Project Report on Book Bridge

Course Name: Project Management (PMG 4101)

Section: A

Submitted to:

Md. Muhyminul Haque

Lecturer, Department of CSE, United International University

Submitted by:

Group 6

Name	**ID**
Abu Sayed Rabbi	011221327
Foyez Ahammed Naeem	011222011
Mahmudul Mashrafé	0112230115
Jubair Ahmed	011222136
Shadhin Nandi	0112230604

Date: October 21, 2025

Abstract

BookBridge is all about creating a friendly community where sharing books feels simple, fair, and something everyone can enjoy. It's a safe space where members can add books to the collection and, in return, borrow from others. The idea is straightforward: give a book before you borrow one, which keeps everything balanced and encourages responsibility. BookBridge makes it easy for people to swap books, reduce waste, and enjoy reading without spending too much. It's a simple way to share what you love and access new stories without breaking the bank.

BookBridge goes beyond swapping books. It has features like checking in members, keeping tabs on what everyone's sharing, and making sure exchanges go smoothly and in a friendly way. Most importantly, it's about bringing readers together and building a real sense of community.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Description

Imagine a platform called **BookBridge**, which is like a cozy, secure place online where students, book lovers, and avid readers can come together to swap books. The goal is to make exchanging books easy and fun while building a community that loves reading. You can contribute books, request ones you want, and use a simple credit system to manage exchanges. Plus, there are local Literal Clubs, honest reviews from users, and strict checks to keep everything safe and trustworthy. Unlike casual or messy ways of sharing books, BookBridge keeps things organized and easy to track. It's designed to be inviting and scalable, starting on college campuses and expanding to wider cities, even across the whole country. This way, unused books can find eager new readers instead of gathering dust.

1.2 Motivation Behind the Project

BookBridge was born out of a need to fix the messy, inefficient way people usually share books. Usually, book swaps happen through social media or just friends, which often means missed chances, unused books sitting around, trust issues, and deals falling through. We wanted to create something better. Here's what motivates us:

- **Access Issues:** Lots of students and readers find it hard to get the books they need, especially if they're pricey, rare, or academic.

- **Building a Community:** There's a real need for organized, trustworthy reading groups that do more than casual swaps.
- **Sustainable Culture:** We want to promote a circular economy by cutting down on wasted books and encouraging sharing.
- **Tech Opportunities:** Modern digital platforms give us a chance to make book swapping easier, gamify the process, and get more people involved.
- **Giving People Rewards:** We're working on a credit system that rewards those who contribute, encouraging everyone to participate actively.

All these factors shaped how BookBridge was built and focused on verified users, fair access, reliable inventory tracking, strong moderation, and rewards to turn informal swaps into a meaningful, strategic system.

1.3 Stakeholders

unning a successful book-sharing platform really depends on a bunch of different people with their roles and interests. Here's a quick rundown:

- **General Users:** These are the folks who want to contribute books, borrow some, or just browse what's available. Their activity keeps the platform alive and helps it grow.
- **Litera Club Members:** People who are part of local reading groups. They add books to their club's collection, go to events, and help build a sense of community.
- **Litera Club Managers:** The organizers who handle memberships, coordinate activities, manage the book inventory, and keep everything running smoothly.
- **BookBridge Admins:** The folks behind the scenes who oversee the platform, keeping an eye on content, managing users, and making sure policies are followed.
- **Educational Institutions:** Schools or colleges that could partner up to help expand the platform, connect with students, and add credibility.
- **Local Bookstores and Libraries:** External partners who help out with logistics, like drop-off or pick-up points, and might even team up on special projects.

- **Technical Development Team:** The engineers, designers, testers, and project managers working behind the scenes to build, update, and improve BookBridge.

All these different stakeholders have specific roles and responsibilities that are reflected in the system. Their feedback and needs directly shape how the platform evolves over time.

1.4 Vision and Scope Documents

1.4.1 Vision Document

Our vision is to build strong local communities by creating a friendly and organized space where people can share books and connect, learn from each other, and access knowledge.

1.4.2 Scope Document

In-Scope

1. **Sign-up and Verification**
    - Students and readers can sign up quickly and easily, with simple steps in place to ensure everyone's identity is verified.
    - Admins review new accounts personally to confirm authenticity and maintain a trustworthy community.
2. **Sharing and Requesting Books**
    - Users have the ability to add their own books or make requests for books shared by others.
    - Moderators review each new contribution to ensure quality and appropriateness.
3. **Book Wallet with Credits**
    - Users earn credits by sharing books and can use those credits to borrow books from fellow members.
    - Admins monitor credit exchanges regularly to keep everything fair and transparent.

4. **Litera Clubs**

   - Members can join local book clubs, stay updated with club news, chat with other members, and share reading lists.
   - Club managers assist discussions and organize activities to keep the club lively and organized.

5. **Discovering Books with Search and Filters**

   - Users can browse the entire catalog, applying filters such as genre, location, or associated clubs to find specific titles easily.
   - The admin team updates and manages the main book database to keep it current and well-organized.

6. **Reviews and Star Ratings**

   - Members can leave reviews, rate books, and share feedback about contributors or shared titles.
   - Moderators oversee reviews to prevent misuse, spam, or inappropriate content, intervening when necessary.

7. **Personalized Dashboards by Role**

   - Different dashboards are personalized for regular users, club managers, and administrators, emphasizing relevant information for each role.

8. **Real-Time Notifications**

   - Users receive instant alerts for their requests, approvals, or other activities happening on the platform.
   - Admins can efficiently send announcements and important updates to all users.

9. **Keeping User Data Safe**

   - Your personal information is protected with strict privacy controls and encryption.
   - Admins ensure compliance with the latest security standards to safeguard all user data.

Out-of-Scope

1. Regional scalability with distributed databases for multi-city and multi-organization support.
2. Continuous feedback-based improvements and advanced analytics for user behavior.
3. Partnerships and integrations with bookstores, libraries, and schools.
4. Enhanced logistics for book delivery and tracking beyond manual exchanges.
5. AI-based recommendations for book suggestions and reading clubs.

Chapter 2

Methodology

2.1 What is Software Development Methodology?

A software methodology is a structure that defines the procedures, techniques, and principles for designing software systems. It offers a systematic method of software development, which leads the team through each stage of the project lifecycle, including planning and requirements analysis, design, implementation, testing, deployment, and maintenance. There are several software methodologies, each with its own set of principles and practices. Some common examples include the Waterfall model, Agile methodologies (such as Scrum), the Spiral model, and Extreme Programming (XP).

The method used in our project to achieve the goals:

- Spiral Methodology
- Rational Unified Process
- Prototype Methodology
- Agile Methodology
- Scrum Development Methodology
- Extreme Programming Methodology
- Waterfall Model

2.2 Spiral Methodology

2.2.1 What is Spiral Methodology?

Spiral methodology is an iterative, risk-driven approach in project development, often referred to as the Boehm Spiral Model. It combines the structured elements of the waterfall model with the flexibility and adaptability of prototyping. Spiral method is highly suitable for projects with complex requirements, rapidly changing technology and high level of uncertainty.

2.2.2 Spiral Methodology Design

In the spiral model, development is progressed as a series of cycles, each representing a spiral loop. Each loop involves four key phases.

1. Planning: Objectives, risks, deliverables are defined for the next iteration. Evaluating all the stakeholders' success factors.
2. Risk Analysis: Identifying potential risks and determining mitigation strategies.
3. Development: Developing and testing a prototype. Incrementing the prototype based on identified risks.
4. Evaluation: Reviewing progress, gathering feedback and determining what next steps are.

2.2.3 Spiral Model Terminologies

1. Iteration: A complete cycle of the spiral procedure. An iteration starts from planning, resulting in a working prototype.
2. Prototype: A simplified version of the software used for reviewing, testing and getting feedback.
3. Risk: Any uncertain event that is considered a negative impact for the project.
4. Risk Mitigation: Alternate ways to do a task to reduce the likelihood of uncertain events or risks.
5. Baseline: The stable version of the software after finishing an iteration.

2.2.4 Spiral Model Procedure

This section describes the working procedure of the spiral model.

1. Identify Objectives: Define the overall project goals and deliverables.
2. Identify and analyze the risks: Evaluate potential risks associated with technical feasibility, market acceptance, and other factors.
3. Mitigate Risks: Based on the risk analysis, find alternate ways to minimize the negative impacts.
4. Development: Implement the chosen approach to create a prototype or product increment.
5. Evaluation: Obtain feedback from stakeholders and evaluate progress against objectives.
6. Review and Plan Next Iteration: Based on evaluation, decide whether to proceed to the next iteration, refine requirements, or mitigate identified risks.

2.2.5 Advantages

1. Flexibility and Adaptability: Iterative approach allows for changes in requirements and technology to be accommodated easily.
2. Early Risk Identification: Emphasis on risk identification and mitigation throughout the development process increases project success rates.
3. Continuous Feedback: Regular evaluation and feedback loops ensure stakeholder satisfaction and product alignment with needs.
4. Quality Assurance: Testing is integrated into each iteration, leading to a more robust final product.

2.2.6 Disadvantages

1. Complexity: Managing multiple iterations and risk assessments can be complex, requiring experienced project management skills.

2. Unpredictable Costs and Timeframes: Iterative nature makes precise cost and schedule estimation challenging.
3. Heavy Documentation: Detailed risk analysis and planning documentation can be time-consuming to create and maintain.

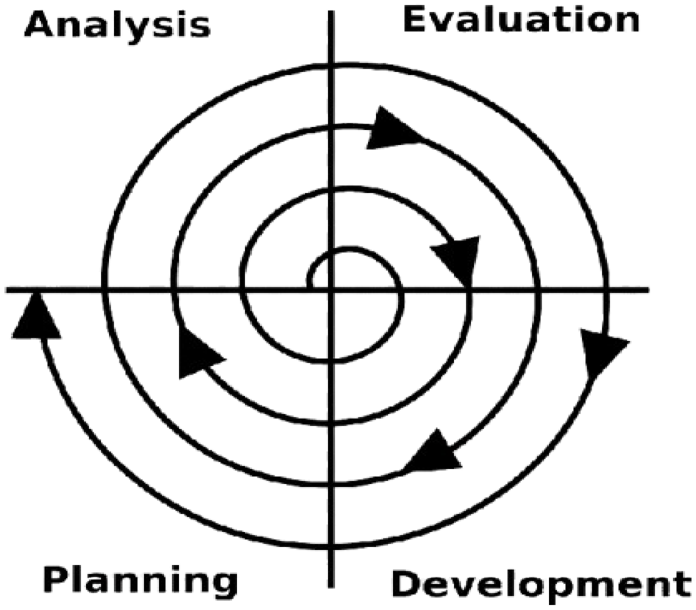

Figure 2.1: Spiral Methodology

2.3 Scrum Methodology

2.3.1 What is Scrum Methodology?

Scrum is an iterative and incremental agile framework that emphasizes developing complex products. It helps teams deliver working products within short cycles called sprints.

