LOANEASE

BY:

Rayansh Chowatia - 200467962

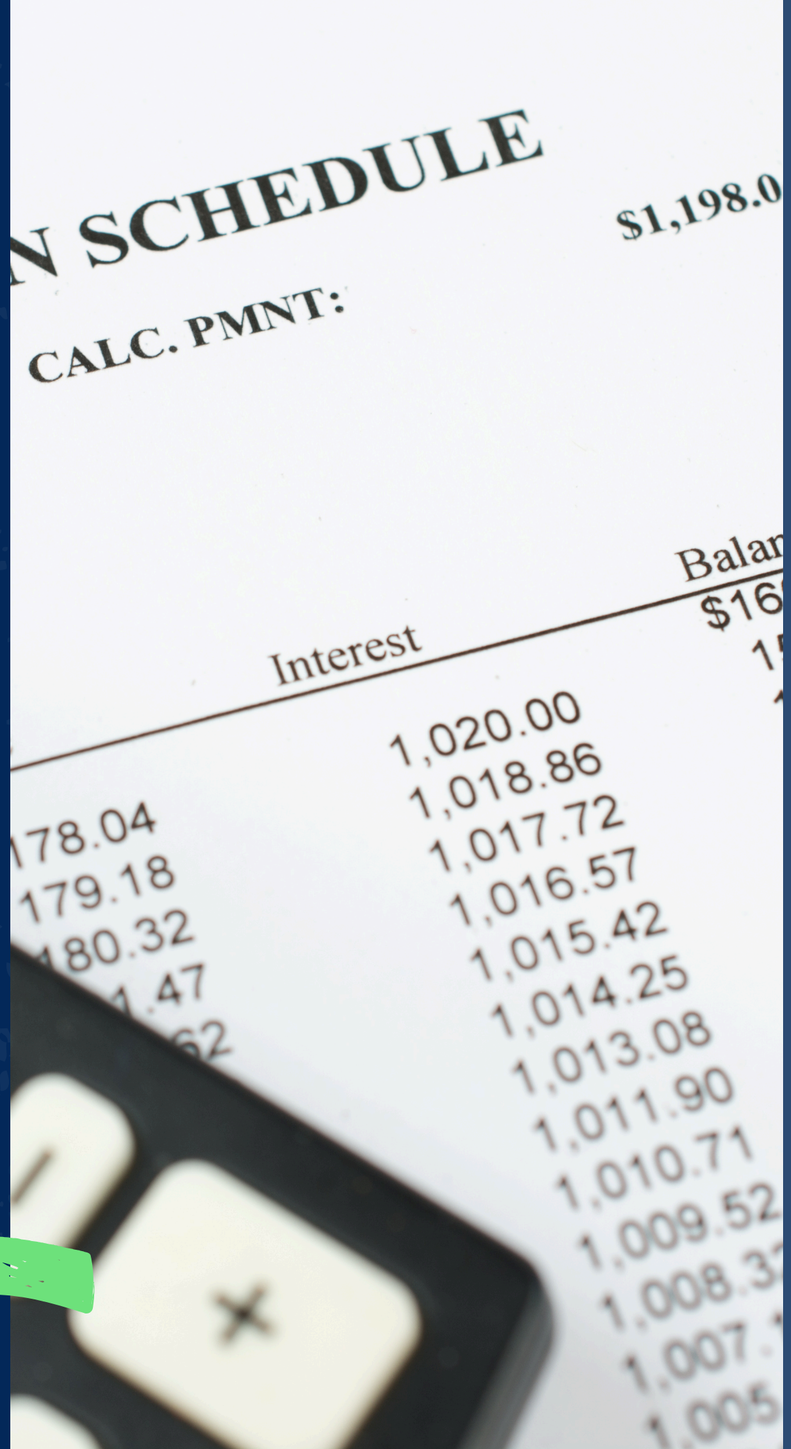
Jeet Patel - 200479073

Jay Patel - 200468891

AGENDA OVERVIEW

- INTRODUCTION
- PROBLEM DEFINITION
- DESIGN REQUIREMENTS
- SOLUTIONS
- TESTING AND DEMONSTRATION
- PROJECT MANAGEMENT

CONCLUSION AND FUTURE SCOPE



INTRODUCTION

LOANEASE IS A USER-FRIENDLY LOAN AMORTIZATION SIMULATOR DESIGNED TO DEMYSTIFY THE TRUE COST OF BORROWING. IT LETS USERS INPUT LOAN PARAMETERS, EXPLORE WHAT-IF SCENARIOS (SUCH AS EXTRA PAYMENTS OR RATE SHOCKS), AND INSTANTLY VIEW DETAILED REPAYMENT SCHEDULES AND VISUALISATIONS.

