KING'S College LONDON

Faculty of Natural, Mathematical, and Engineering Sciences

Department of Informatics

Software Engineering Group Project

5CCS2SEG Major Group Project

KCLMGP

Andrew Brown

Joshua Francis

Nour Hadhoud

Ziming Liu

Doris Nezaj

Fasih-Ur Rahman

Hugo Potier de la Morandiere

Scarlett Smith

Ivan Tachev

March 27, 2025

Contents

Chapter 1

Introduction

Our project aims to develop a dynamic and interactive habit tracking application that allows users to set and monitor personal and professional goals efficiently. By leveraging automation and data insights, the app will provide a seamless experience that minimises manual input, making habit tracking smarter and more intuitive. Users will also be able to gain valuable insights into their progress through clear visualisations (such as graphs and reports), helping them stay motivated and accountable.

The application is designed as a cross-platform system, primarily for iOS users while also being accessible via the web. The app has been developed using React Native with Expo Go, utilising TypeScript and JavaScript, and backend by a MySQL database. By combining user activity tracking and geo-information, the app is able to intelligently record activities.

Chapter 2

Objectives and stakeholders

2.1 Project objectives

The main objective of this project is to create a flexible, user-friendly habit and activity tracking app that helps people monitor and improve their productivity in a simple, yet customisable way.

The value our system provides to the stakeholders includes:

- **Better Productivity:** Users can easily track their habits and activities without too much manual input, helping them stay on top of their goals.
- **Smart Tracking:** Using location data, mobile activity, app usage, and AI, the app will automatically recognise what users are doing based on the information provided.
- **Customisable Habit Tracking:** Users can define habits in their own way, making the system more flexible and suited to different productivity styles. Most habit trackers enforce strict schedules, marking habits as "failed" if missed. Our app offers a flexible approach, letting users set goals like "exercise 3 times a week," dynamically adjusting streaks and sending only a weekly reminder to reduce pressure.
- **Privacy and Control:** Users will have full control over their data, with options to grant permissions, delete data, and choose which apps are tracked.
- **Data Insights:** The app will offer user-friendly, customisable graphs and reports to help users analyse their progress and export this data if needed. This allows users to assess their progress over a period of time, which can be a powerful tool to see where improvements towards their goals can be made in the future.

2.2 Stakeholder

2.2.1 Stakeholder analysis

We have identified each stakeholder group, ensuring that our project delivers meaningful value while also addressing the potential risks and challenges associated with it.

Using the D.A.N.C.E. framework, the key stakeholder groups include:

- **Decisions:** Stakeholders who decide constraints, shape the project direction, and provide guidance. This includes academic supervisors and the development team (university students).
- **Authority:** Stakeholders who must grant permission for the project. This includes academic supervisors who oversee the project, university administration that imposes policies on student projects, App Store regulators for deployment, and data protection authorities for handling user data.
- **Need:** Stakeholders impacted by the system. This includes the primary users such as students, professionals, and individuals most likely above the age of 13 looking to improve their productivity.
- **Connections:** People who can remove obstacles or enable success. This includes the development team, more specifically, front-end developers using TypeScript and JavaScript, and back-end developers managing the MySQL database.
- **Energy:** Stakeholders who influence enthusiasm or skepticism for the project. These include the development team, test users, users, and academic supervisors.

2.2.2 Key stakeholders

The key stakeholders are those who determine how successful this project has been. By identifying them, we can make sure that each group's needs and contraints are considered, allowing for a more effective development process.

Key stakeholders we have identified:

- **Primary Users:** The individuals who will interact with the application daily to track their habits and productivity. They include students, professionals, and individuals above the age of 13 looking to improve their time management and personal growth.
- **Development Team:** The university students responsible for planning, designing, building, and deploying the software. We must ensure the app is functional, includes thorough testing, and is compliant with academic, ethical, and legal standards.
- **Academic Supervisors and Project Proposers:** Professors and researchers who initiated the project and will provide mentorship, assessment, and guidance throughout the development process.
- **Privacy Regulators:** Given the application's reliance on personal data, privacy regulators will play a role in shaping how data is collected, stored, and managed.
- **Technology Providers:** Third-party service providers that allow us to develop and deploy the application. This includes React Native and Expo Go for mobile and web development, MySQL database hosting services, security services for data encryption, user authentication, and privacy compliance, and finally, technologies that allow user activity tracking (for example, using location-based data).

Chapter 3

Specifications

3.1 Functional specifications

The following features have been developed and are accessible in the deployed application. Each contributes directly to the project's objectives of creating a flexible, user-friendly application that can enhance the user's productivity.

3.1.1 Habit Creation and Management

Creating New Habits:

- **Description:** Users can create new habits with customisable details including name, description, type (build or quit), colour, schedule options, and optional goals.
- **Contribution to Objectives:** This is the core functionality that allows users to define specific behaviours they want to track, making the app personalized and adaptable to diverse habit-building needs.
- **Location:** On the habits page, which is the fist page that is shown on login, there is an add habit button which which takes you to a form. This form is used for the user to customise that habit that they are creating.

Habit Types (Build vs Quit):

- **Description:** The app distinguishes between habits to build (positive behaviours to adopt) and habits to quit (negative behaviours to eliminate).
- **Contribution to Objectives:** This differentiation supports different behaviour change strategies and accommodates customisation of habits for the user.
- **Location:** On the habit page the user will be able to tell the difference between a quitting habit and a building habit. When the user is creating a habit using the form at the top of the form they will choose between a build or quit habit.

