BGO Financial

Aaron Borja SID: 200482770
Yuval Glozman SID: 200482531
Zana Osman SID: 200489300

July 31st, 2025

Software Testing and Validation (ENSE 375)

University of Regina | Go far, together.

Agenda

- Introduction
- Problem Definition
- Design Requirements
- Solutions
- Testing and Demonstration
- Project Management
- Conclusion and Future Scope

Introduction

- Testing is very important to make sure programs work and are reliable
- Banking systems must be tested thoroughly.

- Console based bank management application
- Features: account creation, deposit/withdraw, transfer funds, checking account balance
- Practiced testing methodologies and strategies: TDD, Path Testing, Integration, and many more…

Problem Definition

With the ever changing economic landscape, the safe and reliable management of your wealth has never been harder. Without a solution that allows users to manage their wealth and keep it secure, it puts stress and worry on them. This can lead to financial instability, unexpected debt, and difficulty achieving long-term financial goals. Our system will allow users to safely store, deposit, and withdraw and review their financial resources in an easy and reliable menu without too much complexity.

Design Requirements (Functions)

Functions

- Create new bank accounts with unique identifiers
- Deposit funds into user accounts
- Withdraw funds from user accounts
- Display account balances
- Transfer funds between accounts

Design Requirements (Objectives)

Objectives

- App reliability
- Implementing local storage
- User friendly with easy directions
- Easily testable and has no bugs

Design Requirements (Constraints)

Constraints (Practical)

- Budget Constraints
- Team Size
- Time constraints - 7 week project
- Hardware & Software Constraints
- Program must be created in Java, using Junit to test
- Developed using the V-Model

Design Requirements (Constraints)

Constraints

- Reliability - Utilization of a Database to ensure concurrent data and testing to ensure reliability of program
- Societal Impact - Our application would allow its users to better manage their finances and store them, keep their finances secure and help them reach their financial goals
- Economic Factors - Our application is free and is cheap to run/manage
- Regulatory Compliance - We ensured to use a login system to keep users data secure, as we continue with the project we will continue to enhance security through password hashing

Solution 1

For our first solution, our idea was to stray towards making a GUI for our application using basic HTML, CSS and Javascript and or other Java frontend frameworks, while also focusing on implementing the main functionalities of our application. This includes:

- Ability to create different types of accounts.
- Depositing funds into accounts.
- Withdrawing funds from made accounts.
- Displaying account balances.
- Ability to delete own account.

With this solution, we would implement any of the big name databases such as MongoDB, or MySQL. When it comes to selecting a testing framework for this solution, we wanted to keep it minimal and stick with JUNit.

After considering all of these features in our application, we found that this solution would make testing and development more complex and severely increase our development time, given our team only has a month to code and develop the main functionalities of our application. This solution would be prioritized over other alternatives if development time was not a primary constraint.

Solution 2

Our second solution would have the main functionalities of our first solution. However, this time instead of utilizing front-end languages/frameworks like HTML, CSS and Javascript for example, we would strictly limit our application to run using a console/terminal based GUI.

This solution would give us more time to focus on testing and functionality of the program rather than it's visual appeal or ease of use when it comes to the graphical interface.

Solutions 3

For our final solution, we ended up leaning towards Solution 2 as an inspiration, which is a CLI-based architecture with key features like deposit/withdraw, and transfer funds feature. This allowed us to maximize the testability in our project, while also being able to deliver reliable business logic within the short timeframe we were given.

With this solution, we were able to reduce the dependency on the front-end frameworks, which allowed us to spend more time with the needed code paths and maintaining the components in the codebase. We used SQLite which allowed us to store data directly in a single, self-contained file on the local file system. This eliminated the need of a server which made the application easier to work with and test.

Solutions Comparison

Solution	Testability	Implementation Effort	Feature Coverage	Total Score
Solution 1	5	2	3	10
Solution 2	6	6	6	18
Solution 3	8	8	9	25