PROBLEM DEFENITION

LoanEase is a user-friendly loan amortization simulator designed to demystify the true cost of borrowing. It lets users input loan parameters, explore what-if scenarios (such as extra payments or rate shocks), and instantly view detailed repayment schedules and visualizations. With LoanEase, users gain a deeper understanding of their financial commitments, helping them make more informed decisions. Whether you're considering taking on a new loan or managing existing debt, LoanEase offers the tools needed to plan effectively and minimize unnecessary costs.

DESIGN REQUIREMENTS OVERVIEW

This section outlines the essential **functions and objectives** for LoanEase, along with the constraints that guide its development process and ensure usability.

FUNCTIONS AND OBJECTIVES

- User-friendly interface for easy navigation
- Accurate loan calculations based on user input
- Real-time feedback and simulation results

CONSTRAINTS

- Limited budget for development resources
- Compatibility with multiple operating systems
- Performance optimization for smooth user experience

SOLUTIONS

WEB-BASED PROTOTYPE

- Built using HTML, CSS, and JavaScript (index.html, app.js)
- Allowed users to input loan parameters and display an amortization table
- Purpose: Validate loan calculations in a browser before adding complexity
- Initial TDD with Jasmine; manual testing for inputs and edge cases (e.g., 0% rate)
- ✅ Quick to develop (~3 days) and accessible via any browser
- ❌ Lacked path/data flow testing due to monolithic structure
- ❌ No integration or state transition testing; relied on manual checks
- Not selected due to limited testability and scalability for course-level testing standards

MOBILE BASED APP

- Built using Flutter (main.dart) for Android and iOS, with data stored locally using SQLite.
- Used the fl_chart package to generate simple line charts for amortization schedules.
- Independent codebase exploring mobile-first accessibility beyond web users.
- Testing Focus: TDD with test package, widget tests for UI components, integration tests for local storage, and basic boundary value & equivalence class testing.
- Strengths:
  - Native mobile experience with offline support.
  - TDD and widget testing detected UI and logic issues early.
  - Integration testing with SQLite ensured reliable data persistence.
- Weaknesses:
  - Limited cross-platform testing resources constrained state transition & use case testing.
  - No decision table testing due to simple logic.
  - App store deployment added testing overhead.
- Reason Not Final: Testing complexity and incomplete coverage of required techniques led to a shift toward a CLI-based solution.

FINAL SOLUTION – MODULAR CLI ARCHITECTURE

WE SELECTED THE CLI-BASED SOLUTION FOR ITS SUPERIOR TESTABILITY, MODULAR DESIGN, AND ALIGNMENT WITH COURSE TESTING GOALS. IT SUPPORTS COMPREHENSIVE TECHNIQUES LIKE PATH, DATA FLOW, BOUNDARY, EQUIVALENCE, AND INTEGRATION TESTING, ACHIEVING >80% TEST COVERAGE USING JUNIT.

THE SOLUTION IS COMPOSED OF INDEPENDENT MODULES:

- LOANCLI FOR USER INPUT,
- FORMULAENGINE FOR FINANCIAL LOGIC,
- AMORTIZATIONSERVICE FOR SCHEDULE GENERATION,
- EXPORTUTIL FOR CSV/PDF OUTPUT,
- VALIDATOR FOR INPUT CHECKS, AND
- A LOCAL SQLITE PERSISTENCE LAYER.

WHILE IT LACKS VISUALS, THE CLI ENSURES RELIABLE TESTING, MAINTAINABILITY, AND PERFORMANCE. WE'LL WALK THROUGH THE COMPLETE CODE AND TEST REPORTS DURING THE GITHUB DEMO.