Habit Customisation:

- **Description:** Users can customise habits with colour-coding, descriptions, and personal tracking parameters.

- **Contribution to Objectives:** Customisation enhances user engagement by allowing personal expression and visual organization of different habits.
- **Location:** All customisation of the habits can be done when the user presses the add habit button, which opens the form for the user to create their habit.

Editing Existing Habits:

- **Description:** Users can modify all aspects of their habits after creation, including schedules, goals, and appearance.
- **Contribution to Objectives:** Allows flexibility as users refine their habits and tracking needs over time, preventing abandonment when circumstances change.
- **Location:** In the top right corner of every habit there is a little pencil icon. When it is pressed a form is open with pre-filled in data which the user can change to edit the habit to their liking.

3.1.2 Scheduling and Frequency

Interval-Based Scheduling:

- **Description:** Users can set habits to repeat at regular intervals (every X days).
- **Contribution to Objectives:** Supports habits that don't follow a weekly pattern and accommodates varying frequency needs.
- **Location:** In the add habit and edit habit forms the user can choose to schedule the habit at regular intervals. So they enter a number for the habit to show every X days.

Weekly Day-Based Scheduling

- **Description:** Users can schedule habits for specific days of the week (e.g., Monday, Wednesday, Friday).
- **Contribution to Objectives:** Accommodates routines that follow weekly patterns, supporting integration with typical work/life schedules.
- **Location:** In the add habit and edit habit forms the user can choose to schedule the habit by selecting days of the week. The user will highlight the days of the week they want the habit to appear.

Instance Generation

- **Description:** The system automatically generates future instances of habits based on their scheduling patterns.
- **Contribution to Objectives:** Ensures users always have upcoming habits ready to track without manual creation.
- **Location:** Using the calendar at the top of the habits page, the user can select future days of the week to see the habit they will have to complete. The user can also check the habits on dates that are on future weeks.

3.1.3 Goal Setting and Tracking

Numeric Goal Setting

- **Description:** For "build" habits, users can set specific numeric goals with custom units (e.g., read 30 pages, meditate 20 minutes).
- **Contribution to Objectives:** Provides clear, measurable targets that make progress tracking more concrete and satisfying.
- **Location:** Clicking on the habit will allow the user to enter how much progress they have made so far on the habit.

Binary Completion Tracking

- **Description:** For "quit" habits, users track success with yes/no completion options.
- **Contribution to Objectives:** Simplifies tracking for avoidance-based habits where the goal is simply not engaging in a behaviour.
- **Location:** Clicking on the habit will allow the user to enter if they have completed the habit or not.

Progress Recording

- **Description:** Users can record daily progress toward habit goals including partial completion.
- **Contribution to Objectives:** Encourages consistent habit engagement by acknowledging incremental progress rather than all-or-nothing success.
- **Location:** Clicking on the habit will allow the user to enter how much progress they have made so far on the habit.

Habit Streak Tracking

- **Description:** Each habit has a streak of how many days in a row the user has fully completed the habit.
- **Contribution to Objectives:** This encourages the user to keep up with their habits as the larger their streak goes, the harder the user will try to keep it going.
- **Location:** The streak is shown by a fire icon, followed by a number (the number of days the streak is on). It's found on every habit next to the edit habit button.

3.1.4 Visualisation and Statistics

Calendar View

- **Description:** Provides a monthly calendar visualization showing total habit completion of each day for the month.
- **Contribution to Objectives:** Gives users a holistic view of their habit consistency, helping identify patterns and trends.

- **Location:** This can be found on the calendar page which can be accessed using the menu at the bottom of the screen.

Average Completion Percentage

- **Description:** Shows the average completion percentage of each day for the whole month
- **Contribution to Objectives:** Provides a quick measure of overall habit success to allow the user to track their progress.
- **Location:** There is a circle at the bottom of the calendar on the calendar page that shows the average completion percentage for that month.

Daily Streak Tracking

- **Description:** Records and displays the current and longest streak of consecutive days with all the habit fully completed.
- **Contribution to Objectives:** This encourages the user to keep up with their habits as the larger their streak goes, the harder the user will try to keep it going.
- **Location:** The streaks are found in between the monthly calendar and the average completion percentage circle, on the calendar page.

3.1.5 Date Navigation and Filtering

Weekly Calendar Navigation

- **Description:** A scrollable weekly calendar view allowing selection of specific dates to view and manage habits
- **Contribution to Objectives:** Provides intuitive temporal navigation for managing habits on different days.
- **Location:** On the habits page at the top of the screen.

Date-Specific Habit Filtering

- **Description:** Displays only relevant habits for the selected date based on scheduling rules.
- **Contribution to Objectives:** Reduces cognitive load by showing only what's relevant for the current day or selected date.
- **Location:** By using the weekly calendar on the habits page the habits that are displays will only be relevant to the date selected.

3.1.6 Data Persistence and Synchronization

User Authentication

- **Description:** Secure user account system with email-based authentication.

- **Contribution to Objectives:** Protects user data and enables personalized habit tracking.
- **Location:** The user can create an account using the sign in page. If they already have an account they can use the login page to login to their account.

3.1.7 User Interface Features

Theme Support

- **Description:** Application supports light and dark themes with appropriate colour schemes.
- **Contribution to Objectives:** Enhances user experience and accessibility in different lighting conditions and according to user preference.
- **Location:** The user can switch between light and dark mode using a selector that can be found in the settings page.

Responsive Design

- **Description:** UI adapts to different screen sizes and orientations.
- **Contribution to Objectives:** Ensures a consistent experience across different devices.
- **Location:** This is found through out the whole app.

