password are the same as the ones located in their account information. |
| Interactive GUI | This functionality provides the user with an interactive graphical interface making navigation clear and easy to understand. All functions are outlines and easy to find. |

Figure 10: Online Banking Simulator Features

### Environmental, Societal, Safety, and Economic Considerations

Explain how your engineering design took into account environmental, societal, economic and other constraints into consideration. It may include how your design has positive contributions to the environment and society? What type of economic decisions you made? How did you make sure that the design is reliable and safe to use?

The design of our solution displays several environmental, social, economical, and safety-oriented considerations. Since we decided to implement less features, we consider this solution to be more environmentally friendly as there will be less resources needed in order to develop, test, and maintain our application. This same argument can be made towards our design being more economical since our application has less features and no longer requires a database. There are less resources needed to operate it, and so organizations will be more inclined to offer it as a free service to their clients. This application has a design that is centered around the user experience, and as such will provide a positive contribution to society.

To maximize user neutrality, the development team has taken into account neutral colours and intuitive design, resonating a positive societal impact in the environment. This application strives to allow users to be able to test out how an online banking application would work without having to commit to a specific banking plan first. By allowing users to test out the differences between accounts, they are able to make more educated decisions in regard to how their financials will be handled.

Lastly, this design takes into consideration how important safety and reliability are to the user since they will be trusting the bank with their financial needs. Our application simulates login, username, and password capabilities that would be similarly featured in a typical banking application. We also understand that privacy is also important, and so no information is remembered or saved after a user exits/ends the simulation.

### Test Cases and results

What test suits did you design to test your prototype? How did you execute the test cases to test the prototype?

When the first version of the product was programmed, the development team began the software testing phase. The first testing suit was composed of unit tests that spanned and covered all branches for each method. As the factory design pattern was used, this meant that unit testing was practiced for all classes that implemented the “Plans” interface. This included the following classes: Basic.java, Student.java, and Kids.java. This is represented in table 2 where for version 1, test cases 1-23 were executed.

A second version of the product was created with the creation and addition of the Tuple class that allows the methods to return two values: a Boolean, and result description. As such Regression Testing was performed for all of the Basic, Student, and Kids classes, where test cases 1-21 were modified to accommodate the return type from Boolean to Tuple. Integration testing was then carried out in using the Code Coverage feature through the IntelliJ framework. The Code Coverage Testing was performed on the FactoryPlanController.java that makes use of the SimulatorGUI.java class to drive the application. The product was then tested against the specifications using the Robust Boundary Value Analysis (BVA).

A few test cases from the BVA resulted in a failure where the expected result did not match with he actual results. This was particularly in the test cases with negative withdrawals and deposits. The Integration testing using Code Coverage had also highlighted some dead code methods. As such a third version of the product was created that handled the discrepancies from the BVA and Code Coverage. Once again, the regression and Integration testing were executed again with the additional test cases 24-28 as shown in table 2. Lastly, BVA, State Transition and Use Case Testing, were executed to test against the specifications.

Table 1 from the appendix demonstrates all the test cases method signatures in detail that are present in the three aforementioned classes. The following table summarized the test suits that were designed to test the prototype for each version of the software:

|  |  |  |
| --- | --- | --- |
| **Software Version** | **Testings** | **Test Suit with Test Cases** |
| Version 1 | Unit | T1 = { t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23} |
| Version 2 | Regression  Integration  BVA | T2 = { t1’, t2’, t3’, t4’, t5’, t6’, t7’, t8’, t9’, t10’, t11’, t12’, t13’, t14’, t15’, t16’, t17’, t18’, t19’, t20’, t21’, t22, t23} |
| Version 3 | Regression  Integration  State Transition  Use Case  BVA | T3 = { t1’, t2’, t3’, t4’, t5’, t6’, t7’, t8’, t9’, t10’, t11’, t12’, t13’, t14’, t15’, t16’, t17’, t18’, t19’, t20’, t21’, t24, t25, t26, t27, t28} |

Table 2: Shows all the test suits with all the test cases.

### Limitations

Every product has some limitations, and so is the case with your design product. Highlight some of the limitations of your solution here.

There were several limitations regarding our final solution. Due to the time constraint associated with this project, we had to limit some of the features and functionality available to our users including the ability to roll the date forward, saving your simulation so that you can return at a later date without restarting the entire simulation process, and having the ability to pay bills or send e-transfers to other account holders. We did not have time to implement any of these options as we would have had to implement proper testing methodologies and strategies for all these additional features. Another limitation we considered was the required programming language component for this project. It was outlined that we were to use Java, and so this provided some limitations regarding the libraries needed to make a functioning GUI, as well as how we would need to refactor some design choices to accommodate the capabilities of these libraries.

# Team Work

Since this is a group project, you must have a fair distribution of tasks among yourselves. To this end, you must hold meetings to discuss the distribution of tasks and to keep a track of the project progress.