2.3.2 Key Elements of Scrum

Scrum Roles:

- **Product Owner:** The product owner manages the product backlog by prioritizing the list of features and improvements for the product, represents the user needs, conveys product progress, and facilitates communication between the development team and actual users.

- **Scrum Master:** Contrary to traditional project managers, Scrum Masters do not directly manage the development team rather they focus on creating an environment where the self-managed development team can concentrate on its development work. He creates a smooth working environment for the team and handles any problems (i.e., system failure, electricity outage, office maintenance) that the team faces.

- **Development Team:** The development team performs all the tasks related to development, such as design, construction, and testing of the product. The product owner gives his direction to the team, and the team self-organizes to deliver the desired product within the schedule as per the direction.

Scrum Events:

- **Sprint:** A sprint, a time-bound iteration cycle in Scrum, involves sprint planning, work approval by the team and product owner, development, evaluation, and potential acceptance/rejection of completed work.

- **Meetings:** Sprints involve four types of meetings: sprint planning (8 hours), daily scrums (15 minutes), sprint review (4 hours), and sprint retrospective (after review). They facilitate planning, development, analysis, and improvement.

- **Backlog Refinement:** The product backlog requires regular updates to accommodate changing priorities and evolving requirements. Maintaining its currency is essential to preserve its integrity and ensure that priorities remain appropriately aligned.

- **Scrum of Scrums:** Scrum of Scrums is a technique for scaling up project teams by forming smaller teams, each represented by an ambassador in meetings. It focuses on coordinating overlapping responsibilities and integration points between teams.

2.3.3 Scrum Artifacts

- **Product Backlog:** Dynamic list of tasks, features, and improvements, ordered by value and priority.

- **Sprint Backlog:** Selection of backlog items for a particular sprint, with actionable tasks.

- **Increment:** Cumulative sum of backlog items completed during a sprint.
- **Burndown Chart:** Visual tracking of remaining work versus time.

2.3.4 Advantages

- **Agile methodology:** Flexible and adaptable to changing needs.
- **Reduced communication gaps:** The product owner bridges the gap between stakeholders and developers.
- **Daily scrum meetings:** Clear visibility and quick issue resolution.
- **Scalability:** Large projects managed through "Scrum of Scrums" with cross-team collaboration.
- **Globally distributed teams:** Supported by video conferencing and web-based tools.
- **Incremental development:** Early product releases with core features, adding more later.
- **Faster time to market:** Launch early and gather feedback quickly.

2.3.5 Disadvantages

- **Requires Skill:** Effective scrum teams must be experienced and disciplined; weak teams may slow progress.
- **Scope Creep:** Frequent changes and additions can lead to uncontrolled growth of scope.
- **Burden of Meetings:** Daily standups and frequent reviews may frustrate members.
- **Limited Suitability for Large Projects:** May not scale well for highly complex or extensive undertakings.

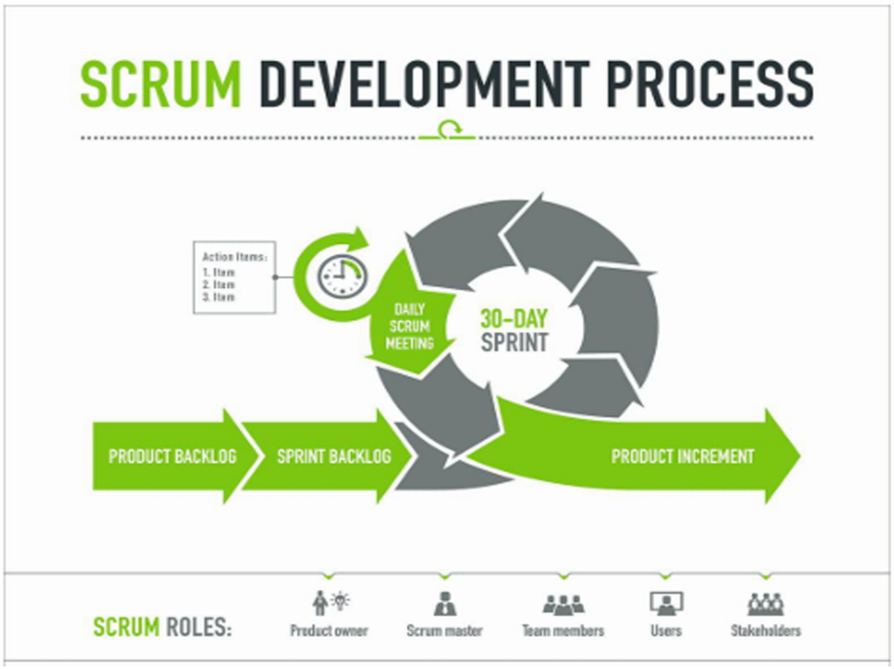

Figure 2.2: Scrum Methodology

2.4 Prototype Methodology

2.4.1 What is Prototype Methodology?

A prototype is a sample implementation of the system. It provides the limited and main functional capabilities of the proposed system. (moves in a circle) Build on earlier versions. It begins with analysis and requirements, followed by a quick design to give feedback on an "idea" of a system or product.

2.4.2 Advantages

- Users are actively involved in the development.
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available, leading to better solutions.
- Missing functionality can be identified easily.

2.4.3 Disadvantages

- Leads to implementing and then repairing the way of building systems.
- Practically, this methodology may increase the complexity of the system, as the scope of the system may expand beyond the original plans.
- An incomplete application may cause the application not to be used as the full system was designed.
- Incomplete or inadequate problem analysis. Focusing on prototypes may overshadow the comprehensive analysis needs.

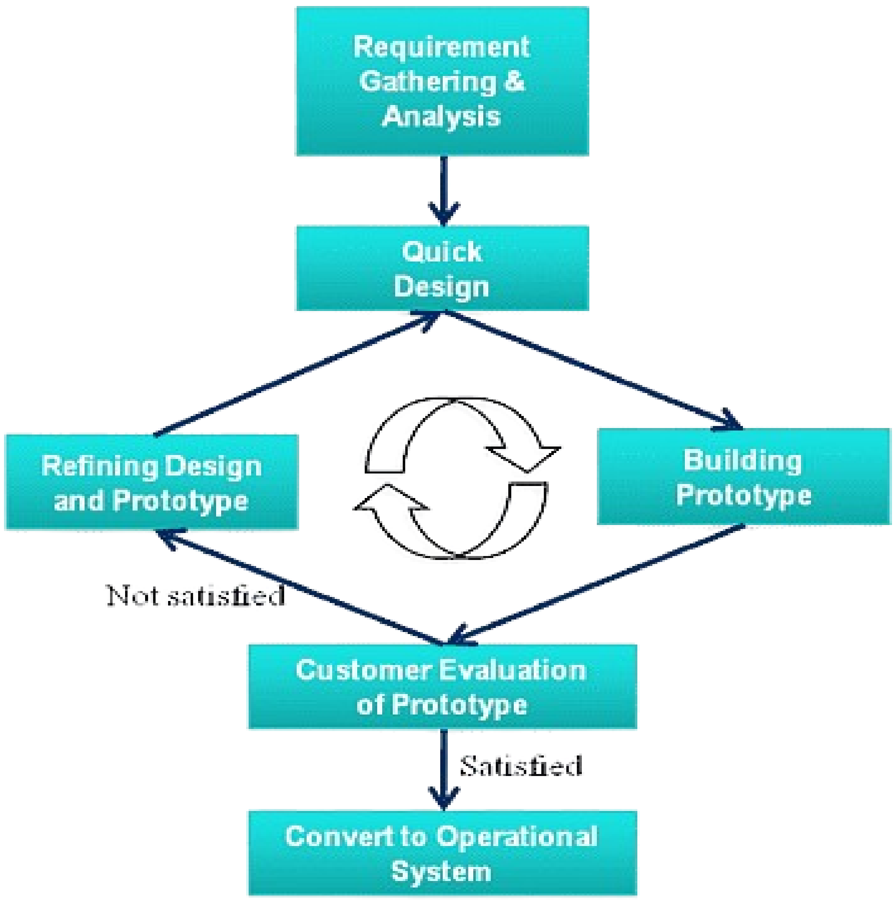
Requirement Gathering & Analysis
Quick Design
Building Prototype
Refining Design and Prototype
Not satisfied
Customer Evaluation of Prototype
Satisfied
Convert to Operational System

Figure 2.3: Prototype Methodology

2.5 Rational Unified Process (RUP)

2.5.1 What is RUP?

The Rational Unified Process (RUP) is an iterative software development framework that divides the process into distinct phases: Inception, Elaboration, Construction, and Transition. Each phase is defined by specific objectives, deliverables, and reviews, and each iteration produces incremental improvements. RUP emphasizes documentation, architecture-focused development, and stakeholder collaboration.

2.5.2 Advantages

- **Structured Process:** Clear definition of phases and deliverables provides guidance throughout the project.
- **Iterative Delivery:** Supports incremental development, allowing for progressive refinement.
- **Change Management:** Facilitates the handling and integration of changing requirements.
- **Quality Focus:** Strong emphasis on testing, analysis, and documentation throughout development.

2.5.3 Disadvantages

- **Complexity:** Highly structured and documentation-heavy, which may slow small teams or projects.
- **Resource Intensive:** Requires extensive planning, reviews, and artifacts, potentially increasing costs.
- **Learning Curve:** New teams require training to adopt and fully utilize RUP.

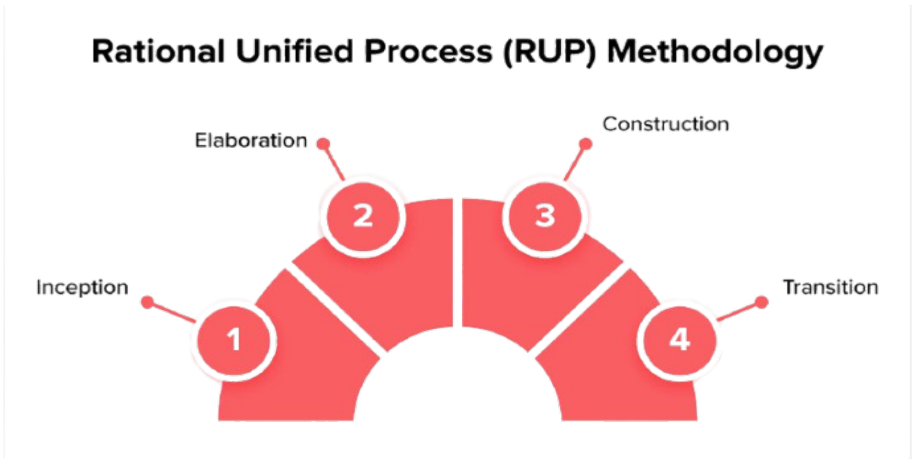

Figure 2.4: Rational Unified Process Methodology

2.6 Agile Methodology

2.6.1 What is Agile Methodology?

Agile is an incremental, iterative approach that values flexibility, collaboration, and rapid delivery of functional software in short cycles. Agile focuses on continuous involvement of stakeholders, adapting to changes throughout the project, and delivering working releases frequently.