3.2 Non-functional specifications

3.2.1 Security

Password Hashing

- **Description:** User passwords are securely hashed before storage.
- **Contribution to Objectives:** Protects user credentials even in case of a data breach.
- **Location:** All of the hashing is done on the client side so only the hashed password gets stored on the server side.

3.2.2 Reliability

Error Handling

- **Description:** Comprehensive error handling for all API calls and user interactions.
- **Contribution to Objectives:** Ensures graceful failure and prevents app crashes when operations fail.
- **Location:** Can be found throughout the app.

3.2.3 Usability

Intuitive UI Elements

- **Description:** User interface designed with intuitive controls and visual feedback.
- **Contribution to Objectives:** Reduces learning curve and improves user experience.
- **Location:** Can be found throughout the app.

Form Validation

- **Description:** Input validation for all user-submitted data.
- **Contribution to Objectives:** Prevents errors from invalid data and provides immediate feedback.
- **Location:** Can be found throughout the app (e.g., logging in or signing in, creating or editing a habit, inputting habit progress, etc.)

3.2.4 Maintainability

Modular Architecture

- **Description:** Code organized into reusable, single-responsibility components.
- **Contribution to Objectives:** Simplifies maintenance and enables easier feature additions or modifications.
- **Location:** Found throughout all the files in the code.

Typescript Implementation

- **Description:** Strong typing throughout the application using Typescript.
- **Contribution to Objectives:** Catches type-related errors at compile time and improves code documentation.
- **Location:** Used throughout the code.

3.2.5 Compatibility

Cross-platform Support

- **Description:** Implemented using React Native to work across different device types.
- **Contribution to Objectives:** Provides consistent user experience on iOS, Android, and potentially web platforms.
- **Location:** React Native components are used throughout the code.

3.2.6 Data Management

Efficient Data Storage

- **Description:** Optimized database schema with appropriate relationships and indexing.
- **Contribution to Objectives:** Minimizes storage requirements while maintaining data integrity.
- **Location:** Found in the server.js file in the project code.

Offline Capability

- **Description:** Progress data and account details for the use is saved even when the app is closed.
- **Contribution to Objectives:** Allows user to access account on different devices
- **Location:** Data is saved on a cloud database.

Chapter 4

Project management

4.1 Project Overview and Scope

During initiation, we formally agreed on the project scope as captured in our scope statement. The primary outcome of this phase was to gain clarity on what our application would and would not include. We found this step highly effective at allowing us to understand the true value we want to provide, which allowed us to implement features that are coherent and make sense. Identifying stakeholders early was also key to gain clarity on who we are providing value to. Below are some of the key decisions made during our first two project initiation meetings in week 1.

Project Purpose and Description

A mobile-web hybrid application that provides a flexible way for users to track their habits. The app will help users enforce new good habits and assist with ending bad habits. By offering clear data visualisations, a user-friendly interface, and customisable habits, the system helps users manage their goals.

Desired Results and Exclusions

In Scope:

- Building a habit and activity tracking app using React Native with Expo Go, TypeScript/JavaScript, and a MySQL database.
- Allowing users to register/log in, set custom habit goals, and track progress.
- Providing clear data visualisations (charts, graphs, analytics) and a data export feature.

Exclusions:

- Frequent reminders or push notifications are out of scope; notifications will be weekly.
- Direct integration with third-party apps/services.

Constraints

Time: The project must be completed within a 10-week window (13/01/25 – 27/03/25).
Team Size: There are 9 group members, requiring effective communication to manage coordination.
Skill Sets: The team's familiarity with React Native, Expo Go, TypeScript, and MySQL may limit feature complexity.
Workload Balance: The team must juggle other academic responsibilities alongside this development project.

4.1.1 Stakeholder Identification

We used the D.A.N.C.E. framework to classify stakeholders, as detailed in Chapter 2 of this report. At this stage, we focused on:

- **Primary Users** - those who directly interact with the app.
- **Development Team** - responsible for planning, designing, building, and deployment.
- **Academic Supervisors** - providing guidance and evaluation.

4.1.2 Our Project Management Approach

Initially, we agreed to combine Agile techniques (e.g., using Trello with story points) with a more traditional Waterfall Model (e.g., a fixed final deadline, staged deliverables). This hybrid approach provided us with:

- Flexibility to reprioritise features when faced with new insights or changing team availability.
- Structure to ensure mandatory deliverables (e.g., report chapters, final submission) meet set deadlines.

4.2 Planning

Planning was critical in laying out how we would achieve the objectives defined in our scope statement. As recommended by project management theory [1], we focused on risk management, scheduling, and communication planning.