## Meeting 1

## **Time:** May 23, 2024 9:30 pm to 11:45 pm

**Agenda:** Discuss and decide on Project Topic, Create link for project on github, Email Prof GitHub link of project

|  |  |  |  |
| --- | --- | --- | --- |
| **Team Member** | **Previous Task** | **Completion State** | **Next Task** |
| Brooklyn Coulson | Brainstorm project ideas | 100% | Project Idea  Email prof the GitHub link of the project |
| Quinn Maloney | Brainstorm project ideas | 100% | Potential Restrictions |
| Shahzil Siddiqui | Brainstorm project ideas | 100% | Project Motivation  Create Project GitHub |

## Meeting 2

## **Time:** May 29, 2024 4:00 pm to 5:30 pm

**Agenda:** Discuss design requirements (functions, objectives, constraints)

|  |  |  |  |
| --- | --- | --- | --- |
| **Team Member** | **Previous Task** | **Completion State** | **Next Task** |
| Brooklyn Coulson | Project Idea  Email prof the GitHub link of the project | 100%  100% | Design Requirements |
| Quinn Maloney | Potential Restrictions | 100% | Design Requirements  Update the Report template and push to GitHub |
| Shahzil Siddiqui | Project Motivation  Create Project GitHub | 100%  100% | Design Requirements  Clarify some questions with the prof |

## Meeting 3

**Time:** June 15, 2024 10:32 am to 11:30 am

**Agenda:** Discuss Project Solutions 1, 2, 3

|  |  |  |  |
| --- | --- | --- | --- |
| **Team Member** | **Previous Task** | **Completion State** | **Next Task** |
| Brooklyn Coulson | Design Requirements | 100% | Final Solution |
| Quinn Maloney | Design Requirements  Update the Report template and push to GitHub | 100%  100% | Second Solution |
| Shahzil Siddiqui | Design Requirements  Clarify some questions with the prof | 100%  100% | First Solution |

## Meeting 4

**Time:** June 20, 2024 5:05 pm

**Agenda:** Finalize Solutions

|  |  |  |  |
| --- | --- | --- | --- |
| **Team Member** | **Previous Task** | **Completion State** | **Next Task** |
| Brooklyn Coulson | Final Solution | 100% | Finalize Solutions |
| Quinn Maloney | Second Solution | 100% | Finalize Solutions |
| Shahzil Siddiqui | First Solution | 100% | Finalize Solutions |

# Project Management

Provide a Gantt chart showing the progress of your work here. Mention all the tasks along with their predecessors. Provide the slack time of each task and identify the critical path.

# Conclusion and Future Work

* A summary of what you achieved. Mention all the design functions and objectives that you achieved while satisfying the constrains?
* While keeping the limitations of your solution, provide recommendations for future design improvements.

# References

* Use the IEEE reference style.
* Do not put any reference if it is not cited in the text.

# Appendix

If you want to provide an additional information, use this appendix.

|  |  |
| --- | --- |
| **Test Case** | **Test Method Signature** |
| t1, t1’ | test\_Constructor\_AccountHolderInfo |
| t2, t2’ | test\_Constructor\_Chequing\_Balance |
| t3, t3’ | test\_Constructor\_Chequing\_AccNum |
| t4, t4’ | test\_Constructor\_Chequing\_AccType |
| t5, t5’ | test\_Constructor\_publicVars\_WithdrawCount |
| t6, t6’ | test\_Constructor\_publicVars\_TransActionCount |
| t7, t7’ | test\_Withdraw\_Success |
| t8, t8’ | test\_Withdraw\_Failure\_notEnoughBalance |
| t9, t9’ | test\_Withdraw\_Failure\_exceedsWithdrawLimit |
| t10, t10’ | test\_Withdraw\_Failure\_exceedsTotalWithdrawLimit |
| t11, t11’ | test\_WithdrawBoundary\_Success\_DailyTransactionCount |
| t12, t12’ | test\_Withdraw\_Failure\_ExceedsDailyTransactionCount |
| t13, t13’ | test\_Deposit\_Success |
| t14, t14’ | test\_DepositBoundary\_Success\_ |
| t15, t15’ | test\_Deposit\_Failure\_ExceedsTransctionLimit |
| t16, t16’ | test\_ViewBalance\_Success\_Chequing |
| t17, t17’ | test\_ViewBalance\_Success\_Savings |
| t18, t18’ | test\_TransferFunds\_Success |
| t19, t19’ | test\_TransferFundsBoundary\_Success |
| t20, t20’ | test\_TransferFunds\_Failure\_ExceedsTransactionLimit |
| t21, t21’ | test\_TransferFunds\_Failure\_exceedsFunds |
| t22 | test\_ResetCounter\_Success\_withdrawCount |
| t23 | test\_ResetCounter\_Success\_transactionCount |
| t24 | test\_Withdraw\_Failure\_NegativedailyWithdrawCount |
| t25 | test\_Withdraw\_Failure\_NegativeWithdrawal |
| t26 | test\_Withdraw\_Failure\_NegativeDailyTransactionCount |
| t27 | test\_Deposit\_Failure\_Negativedeposit |
| t28 | test\_Deposit\_Failure\_NegativeDailyTransactionCount |

Table 1: this table contains all the test cases.