COMPARISON OF SOLUTIONS

EACH PROTOTYPE WAS ASSESSED USING KEY SOFTWARE TESTING CRITERIA TO DETERMINE THE MOST TESTABLE AND MAINTAINABLE SOLUTION:

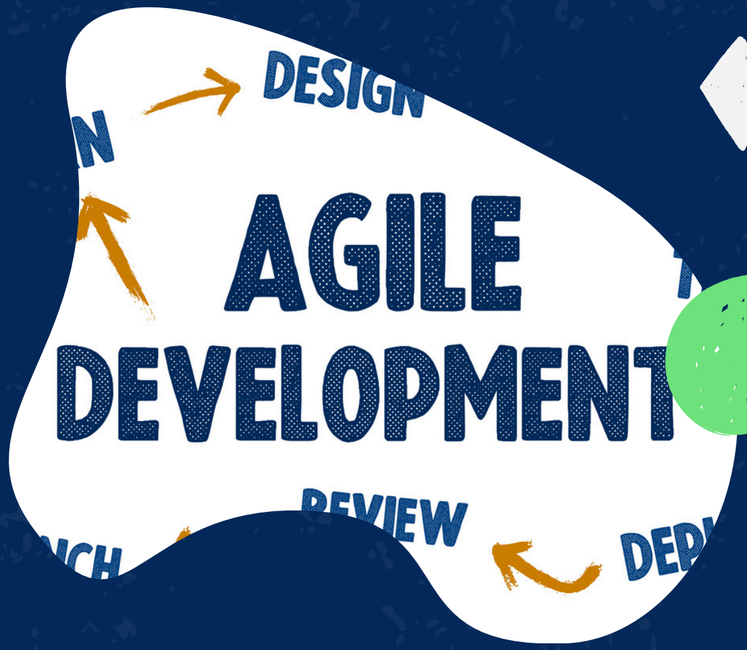
- ✅ TESTABILITY: ONLY THE CLI SOLUTION SUPPORTED FULL TDD, PATH, DATA FLOW, AND INTEGRATION TESTING.
- ✅ COVERAGE: CLI ACHIEVED >80% TEST COVERAGE, SURPASSING THE MOBILE (~60%) AND WEB (<50%) IMPLEMENTATIONS.
- ✅ MAINTAINABILITY: THE MODULAR JAVA CLI STRUCTURE ALLOWED ISOLATED TESTING OF COMPONENTS, UNLIKE THE MONOLITHIC DESIGN OF THE OTHER TWO.
- ✅ TESTING TECHNIQUES: ONLY THE CLI SOLUTION SUPPORTED COMPREHENSIVE TECHNIQUES LIKE BOUNDARY VALUE, EQUIVALENCE CLASS, DECISION TABLE, STATE TRANSITION, AND USE CASE TESTING.
- ✅ FINAL DECISION: THE CLI-BASED ARCHITECTURE WAS CHOSEN FOR ITS ALIGNMENT WITH COURSE TESTING OBJECTIVES AND SUPERIOR SCALABILITY.