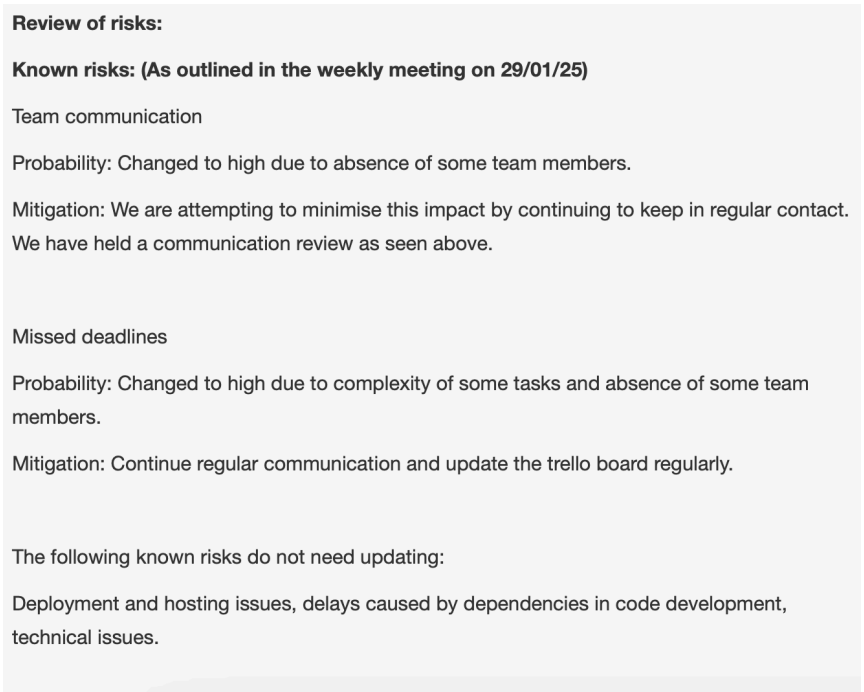
4.2.1 Risk Management

We identified potential risks early on and agreed on their probability and impact. Identifying risks early allowed us to understand how to quickly minimise problems when they would inevitably arise, helping us waste less time. For example, as seen in Table 4.1., one risk we identified was "Missed deadlines" which led us to understand the importance of setting weekly task deadlines in our project.

Table 4.1: Project Risk Management (source: Team Feedback Meeting Minutes 29/01/25)

Risk	**Probability**	**Impact**	**Mitigation**
Poor team communication	Medium	High	Establish a weekly meeting date (Wednesdays) and communicate regularly using Discord.
Missed deadlines	Medium	High	Use Trello regularly, assign deadlines to each task, and keep the group updated.
Deployment and Hosting issues	Low	High	Choose a reliable hosting provider. Test deployment.
Delays caused by dependencies in code development	Medium	High	Assign tasks thoughtfully and keep the group updated with tasks.
Technical issues	Medium	High	Make sure to test regularly. Regular communication with the team.

We also re-evaluated risks during the project review meeting, which was effective at helping us identify where we were falling behind as a team, such as addressing security issues (see Figure 4.1). (See Section 4.3 for a further evaluation of our project review meeting).



Review of risks:

Known risks: (As outlined in the weekly meeting on 29/01/25)

Team communication

Probability: Changed to high due to absence of some team members.

Mitigation: We are attempting to minimise this impact by continuing to keep in regular contact. We have held a communication review as seen above.

Missed deadlines

Probability: Changed to high due to complexity of some tasks and absence of some team members.

Mitigation: Continue regular communication and update the trello board regularly.

The following known risks do not need updating:

Deployment and hosting issues, delays caused by dependencies in code development, technical issues.

Figure 4.1: Reevaluation of Risks (source: Team Feedback Meeting Minutes 26/02/25)

4.2.2 Scheduling

Our scheduling used Trello to break down epics into manageable tasks. We assigned story points on a scale of 1 to 5. This allowed us to:

- Estimate the project velocity (around 15 story points/week during our project review meeting).
- Track our backlog of tasks and ensure we were on track for submission by the deadline.

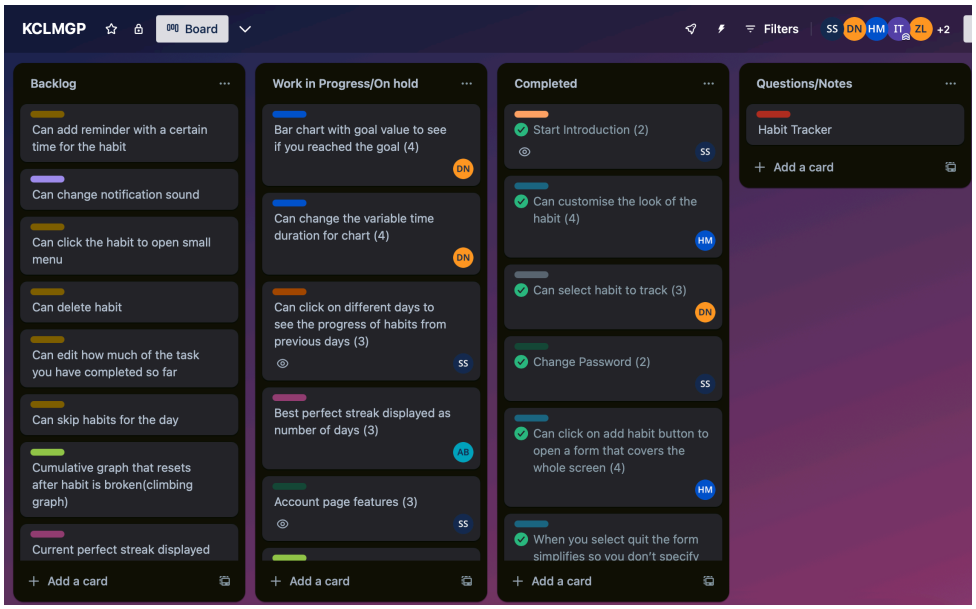


Figure 4.2: Screenshot of Trello board showing tasks and progress

As seen in Figure 4.2, something we would improve in the future is to include a 'Testing' section on our Trello board. This would have ensured that all tasks marked as 'complete' were thoroughly tested beforehand, preventing bugs and the need for retrospective testing.

One aspect that we agreed worked well was using colour-coded labels for each task (see Figure 4.2), which allowed us to separate different areas of our app. This helped organise task allocation and reduced dependencies in code development, which addressed one of our 'High' impact risks (see Table 4.1).