2.6.2 Advantages

- Software is continuously delivered using the agile technique.
- Customers are happy since they receive all of the software's functional features after each Sprint.
- Customers are able to view the functional features that meet their needs.
- The current version of the product can incorporate any feedback or feature modifications from the customers.
- The Agile methodology necessitates regular communication between developers and business stakeholders.
- The well-designed product is given consideration in this process.

- Even in the final phases of development, modifications to the specifications are accepted.

2.6.3 Disadvantages

- Agile technique requires less documentation. The Agile process can occasionally include requirements that are not quite clear, making it challenging to forecast the desired outcome.
- It can be challenging to estimate the true work needed for a few projects that are just getting started in the software development life cycle.
- There is always a chance that the project will take forever because of its constantly changing characteristics.
- It is challenging to estimate the amount of work and resources needed for complex projects.

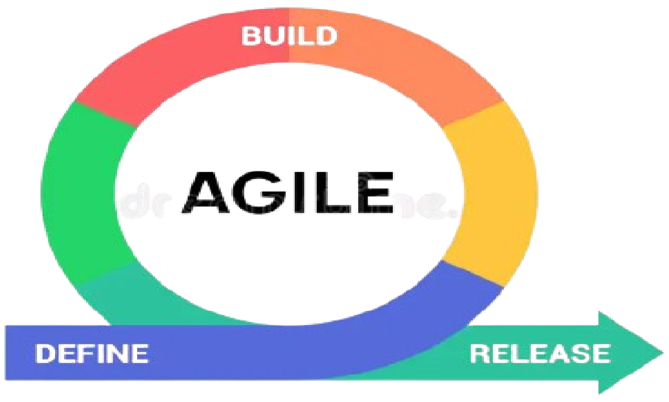

Figure 2.5: Agile Methodology

2.7 Extreme Programming Methodology (XP)

2.7.1 What is Extreme Programming (XP) Methodology?

Extreme Programming XP is a revolutionary agile software development methodology that emerged as a response to the limitations of traditional approaches. Software engineer Ken Beck introduced XP in the 90s with the goal of finding ways to write high-quality software quickly and being able to adapt to customers' changing requirements. In 1999, he refined XP approaches in the book "Extreme Programming Explained: Embrace Change." At its core, XP places a premium on customer satisfaction and adaptability, offering a dynamic framework that excels in the face of evolving project requirements.

2.7.2 Key characteristics of Extreme Programming

1. **Flexibility and Adaptability:** XP is designed to accommodate changes in requirements, even late in the development process. It emphasizes the importance of being able to respond quickly to customer feedback and changing business needs.

2. **Iterative Development:** XP employs iterative development, with short development cycles called iterations. Each iteration produces a potentially shippable product increment.

3. **Continuous Feedback:** Regular communication and feedback loops are essential in XP. Frequent releases and continuous integration ensure that the team receives constant input, allowing for quick adjustments and improvements.

4. **Collaboration:** Team collaboration is a key principle in XP. Developers, customers, and other stakeholders work closely together, promoting shared ownership of the project and collective decision-making.

5. **Test-Driven Development:** TDD XP places a strong emphasis on writing tests before writing the actual code. This ensures that the code is reliable and that changes do not introduce new bugs.

6. **Pair Programming:** In XP, developers work in pairs, with one actively writing code and the other reviewing each line as it is written. This collaborative approach helps catch errors early and encourages knowledge sharing.

7. **Simple Design:** XP advocates for simple and straightforward design solutions. The goal is to keep the codebase clean and easily maintainable while avoiding unnecessary complexity.

8. **Continuous Integration:** Developers integrate their work frequently, and automated builds and tests are run to ensure that the codebase is stable and functional.

2.7.3 Advantages

1. Continuous customer involvement ensures that the delivered product aligns with their expectations.

2. XP is highly adaptable to changing requirements, making it suitable for dynamic and evolving projects.

3. Frequent testing, pair programming, and continuous integration contribute to early error detection, reducing project risks.

4. Practices like pair programming enhance communication and collaboration within the development team.

5. Time savings are available because of the fact that XP focuses on the timely delivery of final products. XP teams save lots of money because they don't use too much documentation.

6. Pair programming ensures fewer mistakes, fewer rewrites, and stable program units.

7. The whole process is transparent with developers committing to what they will accomplish and showing progress.

8. Constant feedback is also the strong side. It is necessary to listen and make any changes needed in time.

2.7.4 Disadvantages

1. XP may be challenging for teams without the necessary skills and experience in agile and XP practices.

2. This methodology does not measure code quality assurance. It may cause defects in the initial code.
3. XP is not the best option if programmers and customers are separated geographically.
4. XP requires a lot of persistence, creativity, and lean thinking to adjust the system to the client's needs day by day. There is a lot of stress and pressure due to tight deadlines when the code has to be designed and tested immediately.
5. The focus of XP is definitely on the code rather than the design. Sometimes this can result in failing to implement the software requirements fully.
6. Documentation is given very little priority. Lack of defect documentation may lead to the occurrence of similar bugs in the future.
7. In many instances, the customer has no clear picture of the end result, which makes it almost unrealistic to accurately estimate scope, cost, and time.

2.7.5 When to Use Extreme Programming (XP) Methodology

- Requirements are highly dynamic.
- High-risk projects with fixed timelines.
- Development teams are small and work closely together.
- The technology stack supports automated testing.

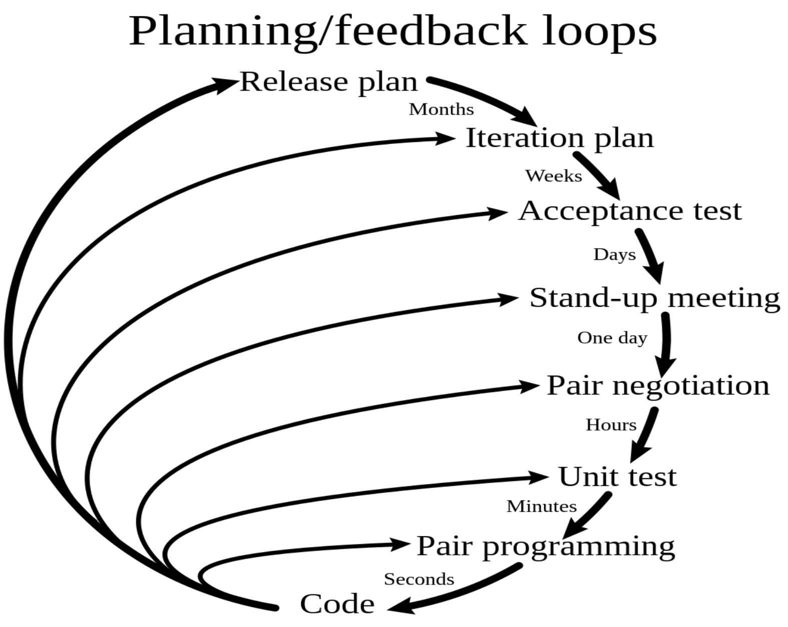

Figure 2.6: XP Methodology

2.8 Waterfall Methodology

2.8.1 What is Waterfall Methodology?

According to the waterfall project management methodology, project phases are executed in a precise order and don't move on until they have received final clearance. It is a widely used project management method with a linear approach. Every step in the Waterfall workflow must be finished before going on to the next.

2.8.2 Advantages

- Requires less coordination due to clearly defined phases and sequential processes.
- A clear project phase helps to clearly define the dependencies of work.
- The cost of the project can be estimated after the requirements are defined.
- Better focus on documentation of designs and requirements.
- The design phase is more methodical and structured before any software is written.

2.8.3 Disadvantages

- Risk of time wasted due to delays.
- Delaying testing until the end of development is common.
- There's no consideration for error correction.
- The model doesn't accommodate changes, scope adjustments, and updates well.
- Work on different phases doesn't overlap, which reduces the efficiency.
- Projects don't produce a working product until later stages.
- Not an ideal model to use for complex and high-risk projects.

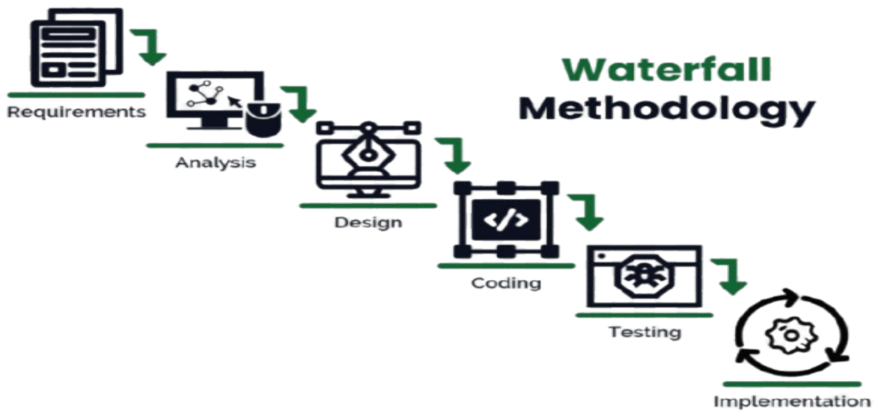

Figure 2.7: Waterfall

2.9 Benchmark Analysis

Benchmarking compares the major software development methodologies under parameters of flexibility, stakeholder involvement, risk management, and suitability for dynamic project requirements such as those presented by BookBridge.