1 – 10 scale, where 10 is best and 1 is worst

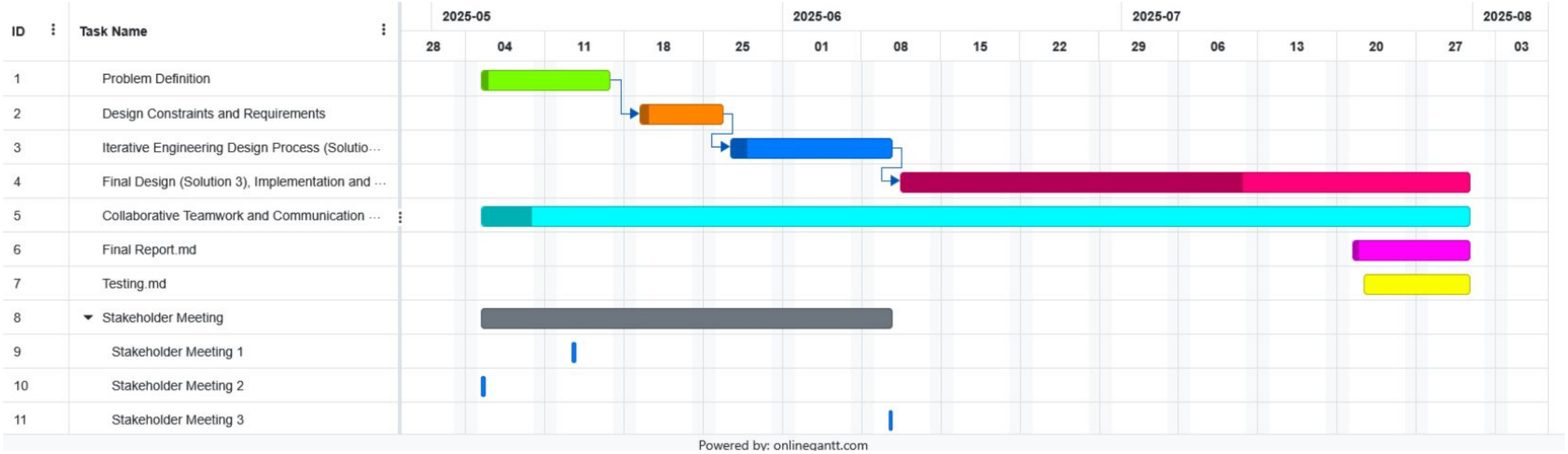
Solution 1: Full scale GUI using HTML,CSS, and JS would require more time and be much more complex to test. This makes its testability relatively lower to the other solutions and its implementation effort a lot lower as it requires a lot more effort to make and properly develop. Due to the increased time working on the front-end and testing, we wouldn't be able to implement as many features therefore the feature coverage is also low

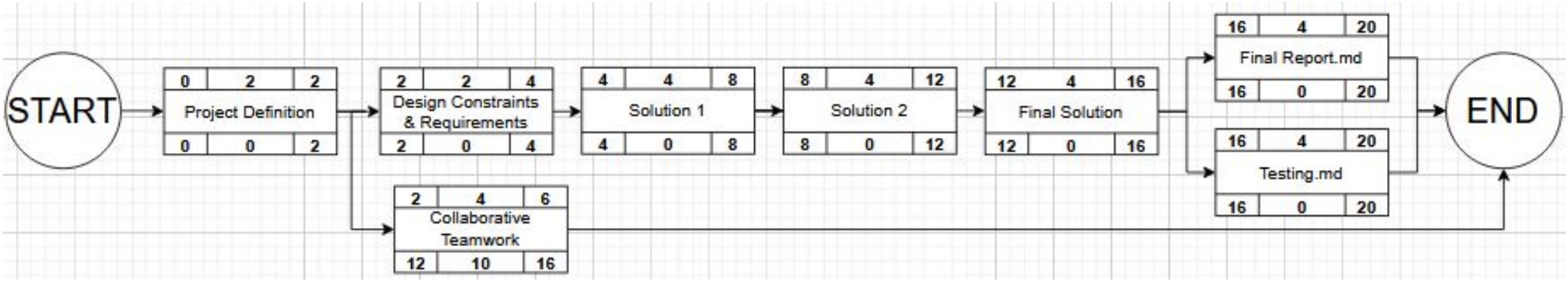
Solution 2: Moving to a Command/Terminal Line interface as compared to a full scale GUI using HTML,CSS, and JS would give us more time to focus on testing and would make testing a lot easier as now less integration testing would be needed between all the front-end components. The amount of effort needed to implement this solution would also go down since we now are able to code in just Java and some SQL. Finally the feature coverage also goes up as we are able to spend less time on the front-end and more on features and testing those features

Solution 3: Our final solution has a higher testability, implementation effort and feature coverage as we implemented more functions (transferFunds), we also moved to use SQLite which we were exposed to in the lab and knew how to use, therefore it made our testability a lot easier and implementation as we didn't have to waste to figure out how to utilize another DB system in our application.

Testing and Demonstration

Project Management




START
0 2 2
Project Definition
0 0 2
2 2 4
Design Constraints & Requirements
2 0 4
4 4 8
Solution 1
4 0 8
8 4 12
Solution 2
8 0 12
12 4 16
Final Solution
12 0 16
2 4 6
Collaborative Teamwork
12 10 16
16 4 20
Final Report.md
16 0 20
16 4 20
Testing.md
16 0 20
END

Conclusion and Future Work

Objectives Completed

- Implemented, designed, and tested CLI based banking management application
- Uses local storage, simple and user-friendly interface, and easily testable.
- Supports basic bank operations

Testing Conclusion

- Utilized multiple test methodologies to ensure all of our bank operations operated as intended
- Used all testing methodologies listed in the project description
- Passed all tests

Future Work

- Transition to a proper Front-end system
- Add signup capability for user
- Add new accounts and more advanced banking features
- Move to a server/cloud based DB to allow access to banking info from multiple devices
- More security through password hashing