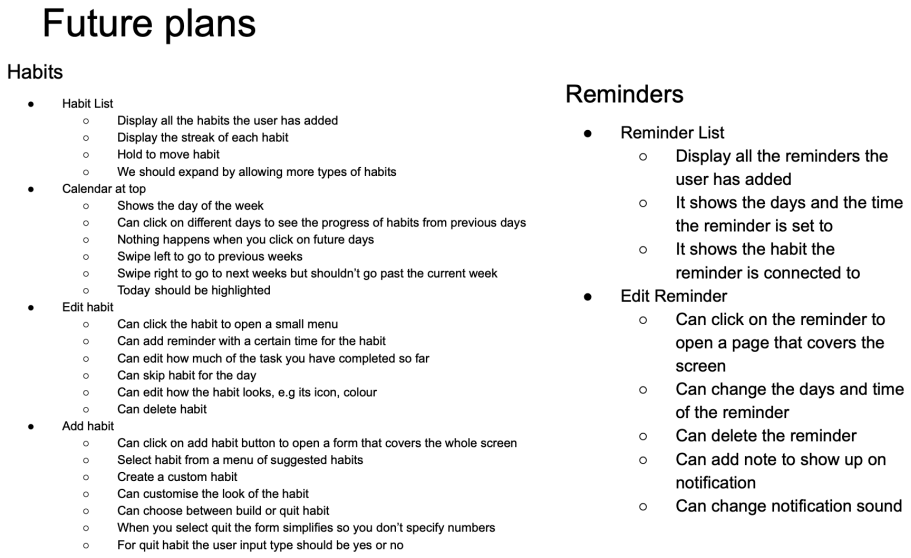


Figure 4.3: Page 1 of our backlog of organised tasks defined during the Week 2 Planning meeting (28/01/25)

4.2.3 Communication Planning

We formally agreed on the following communication strategies (as recorded in Team Feedback Meeting Minutes 29/01/25):

- **Weekly Team Meetings:** Every Wednesday, in FWB Library.
  - Discuss tasks completed, tasks in progress, and next steps.
- **Online Communication (Discord):** For quick updates, online meetings, and working together on tasks.
- **Team Feedback & GitHub Integration:** We used Team Feedback to share commits and attribute coding work. Collaborative coding sessions were recorded to ensure proper credit distribution.

One aspect of communication that we found effective and would implement again is the use of different communication channels on Discord. This allowed us to organise and manage our online discussions effectively, ensuring all saved materials were in one place and weekly targets were clearly communicated (see Figure 4.4).

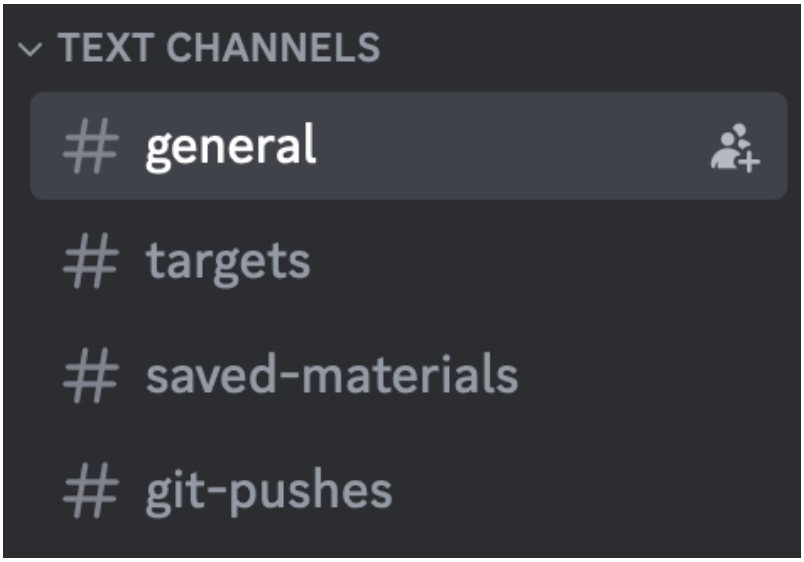


Figure 4.4: Different Discord communication channels used in the project

A critical reflection here is that, despite this plan, some members disengaged, which caused delays in certain tasks (see Section 4.2.2). This demonstrates how initial communication plans can be tested under real project conditions.

4.3 Execution

Execution involved turning our plans into tangible deliverables—writing code, creating the database structure, designing interfaces, and more.

4.3.1 Task Allocation and Progress Tracking

We adopted a weekly task allocation model at the start of each Wednesday meeting. For instance:

- **Hugo:** Responsible for integrating new habit data into the MySQL database.
- **Doris:** Responsible for the statistics-page database functionalities.

Each person reported on progress the following week, as recorded in meeting minutes. If tasks were incomplete, we reassigned them or extended deadlines.

For evidence of this approach, see the weekly minutes excerpt below:

> **Excerpt 4.1.** Individual accountability of task (Minutes 12/02/25)
> *"Action/commitment: Add selector to statistics page to pick different habits in the table.*
> *Person responsible: Ming*
> *Deadline: 12/02/25*
> *Progress: In progress, deadline extended to 26/02/25"*

This approach worked incredibly well for us, as it allowed us to keep track of individual progress, maintain accountability, and identify tasks requiring extra attention. In the future, we could enhance this method by including more detail in the 'Progress' section of our weekly minutes, requiring each member to specify exactly what was done. This would provide a deeper understanding of the development process.