Table 2.1: Benchmark Analysis of Development Methodologies

Methodology	Flexibility	Risk Mgmt	Customer Collaboration	Documentation	Complexity	Suitability
Waterfall	Low	Low	Low	High	Low	Defined, stable scopes
Spiral	High	High	High	High	High	Large, risky projects
Agile	High	Medium	High	Medium	Medium	Dynamic, evolving scopes
Scrum	High	Medium	High	Medium	Medium	Agile teams with collaboration
XP	High	High	High	Low	High	Small, high-maturity teams
RUP	Medium	High	Medium	High	High	Enterprise-grade projects

2.10 Selected Methodology For Our Project

2.10.1 Introduction

After evaluating various software development methodologies according to their suitability for BookBridge's goals, stakeholder requirements, system complexity, and need for adaptability, we have determined that the Agile methodology offers an optimal foundation for project management and product delivery. Agile principles are particularly suited to environments where requirements evolve, rapid response is critical, and collaboration between diverse roles is required for success.

2.10.2 Agile Methodology: Overview

Agile methodology is an iterative and incremental approach to software development, emphasizing adaptability, collaborative teamwork, continuous feedback, and regular delivery of high-quality software. Unlike rigid, sequential models, Agile breaks down projects into manageable units (sprints or iterations), ensuring that stakeholder feedback, user experience, and emerging requirements are addressed throughout the development lifecycle.

2.10.3 Core Principles of Agile

1. **Customer Collaboration:** Continuous engagement with stakeholders and users.
2. **Incremental Delivery:** Working product releases at the end of each iteration.
3. **Adaptability:** Rapid response to changes in requirements or feedback.
4. **Cross-Functional Teams:** Developers, testers, and other roles work closely together.
5. **Continuous Improvement:** Processes and products are regularly reviewed and improved.
6. **Transparency & Communication:** Daily interactions, reviews, and progress updates.

2.10.4 Agile Process Phases

1. **Planning**
    - Define current goals, user stories, and priorities for the upcoming sprint.
    - Prepare product backlog for new features, fixes, and improvements.
2. **Iteration/Sprint Execution**
    - The development team selects items from the backlog for implementation.
    - Work proceeds in short cycles, typically 2-4 weeks per sprint.

- Daily stand-up meetings maintain communication and identify issues early.

3. **Review**
    - Sprint review meetings showcase completed work to stakeholders.
    - Collect feedback and new requirements based on demonstrated functionality.
4. **Retrospective**
    - The team discusses opportunities for process and product enhancement.
    - Concrete plans are made for continuous improvement in future iterations.
5. **Release**
    - Deploy the completed, tested features to production or pilot groups.

2.11 Reasons for Choosing Agile

We chose to base this project on Agile methodology because it fits really well with the unique nature of this book-sharing platform and the ever-changing environment it operates in. The platform aims to solve real-world issues in how students and communities exchange books, focusing on being flexible, user-friendly, and quickly adding new features. Here's why Agile is the best fit for this project:

2.11.1 Facilitates Frequent Incremental Updates

Since our book-sharing system is all about community and sharing, keep in mind that we can roll out new features gradually and smoothly. Using an agile approach lets our team constantly deliver small, useful updates—things like adding ways to contribute books, managing book clubs, tracking credits, or sending notifications—all by the end of each working cycle. This way, everyone gets to see some real progress early on, which keeps things transparent and exciting. Plus, it helps the platform start delivering value right away, even while we're still working on the more advanced features for later on.

2.11.2 Enables Responsive Feature Modification

User feedback, changing academic needs, and new developments in similar platforms mean we need to be able to tweak and improve the system quickly. That's where Agile comes in with its sprint-based setup, making sure that suggestions, bug fixes, and new feature ideas are quickly added to the to-do list and tackled in the next planning session. Whether it's making the UI smoother, simplifying how book requests work, adding verification steps, or improving reports, Agile helps us stay flexible and keep things moving without major hiccups.

2.11.3 Supports Comprehensive Stakeholder Collaboration

When it comes to a book-sharing system, there are a bunch of different folks involved the readers, club moderators, system admins, and more. Using Agile methods helps keep everyone in the loop, making sure what users need, what faculty suggest, and what the admins decide all come together smoothly. During sprint reviews and retrospectives, everyone including faculty, developers, users to gets a chance to share their thoughts, tweak priorities, and make sure we're all on the same page for what's coming next.

2.11.4 Accommodates Complex, Evolving Requirements

A great book-sharing site isn't just a boring, static page. It's a lively, ever-changing platform that adapts to things like school calendars, new reading trends, or community events. You might see new stuff like digital wallets for books, clubs based on regions, custom dashboards for different user roles, or even connections with schools and libraries. These features can be added, tested, and improved over time as the platform grows. An agile approach works really well here because it keeps things flexible, letting teams keep discovering what users need and delivering small improvements instead of sticking to strict, fixed plans from the start.

2.11.5 Ensures Real-time Quality Feedback and Risk Management

Breaking down the work into small, manageable releases and constantly testing with users helps make sure quality is built right in from the start. Any issues, whether technical glitches or process hiccups, tend to show up early during review sessions and hands-on testing. This way, we catch big problems or user frustration early on, instead of them sneaking in at the end. Each sprint zeroes in on specific goals, allowing us to check and improve key parts like approving contributions, fulfilling requests, or managing clubs before moving on to the next big update.

2.11.6 Accelerates Academic and Community Alignment

This project is built on a real understanding of what students need and how to enable the community. We use an agile approach, so feedback from teachers, club leaders, and users is part of the process all the way through—not just at the beginning. This helps make sure the platform stays easy to use, affordable, and fun for everyone involved. At the same time, it encourages everyone to work together and keep reading a special, ongoing part of the culture.

2.11.7 Maximizes Flexibility for Future Growth

Since we're looking at expanding into new regions, adding gamification features, and building out a mobile app in the future, we want a methodology that can handle both what we need now and what's coming soon. Agile is perfect because it gives us the flexibility to add things like smart filters, analytics dashboards, or even integrate with other tools as we grow whether we're reaching more users around the world or targeting different groups locally.

2.12 Reasons Not to Choose Other Methodologies

When it comes to building this community-focused, feature-packed book-sharing platform, sticking with traditional or other modern development methods just doesn't hold up quite as well when you compare them to Agile. Here's a quick rundown of why some of those older and newer approaches might not be the best fit for hitting our goals or tackling the unique challenges this project throws at us.

2.12.1 Waterfall Model

The Waterfall method follows a straight-line process—requirements, design, coding, testing, launching, and upkeep. But this rigid way of working doesn't really fit if your project's requirements are likely to change based on feedback from users, faculty, or how things are running. Once you finish a phase in Waterfall, going back to make changes can be really difficult and time-consuming. Since we'd like to keep adding features, like club upgrades, a wallet system for books, new user roles, or connecting with other schools, the inflexibility of Waterfall and discovering problems late in the game could cause delays, make improvements slower, and leave stakeholders less happy.

2.12.2 Spiral Model

The Spiral model is really good at managing risks and trying out prototypes in small steps, but it also relies a lot on detailed paperwork, regular risk checks, and a lot of planning before actually starting. While this approach works well for really big or risky projects, it can be overkill for smaller projects where quick feedback and constant input from stakeholders are more effective. Its complicated nature, longer planning stages, and focus on documentation can slow down delivering new features and make it harder to keep up with changing academic schedules, user needs, and staying ahead of competitors.

2.12.3 Prototype Methodology

Prototyping is great for getting early feedback, playing around with ideas, and testing out specific features. But on its own, it doesn't have the structure needed to develop, track, and maintain a full, ready-for-production platform over multiple rounds. This project needs more than messing around with interface ideas or workflows. We need to gradually roll out fully working, secure, and scalable modules—like verified contributions, club management, or real-time notifications—which can't be handled through prototyping alone. Relying only on prototypes risks scope creep, incomplete integration, and a lack of solid process control, making prototyping not enough as the main approach.

2.12.4 Incremental Model

The Incremental model lets you deliver parts of the product and add features step by step. However, it doesn't automatically promote regular feedback from users, flexible planning, or teamwork across different areas. That's a real issue for a platform that depends on input from the community, guidance from teachers, and learning from real-world use. Plus, it can lead to splitting the project into separate pieces that may not fit together well, especially if there's no clear process for coordinating, setting priorities, or adjusting based on student and user feedback.

2.12.5 Rational Unified Process (RUP)

RUP divides the project into clear phases and emphasizes detailed documentation, architecture-focused development, and predefined deliverables. However, for a small, community-focused project that values speed, quick updates, and practical flexibility, these requirements can feel overwhelming. The need to keep detailed records and stick strictly to producing specific artifacts might slow things down, making it harder to keep up with the fast pace of academic and community book sharing.

2.12.6 Scrum, Extreme Programming (XP), and Other Agile Hybrids

While Scrum and XP offer some benefits of Agile, Scrum's fixed roles and sprint planning can feel pretty rigid, especially in a university setting where students, faculty, staff, and other stakeholders often have changing roles or overlapping responsibilities. XP requires strict engineering practices like pair programming and test-driven development, but those can be tricky to implement when your team has varied skill levels or limited resources. Even though these methods are part of the Agile toolkit, they can sometimes introduce unnecessary rules or steep learning curves if they aren't adjusted to fit the specific needs of your team.

Chapter 3

Work Breakdown Structure (WBS)

3.1 Introduction to Work Breakdown Structure

A Work Breakdown Structure (WBS) is a way of breaking down a large and complex project into smaller and manageable parts. It is like a tree diagram where the top represents the overall project goal, and the branches represent smaller deliverables, sub-deliverables, and tasks. The main purpose of a WBS is to organize all the work required for the project in a clear and structured way so that nothing is left out and everyone knows their responsibility. The WBS ensures that the project team and stakeholders share the same understanding of the scope. By breaking down the project into deliverables, it becomes easier to estimate time, cost, and resources. Each work package in the WBS can be assigned to a responsible person or team, making accountability clear. The structure also helps in monitoring progress since each smaller task can be tracked to see if it is completed on time and meets the quality standard. For the BookBridge Project, the WBS covers all essential parts such as user sign-up and verification, book sharing and requesting, book wallet, Litera Clubs, search and filter functions, reviews and ratings, dashboards, notifications, and data security. It also includes supporting aspects like project management, requirements analysis, system design, testing, deployment, documentation, and risk management. Without a WBS, it would be difficult to manage all these elements together, and important deliverables could easily be missed. Therefore, WBS is not just a chart but a tool that gives structure and clarity. It provides a clear roadmap of how the BookBridge project will be completed, step by step. This helps in avoiding confusion, reducing risks, and ensuring that the final outcome is aligned with the project objectives.