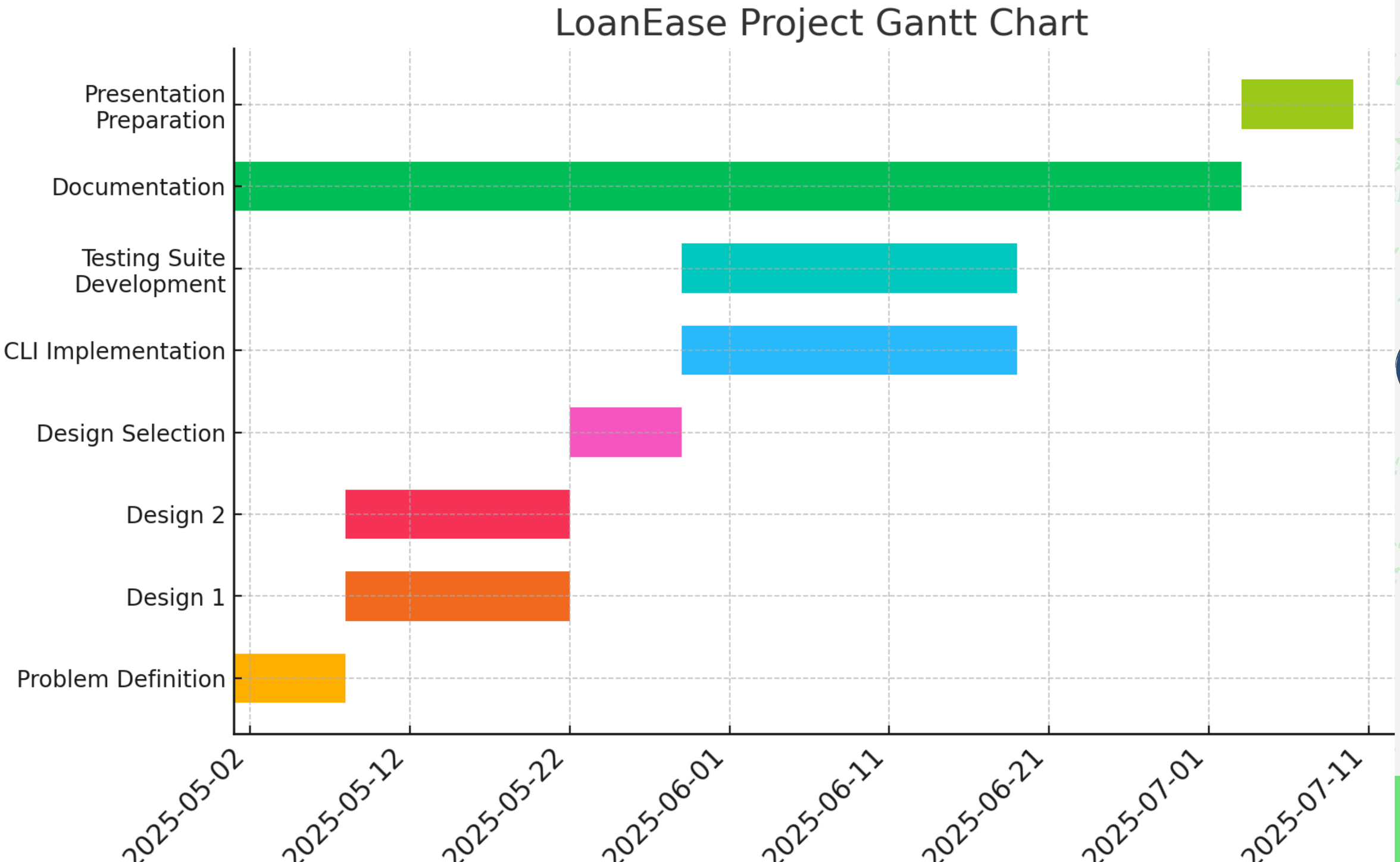
TESTING AND DEMONSTRATION OVERVIEW

- Our final CLI-based solution underwent comprehensive testing aligned with all required techniques.
- JUnit was used with a TDD approach from the start, ensuring consistent test coverage and early bug detection.
- Each testing technique (e.g., path, data flow, boundary value, equivalence class, decision tables, etc.) was applied to appropriate components (e.g., LoanCLI, ExportUtil).
- Test cases were written to satisfy each technique’s requirement, with >80% coverage verified via JaCoCo.
- Full test results, workflows, and working demonstrations will be shown live via GitHub.

TEAM MANAGEMENT

To ensure structured progress and timely delivery, we developed a detailed Gantt chart outlining each project task, its duration, dependencies, and critical path. Tasks ranged from problem definition and multiple design phases to implementation, testing, and final documentation. This chart helped track progress, manage slack, and allocate team efforts effectively.




LoanEase Project Gantt Chart
Presentation Preparation
Documentation
Testing Suite Development
CLI Implementation
Design Selection
Design 2
Design 1
Problem Definition
2025-05-02
2025-05-12
2025-05-22
2025-06-01
2025-06-11
2025-06-21
2025-07-01
2025-07-11

Conclusion

Our project explored three alternative designs—a web-based POC, a mobile Flutter app, and a final CLI-based system. Through iterative design, modular development, and diverse testing strategies, we selected the CLI solution for its testability, simplicity, and compatibility with TDD workflows.

Future Work

- Extend CLI with JSON config file support and export options.
- Build a lightweight web frontend using the same core logic.
- Add automated testing for edge-case financial inputs.
- Explore CI/CD integration for faster iteration and test feedback.

THANKS

SUCCESSFUL COMPLETION OF PROJECT TOOK PLACE UNDER THE GUIDANCE OF

DR YOGESH SHARMA