4.3.2 Dealing with Disengagement

A challenge we faced was the partial disengagement of some team members, which manifested through missed meetings and uncompleted tasks. Drawing on adaptive development principles, we:

- We reassigned tasks when a member missed more than two consecutive meetings without prior notice.
  Example: As recorded in Meeting Minutes 26/02/25, under New Commitments: "Reassigned calendar page task to Andrew."
- We reduced the scope of certain features (e.g., scaled back the AI-based habit recommendation system).

4.3.3 Ensuring Code Quality

Initially, we planned code inspections and peer reviews for each major functionality. However, in practice:

- We relied heavily on automated testing instead of thorough peer reviews.
- Due to time constraints, we conducted fewer formal inspections and prioritised feature completion.

In retrospect, while automated testing coverage was beneficial, face-to-face reviews could have caught inconsistencies earlier. This aligns with standard software engineering advice, which highlights that code reviews often catch design and logic flaws beyond what unit tests detect [2].

4.4 Monitoring and Control

Monitoring and control involved continuously checking our progress against the plan and adapting as needed.

4.4.1 Regular Check-ins and Task Status

Each weekly meeting served as a control point:

- We reviewed completed tasks vs. planned tasks.
- We updated the Trello board (e.g., moving items to "Done" or reassigning them).
- We merged completed branches to the master branch on GitHub together, preventing serious conflicts.

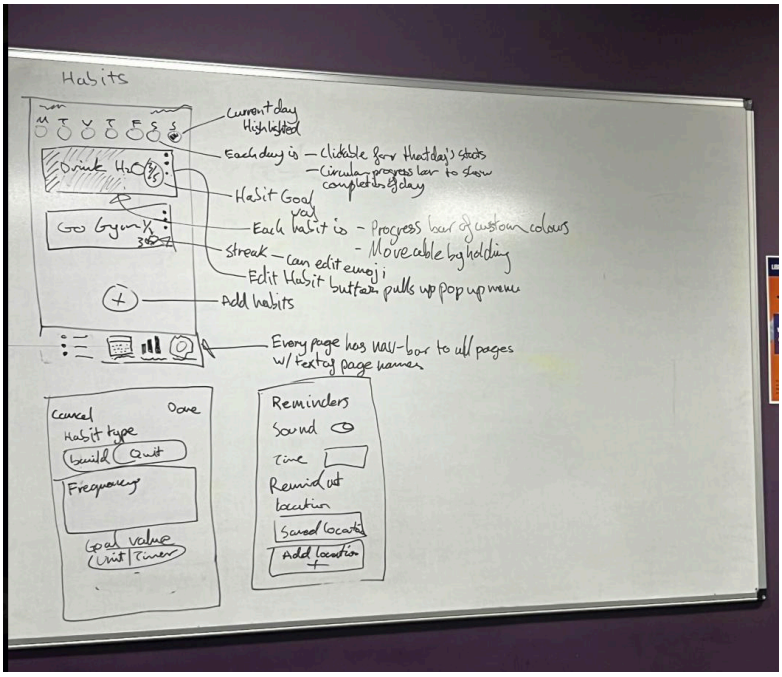


Figure 4.5: Designing the app during a weekly meeting

We conducted a project review meeting where we reassessed project risks, scheduling, and communication plans (Meeting Minutes 26/02/25). A key issue we identified was that we only held one project review meeting, which we believe was insufficient for a project of this duration. We have discovered the importance of regular reviews, as it allows us to maximise productivity by taking a look at our progress and challenges. Perhaps if more reviews were conducted, we may have caught arising issues (e.g. security risks) earlier on.

4.5 Close

Finally, closing the project required delivering the final software product and supporting materials. Whilst our official deadline is 27/03/25, we began planning for closure early to mitigate last-minute rush or errors.

4.5.1 Preparing Final Deliverables

- **Codebase:** Fully merged into the main branch on GitHub.
- **App Deployment:** Tested on Expo Go and verified basic web accessibility.
- **Test Deployments:** Conducted in a staging environment.

Even though we planned for early completion, we still felt slightly rushed as a team by the end. To prevent this problem in the future, we could have set our own deadline a week prior to submission, which would allow the last week to be focused solely on refinements/polishing the deliverables.

4.6 Conclusion

Overall, our hybrid Agile/Waterfall project management strategy provided us with structure and flexibility. Despite challenges, we successfully adapted by reallocating tasks, adjusting our scope, and focusing on core deliverables. Our experience shows that a structured and thoughtful project management approach, based on best practices and real-time adaptation, is key to delivering complex software projects.

Chapter 5

Design and implementation

5.1 Design and Implementation

This section discusses how our software is designed and implemented, focusing on meeting both functional requirements (such as creating and managing habits) and non-functional requirements (such as security, maintainability, and reliability). We outline the three-tier architecture and explain our rationale for each design choice, demonstrating how the client, server, and database work together.

5.1.1 Overall Architecture

Our system uses a three-tier structure:

- **Client (React Native and Expo)**: Displays user interfaces for activities like logging in, managing habits, and viewing calendars or statistics. We use Expo Router to organise screens.
- **Server (Node.js and Express)**: Provides REST endpoints (for example, /signup, /addHabit, /export/:email) to handle authentication, validate user data, and manage business logic.
- **Database (MySQL via mysql2/promise)**: Stores user credentials, habit definitions, scheduling information, and user progress records.