3.2 Principles of Work Breakdown Structure

3.2.1 The 100% Rule

The 100% Rule is the most important principle in designing a WBS. It means the WBS should capture the entire scope of the project—nothing more and nothing less. All deliverables, activities, and supporting tasks must be included in the structure. Every level of the WBS should add up to 100In the BookBridge project, this rule ensures that every feature and supporting function is covered. From sign-up and verification to Litera Clubs, wallet, search, reviews, dashboards, and notifications, all features are included in the WBS. At the same time, activities outside the project scope, such as third-party integrations not planned in this phase, are excluded. By applying the 100This rule also helps in project control. When all deliverables are included, stakeholders can clearly see what will be delivered. It avoids surprises during later stages and prevents scope creep. For example, if "user feedback system" was not included, the project would lack an important feature. Similarly, adding unrelated tasks like "partner integration" would waste time and resources. The 100

3.2.2 Planned Outcomes, Not Planned Actions

A WBS should describe outcomes rather than actions. This means that deliverables should be expressed as results, not the steps taken to achieve them. For example, instead of writing "Design Login Page" or "Code Verification System," the WBS should include "User Verification Module." In BookBridge, this approach makes each deliverable easier to measure. Outcomes such as "Wallet System" or "Notification Service" can be validated once they are complete. On the other hand, listing actions such as "coding," "testing," or "designing" can make the WBS repetitive and confusing. Focusing on outcomes also helps stakeholders evaluate success by looking at whether the system feature works as intended, not just whether certain activities were performed. This principle also promotes clarity. Outcomes are easy to understand for both technical and non-technical people. A stakeholder may not understand "backend API integration," but they will understand

"Book Wallet Feature." By using outcomes, the WBS becomes a communication tool that is accessible to everyone.

3.2.3 Level 2 is the Most Important

The second level of the WBS is often the most critical because it defines the main structure of the project. These Level 2 elements represent the major deliverables and categories that divide the project into logical sections. They provide the framework for planning, reporting, and assigning work. For the BookBridge project, Level 2 includes:

- Project Management & Governance
- Requirements & Analysis
- UX/UI Prototyping
- System Architecture & Design
- Core Features
- Integrations & External Services
- Quality Assurance & Testing

These categories ensure that all parts of the project are covered. For example, Core Features contain the user-facing modules like Sign-up, Wallet, Litera Clubs, Search, Reviews, Dashboards, and Notifications. Project Management ensures proper planning and governance. Quality Assurance covers testing and validation, while Risk Management deals with uncertainties. Without a strong Level 2, the WBS would be confusing and incomplete.

3.2.4 The Four Elements in Each WBS Element

Every WBS element should include four important details:

- **Deliverable** – what is being produced.
- **Owner** – who is responsible for delivering it.
- **Start and Finish Criteria** – when the work begins and ends.

- **Performance Standards** – how success will be measured.

For example, in BookBridge:

- Deliverable: Wallet System
- Owner: Development Team under the Product Manager
- Start and Finish: Begins after design approval, ends when tested and deployed successfully
- Performance Standards: Wallet transactions are instant, accurate, and protected against fraud

Including these four elements ensures clarity and accountability. It prevents confusion about who is responsible and when a deliverable is complete. It also makes quality measurable and helps in tracking project progress.

3.2.5 Mutually-exclusive Elements

Each WBS element should be unique and non-overlapping. If two elements cover the same work, it can cause duplication of effort, confusion about ownership, and errors in estimating cost or time. In BookBridge, "Reviews & Ratings" and "Personalized Dashboards" are kept separate even though both relate to users. Reviews deal with feedback, while dashboards present personalized information. By separating them, responsibilities are clearer and work is not repeated. This principle ensures the project remains efficient and transparent. Each task is assigned to one team, with no duplication. It also supports modular development where features can be built, tested, and improved independently without conflict.

3.2.6 The 40-Hour Rule of Decomposition

The 40-hour rule says that no single work package should take more than one work week (around 40 hours) to complete. When it is so, it should be reflected into small portions. This simplifies work to track and assign as well as monitor. A feature such as Sign-up and Verification in BookBridge can be broken down into small packages like UI design, backend integration and email verification. This is less than 40 hours each one can handle. This makes it possible to quickly check the progress, and identify problems at early maturity.The rule also assists in even distribution of jobs among team members and is able to keep them motivated by providing them with some attainable targets.

3.2.7 The 4% Rule of Decomposition

The 4% rule is established to make sure any one work package does not occupy a bigger part of project effort than 4%. This will not allow a project to be overwhelmed by a single task. In terms of the BookBridge project, say that it is estimated that the project is supposed to take 1000 hours no work package ought to exceed the 40 hours mark. At a glance, a module such as Security System may appear to have more content in it but this needs to be broken down into smaller components such as authentication, encryption and tracking, to follow the regulation of 4% rule.This balances the project and eliminates the bottlenecks. It also enhances control as there is a limit to reasonable effort taken in working on everything.

3.3 WBS of Book Bridge

100% Rule

The BookBridge WBS follows the 100% Rule. It includes all features, management processes, testing, deployment, documentation, and risk management. Nothing is left out, and nothing outside the scope is added.

Outcome Focus

The WBS uses outcomes, not actions. Features like "Wallet System" and "Notification Service" are written as deliverables. This makes success easy to measure and ensures clarity for both technical and non-technical stakeholders.

Avoidance of Redundancy

The structure avoids overlaps. For example, "Search & Filters" and "Reviews & Ratings" are treated as separate items even though both are user-related. This prevents duplication of work and confusion about responsibilities.

4% Rule Compliance

Each work package is decomposed so that it does not exceed 4% of the total effort. This ensures balance and prevents one task from dominating resources. Large modules are broken down into smaller, manageable tasks.

The Work Breakdown Structure (WBS) of the BookBridge Project organizes the project scope into six major deliverables at Level 2, each further broken down into sub-deliverables and work packages. The diagram below (Figure X) illustrates the hierarchical structure.

Level 2 Deliverables include:

1. **Project Management & Governance:** covers initiation, planning, monitoring, and project closure.

2. **Requirements & Analysis:** includes elicitation, non-functional requirements, and acceptance criteria.

3. **UX/UI Prototyping:** focuses on information architecture, wireframes, usability testing, and style guidelines.

4. **Core Features:** encompasses sign-up & verification, book sharing, wallet system, Litera Clubs, search, reviews, dashboards, and real-time notifications.

5. **Integration & External Services:** primarily the payment API and related integrations.

6. **Quality Assurance & Testing:** includes component, unit, and security testing.

The diagram visually demonstrates how each of these Level 2 elements is decomposed into Level 3 and Level 4 work packages. For instance, under 4.0 Core Features, the "4.1 Sign-up & Verification" package is broken down into user registration UI/backend, email verification, and admin verification tools. Similarly, the "4.3 Book Wallet" is divided into wallet model design, transaction recording, fraud protection, and admin dashboards. This decomposition ensures that all tasks are measurable, assignable, and traceable. The WBS diagram also confirms alignment with the 100% Rule (all deliverables captured), outcome orientation (focus on results such as "Wallet System" rather than

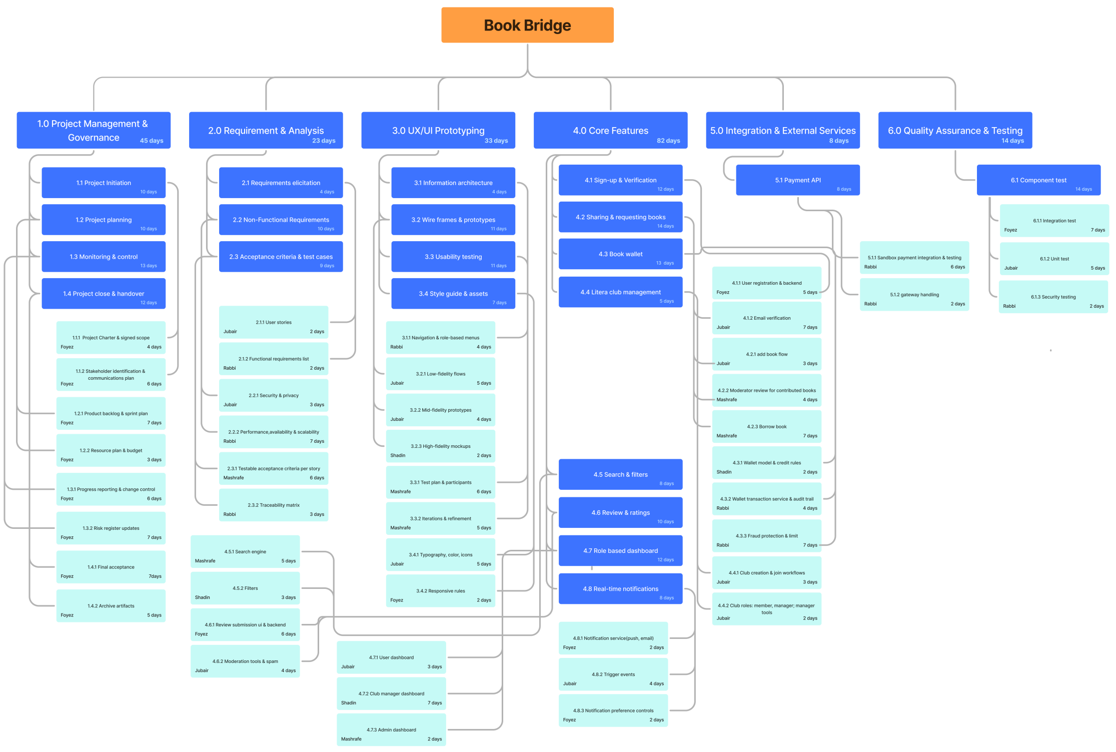
Book Bridge
1.0 Project Management & Governance 45 days
2.0 Requirement & Analysis 23 days
3.0 UX/UI Prototyping 33 days
4.0 Core Features 82 days
5.0 Integration & External Services 8 days
6.0 Quality Assurance & Testing 14 days
1.1 Project Initiation 10 days
1.2 Project planning 10 days
1.3 Monitoring & control 13 days
1.4 Project close & handover 12 days
1.1.1 Project Charter & signed scope Foyez 4 days
1.1.2 Stakeholder identification & communications plan Foyez 6 days
1.2.1 Product backlog & sprint plan Foyez 7 days
1.2.2 Resource plan & budget Foyez 3 days
1.3.1 Progress reporting & change control Foyez 6 days
1.3.2 Risk register updates Foyez 7 days
1.4.1 Final acceptance Foyez 7days
1.4.2 Archive artifacts Foyez 5 days
2.1 Requirements elicitation 4 days
2.2 Non-Functional Requirements 10 days
2.3 Acceptance criteria & test cases 9 days
2.1.1 User stories Jubair 2 days
2.1.2 Functional requirements list Rabbi 2 days
2.2.1 Security & privacy Jubair 3 days
2.2.2 Performance,availability & scalability Rabbi 7 days
2.3.1 Testable acceptance criteria per story Mashrafe 6 days
2.3.2 Traceability matrix Rabbi 3 days
3.1 Information architecture 4 days
3.2 Wire frames & prototypes 11 days
3.3 Usability testing 11 days
3.4 Style guide & assets 7 days
3.1.1 Navigation & role-based menus Rabbi 4 days
3.2.1 Low-fidelity flows Jubair 5 days
3.2.2 Mid-fidelity prototypes Jubair 4 days
3.2.3 High-fidelity mockups Shadin 2 days
3.3.1 Test plan & participants Mashrafe 6 days
3.3.2 Iterations & refinement Mashrafe 5 days
3.4.1 Typography, color, icons Jubair 5 days
3.4.2 Responsive rules Foyez 2 days
4.1 Sign-up & Verification 12 days
4.2 Sharing & requesting books 14 days
4.3 Book wallet 13 days
4.4 Litera club management 5 days
4.5 Search & filters 8 days
4.6 Review & ratings 10 days
4.7 Role based dashboard 12 days
4.8 Real-time notifications 8 days
4.1.1 User registration & backend Foyez 5 days
4.1.2 Email verification Jubair 7 days
4.2.1 add book flow Jubair 3 days
4.2.2 Moderator review for contributed books Mashrafe 4 days
4.2.3 Borrow book Mashrafe 7 days
4.3.1 Wallet model & credit rules Shadin 2 days
4.3.2 Wallet transaction service & audit trail Rabbi 4 days
4.3.3 Fraud protection & limit Rabbi 7 days
4.4.1 Club creation & join workflows Jubair 3 days
4.4.2 Club roles: member, manager; manager tools Jubair 2 days
4.5.1 Search engine Mashrafe 5 days
4.5.2 Filters Shadin 3 days
4.6.1 Review submission ui & backend Foyez 6 days
4.6.2 Moderation tools & spam Jubair 4 days
4.7.1 User dashboard Jubair 3 days
4.7.2 Club manager dashboard Shadin 7 days
4.7.3 Admin dashboard Mashrafe 2 days
4.8.1 Notification service(push, email) Foyez 2 days
4.8.2 Trigger events Jubair 4 days
4.8.3 Notification preference controls Foyez 2 days
5.1 Payment API 8 days
5.1.1 Sandbox payment integration & testing Rabbi 6 days
5.1.2 gateway handling Rabbi 2 days
6.1 Component test 14 days
6.1.1 Integration test Foyez 7 days
6.1.2 Unit test Jubair 5 days
6.1.3 Security testing Rabbi 2 days

Figure 3.1: Work break-down structure of Book Bridge

tasks like "Code Wallet"), mutual exclusivity (distinct modules without overlap), and decomposition rules (no single package exceeding 40 hours or 4% of project effort).

Chapter 4

Risk Management

4.1 What is Risk Management

Risk Management is a systematic approach for identifying, assessing, and addressing potential challenges and uncertainties that may affect the project's success, stability, and delivery. The process enables the anticipation and proactive management of threats, thereby protecting the project from unforeseen complications and improving its resilience and adaptability. Through careful evaluation, risk management ensures critical issues are handled before they escalate, maintaining the integrity and continuity of the project.

4.2 Risk Mitigation, Monitoring, and Management

4.2.1 Risk Mitigation

Risk mitigation involves developing strategies to reduce the likelihood or impact of potential risks. Common mitigation strategies include:

- Implementing preventive measures (e.g., code reviews, backups)
- Using risk avoidance strategies (e.g., selecting stable technologies)
- Allocating contingency resources for unforeseen events

4.2.2 Risk Monitoring

Risk monitoring is the continuous process of tracking identified risks, detecting new risks, and evaluating the effectiveness of mitigation measures. Monitoring ensures that risks remain under control and any escalation is addressed promptly.

4.2.3 Risk Management

Risk management is the overall approach to risk, including:

- Identifying and categorizing risks
- Assessing their probability and impact
- Planning and implementing mitigation strategies
- Monitoring and updating risk plans throughout the project lifecycle

4.3 Risk Plan Sheet

Risk ID	Risk Description	Probability (1-5)	Impact (1-5)	Priority (= Probability × Impact)	Mitigation Strategy	Responsible Person
R1	Misuse of Book Wallet Credits and Community Scamming	4	5	20	Implement strict wallet usage policies, enable fraud detection algorithms, and monitor transactions	Security Lead
R2	Integration Issues with Payment API	4	4	16	Perform early testing with sandbox environment; maintain backup payment option	Backend Developer
R3	Scheduling and Unexpected Increase in Project Cost	3	4	12	Monitor budget weekly, maintain contingency funds, and use project management tools for scheduling	Project Manager
R4	Data Loss during Database Migration	3	5	15	Schedule migration during low-traffic hours, take full backups before migration	Database Admin
R5	Lack of Expert Engineer	2	4	8	Conduct training sessions and hire external consultants if needed	HR Manager
R6	Requirement Changes from Client	4	3	12	Lock requirements after client sign-off and implement a change request process	Project Manager

Table 4.1: Risk Plan Sheet

4.4 Risk Information Sheet for Individual Members

Member: Abu Sayed

Risk ID	R1
Date	August 20, 2025
Probability	4
Impact	5
Description	Misuse of Book Wallet credits and potential community scams may lead to financial losses and user distrust.
Refinement / Context	1. Fraudulent activities like fake transactions or exploiting promotional credits. 2. Inadequate fraud detection systems and insufficient user verification.
Mitigation / Monitoring	1. Implement fraud detection algorithms and wallet usage limits. 2. Regularly monitor transactions for anomalies. 3. Use multi-factor authentication for wallet access. 4. Educate users about safe community practices.
Management / Contingency Plan / Trigger	1. Temporarily freeze suspicious accounts and credits. 2. Deploy real-time alerts for abnormal spending patterns.
Current Status	20/08/25: Fraud monitoring tool configuration in progress.
Originator	Abu Sayed
Assigned	Sayed

Table 4.2: Risk Information Sheet – Abu Sayed

Member: Shadhin Nandi

Risk ID	R2
Date	August 20, 2025
Probability	4
Impact	4
Description	Integration issues with Payment API may disrupt payment processing for Book Bridge users.
Refinement/Context	1. API version changes without prior notice. 2. Poor error handling during high traffic transactions.
Mitigation/Monitoring	1. Test API in sandbox environment before deployment. 2. Maintain backup payment method (alternative gateway). 3. Schedule regular API update checks.
Management/Contingency Plan/Trigger	1. Switch to backup gateway in case of primary API failure. 2. Notify users and provide manual payment options temporarily.
Current Status	20/08/25: Sandbox testing for API completed.
Originator	Shadhin Nandi
Assigned	Shadhin Nandi

Table 4.3: Risk Information Sheet – Shadhin Nandi

Member: Mashrafe

Risk ID	R4
Date	August 20, 2025
Probability	3
Impact	5
Description	Data loss during database migration could result in permanent loss of user and transaction data.
Refinement/Context	1. Migration scheduled during active usage period. 2. No proper backup or rollback plan before migration.
Mitigation/Monitoring	1. Take full backups before migration. 2. Perform migration during low-traffic hours. 3. Implement rollback mechanism for failed migration.
Management/Contingency Plan/Trigger	1. Restore from backup immediately if migration fails. 2. Notify all users about expected downtime in advance.
Current Status	20/08/25: Backup completed, migration scheduled for off-peak time.
Originator	Mashrafe
Assigned	Mashrafe

Table 4.4: Risk Information Sheet – Mashrafe

Chapter 5

Function Point Analysis

5.1 What is Function Point?

A Function Point (FP) is a unit of measurement used in Function Point Analysis (FPA) to quantify the functional size of a software application. It represents the amount of functionality delivered to the user, based solely on the user's logical view of the system.

In essence, a Function Point answers the question, "How much value does this software provide to its end-users?"

5.2 What is Function Point Analysis (FPA)?

Function Point Analysis (FPA) is a standardized method for measuring the functional size of an information system from a user's perspective. It quantifies software size by counting user-recognizable functions, such as inputs, outputs, inquiries, logical files, and external interfaces, then adjusting these counts to derive a total function point value. This metric can then be used to estimate project effort, cost, schedule, and to compare software across different platforms or projects.

5.3 Types of FPA

FPA works by identifying and counting the five key components of software functionality which are considered from the user's perspective:

- **External Inputs (EIs):** Data or control information entering the system from outside.

- **External Outputs (EOs):** Data or control information leaving the system.
- **External Inquiries (EQs):** Requests that result in data retrieval and presentation.
- **Internal Logical Files (ILFs):** User-identifiable groups of logically related data or control information residing within the system's boundary.
- **External Interface Files (EIFs):** User-identifiable groups of logically related data or control information residing outside the system's boundary but referenced by the system.

Each of these components is then assigned a weight (simple, average, or complex) based on its complexity, resulting in an unadjusted Function Point (UFP) count.

5.4 Advantages and Disadvantages of FPA

Function Point Analysis (FPA) is a widely recognized estimation technique, and like any method, it offers distinct advantages and disadvantages.

5.4.1 Advantages of FPA

- **Early Estimates** You can estimate the project size and cost right at the beginning (when you only have the list of features), without needing any design or coding details.
- **Code-Independent** The size measurement is the same whether you write the software in Java, Python, or anything else. It focuses on the user experience, not the technology.
- **Better Management** It gives managers and customers a clear, numbers-based way to talk about the project's scope and size. If the customer asks for a new feature, you can quickly and clearly say how much bigger the project just got.

- **Measure Productivity** It helps compare how efficient different teams or projects are (e.g., "Team A delivers 10 function points per month, while Team B delivers 8").

5.4.2 Disadvantages of FPA

- **Subjective**The process requires the person counting the points to make judgments about how complex each feature is. Different people might count the same project differently.
- **Takes Time** It's a formal, detailed process that can take a long time to complete properly, especially for a very large project.
- **Hard to Learn** It uses strict, official rules (from the IFPUG group). You need special training and practice to do it accurately.
- **Ignores Hidden Work** It mainly counts user features. It is not great at measuring the extra effort needed for non-functional things like top-level security, super-fast performance, or complex math algorithms.

5.5 Function Point Calculation of Book Bridge

5.5.1 Unadjusted FP Count (UFC)

The Unadjusted Function Point Count (UFC), often referred to as Unadjusted Function Points (UFP), is the initial measure of a software application's functional size before any consideration for its technical complexity or execution environment.

In simple terms, the UFC is the raw count of all the features and data groups the system delivers to the user, weighted by their perceived complexity.