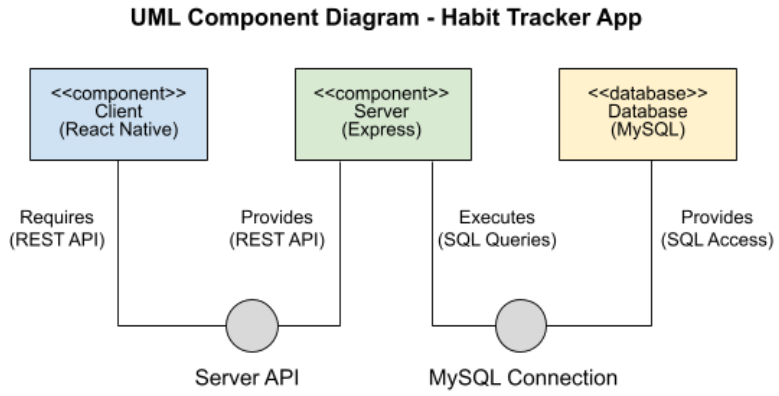


Figure 5.1: UML Component Diagram of our Habit-Tracker Application

Figure 5.1 illustrates these layers. By separating the UI (client), the application logic (server), and storage (database), each layer remains independently maintainable and secure.

5.1.2 Key Design Decisions

Cross-Platform Client Using Expo:

Choice and Reasoning

- We wanted a single codebase (TypeScript) for iOS, Android, and web accessibility, which is why we chose React Native and Expo.
- Expo simplifies building and testing on multiple platforms, preventing the need for a complicated setup. This allowed us to start developing features much sooner, rather than spending time on configuration.

Implementation Details

- File Structure:
  - src/app/(auth)/: Handles login and signup screens.
  - src/app/(protected)/tabs/: Contains the main habits, calendar, stats, and settings screens.
  - src/components/: Contains reusable UI elements such as habit modals, calendars, graphs, themed text, and shared styles for consistent design.
  - src/lib/client.ts: Centralises HTTP request logic to avoid repeating fetch calls across components.

  This modular structure improves code organisation, makes features easier to maintain, and encourages reuse of components across the app, as shown in the SOLID design principles [3].
- Styling and Theming:

- The app uses shared colours and styles so that all screens look consistent.
- Common layouts, buttons, and inputs are defined in reusable style files to avoid repeating code.
- Some components automatically change their appearance based on the selected theme (light or dark), making the app easier to use and nicer to look at.

Server with Node.js and Express:

Choice and Reasoning

- Node.js and Express were chosen because they are widely used, beginner-friendly, and well-supported by online documentation and learning resources.
- Express enables a simple and flexible REST API structure, making it easy to define endpoints that connect to application features like signing up, adding habits, or exporting data.

Implementation Details

- Express is used to define REST endpoints such as `/signup`, `/addHabit`, and `/export/:email`, which the client interacts with to manage user accounts and habits.
- Middleware like `cors` and `dotenv` help with handling requests, loading environment settings, and allowing the mobile app to connect to the server.
- Authentication uses JSON Web Tokens (JWT) to protect routes and make sure only logged-in users can access certain features.

MySQL Database:

Choice and Reasoning

- A relational database structure is ideal for managing related entities such as users, habits, and habit completion logs. MySQL allows us to enforce clear relationships and constraints through primary and foreign keys.

  For example, in the habit_progress table, a foreign key constraint on (user_email, habitName) references the habits table. This ensures that progress entries cannot exist without a matching habit, and enables automatic cleanup of related records when a habit is deleted, maintaining data integrity.
- We use the `mysql2/promise` library to write our own SQL commands directly (such as `CREATE TABLE` and `SELECT`) and run them using modern `async/await` syntax. This gives us full control over database operations, and keeps the code clean, readable, and easier to debug.

Schema Outline

Below is an overview of some of the main tables that make up our database schema.

- users: Stores user account information. The `email` field acts as the primary key and is used to identify users. Each user also has a hashed password and a username.

- habits: Stores details about each habit a user creates. Each habit is uniquely identified by a combination of `user_email` and `habitName`, which together form a composite primary key. Other fields include habit type (build or quit), goal settings, and scheduling options (e.g., interval or weekly).
- habit_progress: Logs daily progress for each habit. The composite primary key (`user_email`, `habitName`, `progressDate`) ensures that only one record exists per user, habit, and date. This table tracks goal progress, completion status, and streak count. A foreign key links back to the `habits` table, ensuring referential integrity.
- habit_days: Supports weekly habit scheduling. Each record maps a habit to a day of the week (e.g., Monday, Wednesday). This allows habits to repeat on specific days, and also uses a foreign key to maintain referential integrity with the `habits` table.