The UFC is calculated by identifying and counting the five main components of software functionality from the user's perspective, assigning a complexity weight to each, and then summing up the results

Table 5.1: Assumed EI / EO / EQ / ILF / EIF counts per BookBridge feature

Feature	EI	EO	EQ	ILF	EIF
User registration (email verification)	1	1	0	1	1
Email verification (provider)	0	0	0	0	1
Login / Logout	0	0	0	0	0
Password recovery / reset	1	1	0	0	1
Role-based access (Student / Admin / Moderator)	0	0	1	1	0
Profile create / edit	1	1	0	1	0
Account verification by admin / moderator	1	1	0	1	0
Personalized dashboard (user-specific)	0	2	1	1	0
Add / contribute a book (upload metadata)	1	0	0	1	0
Edit / remove contributed book	1	0	0	1	0
Upload book cover & metadata (storage)	1	0	0	0	1
View book detail page	0	1	1	0	0
Book status tracking (available / requested / exchanged)	0	1	0	1	0
Search books (by title / author)	0	0	3	0	0
Filters & sorting (location, condition, rating)	0	0	2	0	0
Request / borrow a book	1	1	0	1	0
Accept / reject exchange requests	1	1	0	1	0
Automated status updates (notifications)	0	2	0	0	0
Borrow / share history (user)	0	1	1	1	0
Wallet — earn credits (share book)	1	1	0	1	0
Wallet — spend credits (borrow)	1	1	0	1	0
Wallet transaction history	0	1	1	1	0
Payment gateway integration (top-up / donations)	0	0	0	0	1
Refund / dispute handling	1	1	0	1	0
Add review & star rating	1	0	0	1	0
View average rating (book)	0	1	1	0	0
Report inappropriate review	1	1	0	1	0
Create / join Litera Club	1	0	1	1	0

continued on next page

Table 5.1 – *continued from previous page*

Feature	**EI**	**EO**	**EQ**	**ILF**	**EIF**
Club pages (events, members)	0	1	1	1	0
Club chat / discussion thread	1	1	1	0	0
Real-time notifications (in-app)	0	2	0	0	0
Email + in-app notification delivery (provider)	0	0	0	0	1
Moderator — verify new content	1	0	0	1	0
Moderator — manage disputes	1	1	0	1	0
Admin dashboard & reports (exports)	0	2	1	1	0
Admin — manage users / categories	1	1	1	1	0
Cloud storage for images (external)	0	0	0	0	1
Leaderboard (top contributors)	0	1	1	0	0
Recommendations (rule / AI-based)	0	1	1	0	0
Help & FAQ pages	0	1	1	0	0
Totals:	**19**	**30**	**19**	**23**	**7**

Notes:

- EI = External Inputs, EO = External Outputs, EQ = External Queries.
- ILF = Internal Logical Files, EIF = External Interface Files.
- Totals at the bottom provide the base counts for Unadjusted Function Points (UFP).

Unadjusted Function Count (UFC) Calculation

The Unadjusted Function Count (UFC) or Unadjusted Function Points (UFP) is obtained by multiplying the number of functions of each type by their corresponding complexity weights and summing the results.

$$\boxed{UFC = (EI \times W_{EI}) + (EO \times W_{EO}) + (EQ \times W_{EQ}) + (ILF \times W_{ILF}) + (EIF \times W_{EIF})}$$

where:

- EI = Number of **External Inputs**
- EO = Number of **External Outputs**
- EQ = Number of **External Queries**
- ILF = Number of **Internal Logical Files**
- EIF = Number of **External Interface Files**

and the weights W depend on the **complexity level** of each component as defined by IFPUG or the adopted standard.

Table 5.2: Weights of FPA components for BookBridge

Item	**Simple**	**Average**	**Complex**
External Inputs (EI)	7	8	9
External Outputs (EO)	10	13	20
External Queries (EQ)	8	9	11
Internal Logical Files (ILF)	6	11	15
External Interface Files (EIF)	1	3	5

For Book Bridge the weight of external input is simple (7); external output is average (13); external inquiry is average (9); the weight of the internal logical file (ILF) is complex (15), the weight of external logic files is average(3).

$$UFC = (EI \times 7) + (EO \times 13) + (EQ \times 9) + (ILF \times 15) + (EIF \times 3)$$

$$UFC = (19 \times 7) + (30 \times 13) + (19 \times 9) + (23 \times 15) + (7 \times 3)$$

$$UFC = 133 + 390 + 171 + 345 + 21 = \boxed{1059}$$

Interpretation:

- The calculated Unadjusted Function Points (UFP) for BookBridge is **1059**.
- This represents the raw functional size of the system before applying the Technical Complexity Factor (TCF).

- The distribution reflects realistic complexity: many moderate CRUD inputs/outputs, a fair number of queries, and a handful of integrations.

5.5.2 Technical Complexity Factor (TCF)

Technical Complexity Factors (TCFs) are a set of adjustment factors used in software size estimation methods, most notably Function Point Analysis (FPA), to account for the technical and non-functional complexities of a project.

Their primary purpose is to adjust the initial size estimate (the Unadjusted Function Point Count, or UFC) based on the specific technological environment and required system characteristics.

Table 5.3: Technical Factors (GSC) and Factor Points

Technical Factors		**Factor Points**
F1	Reliable backup and recovery	4
F2	Data communications	5
F3	Distributed functions	3
F4	Performance	2
F5	Heavily used configuration	3
F6	Online data entry	5
F7	Operational ease	1
F8	Online update	5
F9	Complex interface	5
F10	Complex processing	5
F11	Reusability	4
F12	Installation ease	4
F13	Multiple sites	5
F14	Facilitate	4
	Total Sum (TCF):	**55**

In the table, shows the total summation of all factor points. The equation for calculating the technical complexity factor (TCF):

$$\text{TCF} = 0.65 + (0.01 \times \sum_{i=1}^{14} F_i) \tag{5.1}$$

As we calculated sum of all factors to be 55, we can calculate the TCF by the equation. So, TCF is 0. 65 + (0. 01 × 55) = 1. 20.

5.5.3 Counting Function Point (FP)

To calculate Function Point we need UFC and TCF and in previous sections we find:

- The calculated Unadjusted Function Points (UFC) for BookBridge is **1059**.
- TCF is $0.\ 65 + (0.\ 01 \times 55) = 1.\ 20$.

Therefore, FP = 1059 * 1.20 = 1270.8.

Chapter 6

COCOMO Estimation

The Constructive Cost Model (COCOMO) is an essential tool utilized for every software development effort. For the Book Bridge project, COCOMO estimation will be performed to accurately quantify the necessary effort, time, and budgetary cost. These calculations are critical, as they directly inform project planning, optimize resource allocation, and enable robust budget management. Furthermore, the model facilitates early risk identification and provides a quantitative basis for evaluating and comparing various project development alternatives.

6.1 What is COCOMO?

COCOMO stands for Cnnstructive Cost Model.It is a procedural software cost estimation model, widely used in software engineering, that helps to predict the effort, time, and cost required for a software development project.The model was introduced by Dr. Barry Boehm in 1981 (COCOMO 81) and later evolved into COCOMO II. It is based on empirical data from various historical projects

6.2 Types of COCOMO

The Constructive Cost Model (COCOMO) classifies software projects into three development modes based on complexity, team experience, and constraints. This classification determines the specific constants used in the estimation formulas.

- **Organic Project**

- **Semi-detached Project**
- **Embedded Project**

  1. **Organic Project**

     These projects are characterized by simplicity and familiarity. They are generally small-scale with well-understood requirements and are developed by small teams who possess strong domain knowledge and prior experience with similar systems. Requirements are typically few and flexible.

     Example: Simple inventory management systems, basic data processing systems.

  2. **Semi-detached Project**

     This mode represents an intermediate level of complexity, size, and team experience. The project may involve a mix of experienced and inexperienced personnel, and some requirements might be rigid while others are flexible. These projects often involve integrating commercial off-the-shelf (COTS) components or developing moderately complex systems.

     Example: Developing new compilers or utility software, small to medium database management systems.

  3. **Embedded Project**

     These are the most complex projects, defined by strict, inflexible requirements and significant constraints, often involving tight coupling with hardware, regulations, or complex operational procedures. They require large, highly experienced teams and are typically innovative or resource-constrained.

     Example: Air traffic control systems, real-time vehicle diagnostics, or industrial process control software.

6.2.1 Basic COCOMO

This is the simplest and quickest estimation method. It relies almost entirely on the estimated size of the software, measured in Kilo Lines of Code (KLOC), as the primary input. It uses fixed, empirical constants $(\alpha, \beta, \gamma, \delta)$ determined by the project mode (Organic, Semi-detached, or Embedded) to provide a rough, early-stage estimate of effort (in person-months) and development schedule.

6.2.2 Intermediate COCOMO

This model builds on the Basic version by introducing 15 multiplicative Cost Drivers (or Effort Multipliers) to refine the calculation. These drivers systematically account for factors that impact productivity, such as product reliability, database size, personnel capabilities, and the use of modern tools. This yields a significantly more accurate estimate than the basic model.

6.2.3 Detailed COCOMO

This is the most comprehensive model, designed for high-fidelity estimates. It applies the Cost Drivers across individual phases of the software development lifecycle (e.g., requirements, design, coding, testing) rather than applying a single multiplier to the entire project. This granularity is essential for large, high-risk projects where detailed phase-by-phase control is required.

6.3 COCOMO Calculation for Book Bridge

Given that the Book Bridge project is classified as an Organic Project and operates under time constraints, we will initially perform the Basic COCOMO estimation. This choice is deliberate because the Basic model offers a quick, transparent, and manageable way to derive a rough-order-of-magnitude estimate. Its reliance on KLOC is appropriate for a web application project and provides the necessary initial metrics to inform critical project decisions without the overhead of detailed cost driver analysis.

The basic COCOMO model calculates four parameters to estimate the cost of a project. The four parameters are:

- **Effort (E)**
- **Development Time (D)**
- **Staff Size**
- **Productivity**

In basic COCOMO, the estimation is done quickly and roughly as it relies only on some constant parameters and kilo lines of code (KLOC). The constant parameters are derived from empirical data and represent the average values observed for different types of projects. These constants can vary depending on the type of project, for example, organic, semi-detached or embedded project.

Below is a table that shows the constant values for the three types of projects:

Table 6.1: Constant Values for different project types to calculate COCOMO

Project Type	**a**	**b**	**c**	**d**
Organic	2.40	1.05	2.50	0.38
Semi-detached	3.00	1.12	2.50	0.35
Embedded	3.60	1.20	2.50	0.32

Here, a, b, c, and d are constants specified for different types of projects. As **Book Bridge** is an **organic project**, the constant values specified for organic projects are:

$$a = 2.40, \quad b = 1.05, \quad c = 2.50, \quad d = 0.38$$

These values will be used in the COCOMO calculation.

Previously, we calculated the **Function Point** to be:

$$\text{Function Point} = 1270.8$$

As Book Bridge is a web-based application, the codebase is written in JavaScript assuming 30 LOC/FP

Therefore, for Book Bridge the lines of code (LOC) are:

$$30 \times 1270.8 = 38124$$

or 38.124 kilo lines of code (KLOC).

In Summary:

$$\text{Constant Values: } a = 2.40; \quad b = 1.05; \quad c = 2.50; \quad d = 0.38$$

$$\text{KLOC value: } KLOC = 38.124$$

6.3.1 Effort Calculation

The total development effort required is calculated using the equation:

$$E = a \times (\text{KLOC})^b$$

Substituting the corresponding values:

$$E = 2.40 \times (38.124)^{1.05} = 109.766$$

Therefore, the estimated effort for the Book Bridge project is approximately 109.766 person-months.

This means that the total human effort required to complete the project is equivalent to 109.766 months of work by one person.

6.3.2 Development Time

Development time estimates the overall duration needed to complete the project and is calculated using:

$$D = c \times (E)^d$$

Substituting the constants and effort value:

$$D = 2.50 \times (109.766)^{0.38} = 14.90$$

Hence, the estimated development time is approximately 14.90 months.

This represents the total calendar time needed to complete the project assuming optimal resource allocation.

6.3.3 People Required

The average number of personnel required for the project is determined by dividing the total effort by the development time:

$$S = \frac{E}{D}$$

$$S = \frac{109.766}{14.90} = 7.38$$

Therefore, approximately 7 to 8 team members will be needed on average to complete the project within the estimated timeframe.

6.3.4 Productivity

Productivity indicates how efficiently the team produces code, and it is calculated using:

$$P = \frac{\text{KLOC}}{E}$$

Substituting the values:

$$P = \frac{38.124}{109.766} = 0.3473$$

Thus, the productivity of the Book Bride project is approximately 0.3473 KLOC per person-month.

This means that, on average, each team member contributes around 347 lines of code per month when scaled to project size and total effort.

Chapter 7

Gantt Chart

7.1 Concept of Gantt Chart

A Gantt Chart is a project management tool used to illustrate a project schedule. It visually represents the start and end dates of tasks, their durations, and dependencies. Each task is displayed as a horizontal bar on a timeline, where the position and length of the bar reflect the task's start date, duration, and end date. It helps project managers track progress, allocate resources effectively, and ensure that tasks are completed in sequence. Dependencies such as Finish-to-Start (FS) or Start-to-Start (SS) indicate how one task's timeline is related to another. Overall, Gantt charts provide a clear visual overview of project timelines and milestones, supporting better decision-making and workflow coordination.

7.2 Gantt Chart Table

WBS ID	**Task Name**	**Duration**	**Predecessor**	**Start Time**	**End Time**
1.0	Project Management & Governance	8 weeks	-	Week 0	Week 8
1.1	Project Initiation	1 week	-	Week 1	Week 2
1.2	Project Planning	2 weeks	1.1	Week 2	Week 4

WBS ID	Task Name	Duration	Predecessor	Start Time	End Time
1.3	Monitoring & Control	3 weeks	1.2	Week 4	Week 7
1.4	Project Close & Handover	1 week	1.3	Week 7	Week 8
2.0	Requirement & Analysis	4 weeks	-	Week 8	Week 12
2.1	Requirements Elicitation	2 weeks	1.2	Week 8	Week 10
2.2	Non-Functional Requirements	1 week	2.1	Week 10	Week 11
2.3	Acceptance Criteria & Test Cases	1 week	2.2	Week 11	Week 12
3.0	UX/UI Prototyping	7 weeks	-	Week 12	Week 19
3.1	Information Architecture	1 week	2.3	Week 12	Week 13
3.2	Wireframes & Prototypes	3 weeks	3.1	Week 13	Week 16
3.3	Usability Testing	2 weeks	3.2	Week 16	Week 18
3.4	Style Guide & Assets	1 week	3.2	Week 18	Week 19
4.0	Core Features	10 weeks	3.0	Week 19	Week 29
4.1	Sign-up & Verification	2 weeks	3.0	Week 19	Week 21
4.2	Sharing & Requesting Books	2 weeks	4.1	Week 21	Week 23
4.3	Book Wallet	2 weeks	4.2	Week 23	Week 25
4.4	Litera Club Management	2 weeks	4.3	Week 25	Week 27
4.5	Search & Filters	1 week	4.4	Week 27	Week 28

WBS ID	**Task Name**	**Duration**	**Predecessor**	**Start Time**	**End Time**
4.6	Review & Ratings	1 week	4.5	Week 28	Week 29
5.0	Integration & External Services	3 weeks	-	Week 29	Week 32
5.1	Payment API	3 weeks	4.6	Week 29	Week 32
6.0	Quality Assurance & Testing	4 weeks	4.0, 5.0	Week 32	Week 36
6.1	Component Test	3 weeks	5.1	Week 32	Week 35

Table 7.1: Project Gantt Chart Table

7.3 Gantt Chart Diagram

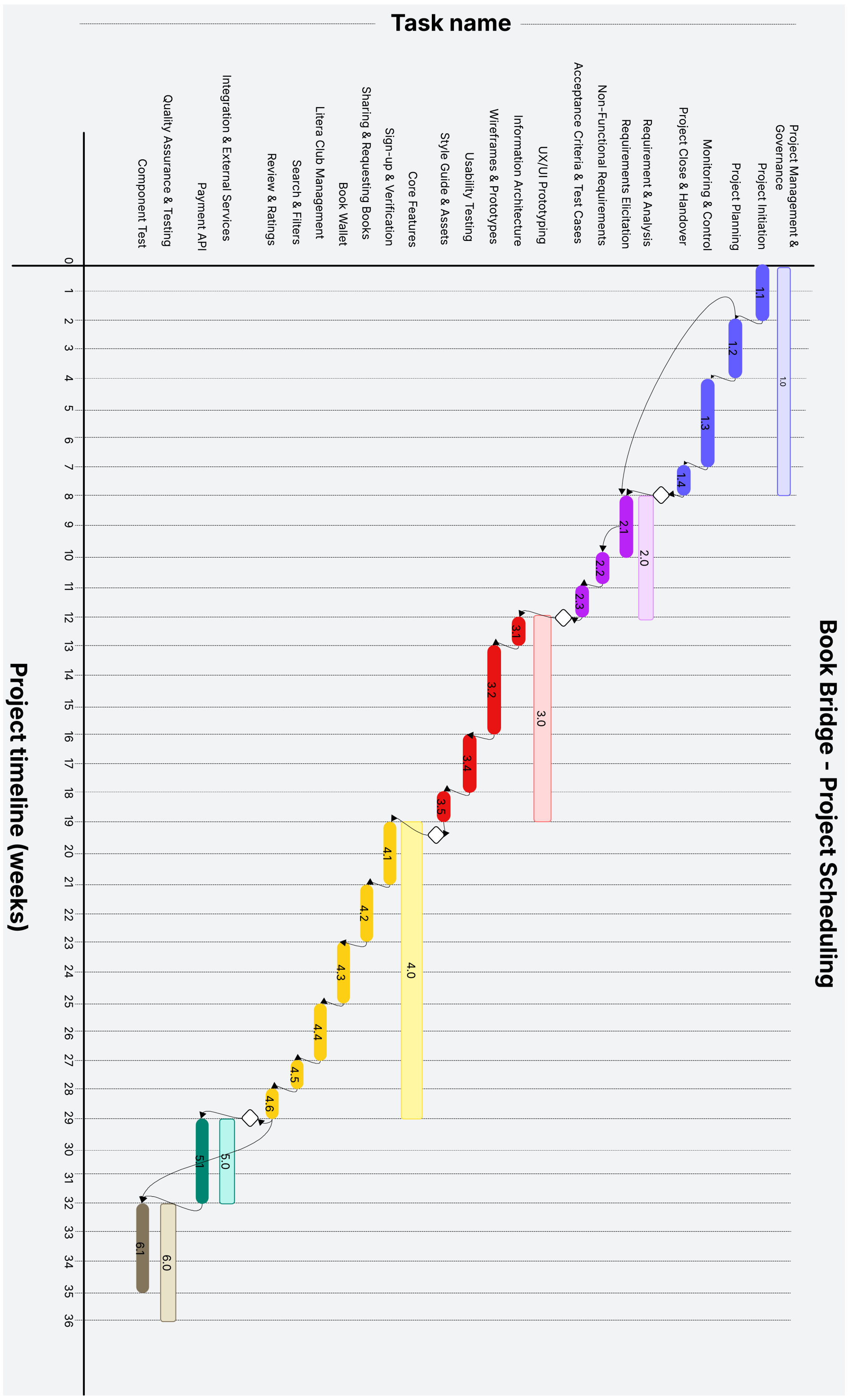

Figure 7.1: Gantt Chart

7.4 Assumption & Consideration

7.4.1 Assumptions

- **Stakeholder Availability:** Key stakeholders will be available for continuous feedback and sprint reviews, which is critical for the Agile methodology.
- **Resource Allocation:** All planned technical and human resources will be available as scheduled throughout the 36-week project timeline.
- **Stable Core Vision:** While requirements may evolve, the core vision of the "Book Bridge" platform as a community-sharing ecosystem will remain constant.

7.4.2 Considerations

- **Fixed Timeline:** The project is strictly constrained to a 36-week delivery schedule. All planning and execution must adhere to this timeline.
- **Scope Management:** The project is limited to the features defined as "In-Scope." Any proposed additions, such as AI-based recommendations, are "Out-of-Scope" and would require a formal change request.
- **External Dependency:** The project's timeline and "Book Wallet" functionality are critically dependent on the successful and timely integration of the third-party Payment API.

Chapter 8

Conclusion

The BookBridge project demonstrates a strategic fusion of agile development, community engagement, and sustainable design. By adopting iterative methodologies and stakeholder-driven planning, the team delivered a platform that facilitates equitable book sharing, fosters trust, and promotes lifelong learning.

Through structured project management practices—including Work Breakdown Structure, Wideband Delphi Estimation, Function Point Analysis, and COCOMO modeling—the team ensured accurate forecasting, risk mitigation, and quality assurance. The platform's modular architecture supports future scalability, making it adaptable for broader educational and urban contexts.

BookBridge exemplifies how interdisciplinary collaboration and human-centered design can yield scalable, socially responsible solutions. It not only meets current needs but also lays the foundation for future innovations in digital resource sharing and community-driven platforms.