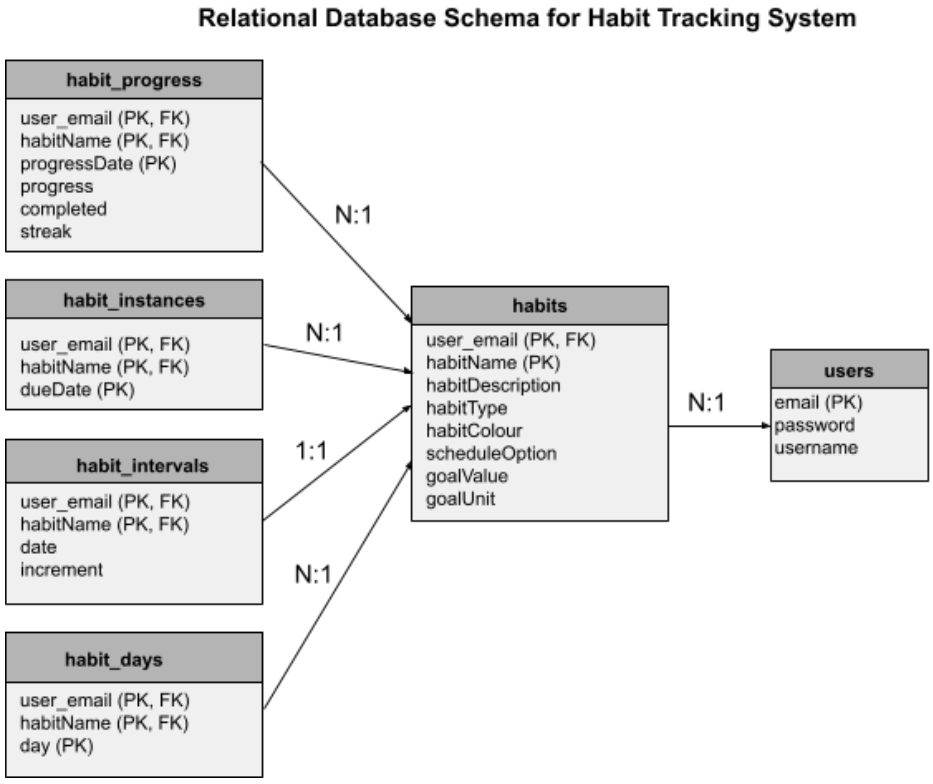


Figure 5.2: Relational Database Schema of our Habit Tracking System

Figure 5.2 shows the tables in the habit-tracking database and their relationships. Arrows represent foreign keys, and cardinality ratios (e.g., 1:1, N:1) indicate how records in one table relate to others, supporting data integrity and structure.

5.2 Meeting Requirements

This section summarises some examples of how our system design and implementation meet the project's key functional and non-functional requirements (as specified in Chapter

3 of this report). We focus on how our architecture, database structure, and code design support these goals (instead of just listing all the features).

5.2.1 Functional Requirements

- **Habit Creation and Management:** Users can create, customise, and update habits with flexible scheduling (interval-based or weekly), goals, and visual settings. These features are implemented through reusable client components and backed by structured database tables (`habits`, `habit_progress`, `habit_days`).
- **Progress Tracking and Visualisation:** The system calculates and displays daily progress, streaks, and completion trends via calendar views and graphs. These visual features reflect real-time data stored and retrieved from the backend.
- **Data Export:** A dedicated API endpoint (`/export/:email`) allows users to download their habit history in JSON format, so users can view and save a copy of their own data.

5.2.2 Non-Functional Requirements

- **Security:** All user passwords are hashed client-side before being sent to the server. Routes requiring authentication use JSON Web Tokens (JWT), and all data operations go through validated REST endpoints.
- **Maintainability:** The codebase is modular, with clear separation between client, server, and database logic. Client-side fetch logic is centralised, and components follow single-responsibility principles for easier reuse and testing.
- **Performance and Scalability:** We use SQL with `mysql2/promise` to make database queries fast and efficient. Composite keys and foreign key links help keep the data consistent and quick to access.
- **Data Integrity:** Relationships between tables are enforced through composite foreign keys. For example, `habit_progress` entries reference existing habits, and `ON DELETE CASCADE` ensures cleanup when habits are removed.

5.3 Alternative Approaches

Throughout development, we considered several alternative technologies and approaches:

- Using an ORM: ORMs can make database tasks easier, but we chose to write SQL ourselves using `mysql2/promise` to have more control over scheduling logic and table relationships like composite keys and foreign keys.
- Native iOS/Android Development: We considered building separate native apps for each platform, but chose React Native with Expo to keep one shared codebase. This made development easier and faster, while also allowing the app to run smoothly on iOS, Android, and the web.

5.4 Conclusion

Our design and implementation emphasise a three-tier architecture that separates the user interface, server-side logic, and database. The result is a cross-platform habit-tracking application that cleanly meets project needs for scheduling, progress tracking, data visualisation, and security measures. The system is also designed to be easily extended in the future, with potential additions like advanced analytics, social features, or smarter notifications.

Chapter 6

Testing

This chapter describes the testing approaches we used to ensure our habit-tracking application is reliable and working as expected. Throughout our development, we combined automated testing with targeted manual tests for cross-platform validation. While our automated coverage is high, our critical reflection reveals areas where we could have improved our quality assurance process.

6.1 Approach and tools

6.1.1 Automated Testing

We relied heavily on automated testing to make sure our features worked as expected. Each suite checks how the component renders, responds to user actions (e.g., button presses), and triggers any side effects. This approach helps us confirm that our UI logic, data calls, and component behaviours work as intended.

Tools Used

- **Jest**: Our main test runner, responsible for executing the test suites.
- **React Native Testing Library**: Lets us simulate user interactions (like tapping buttons or typing text).
- **Some examples of our Mocking Techniques**:
  - AsyncStorage: Replaces native storage calls so tests don't fail on missing native modules.
  - Fetch: Fakes server API calls, which means our tests do not depend on a real network.
  - Expo Router and ThemeContext: We replace real navigation and theming with simple "fake" versions. This way, our tests focus only on the component's behavior, rather than on actual screen transitions or styles.
  - Victory Charts: This library uses SVG (Scalable Vector Graphics) to draw charts. In our tests, we swap the real chart components for simpler "mocks." This means the charts don't actually get rendered, and our tests run faster

with fewer errors. For example, in `BuildHabitGraph.test.tsx`, we check that the chart component receives the right data without drawing any real SVG.

Location of Automated Tests

We keep our tests in the `client/src/components/__tests__` folder, and we name each file after the component it tests. For example:

- `BuildHabitGraph.test.tsx`
- `WeeklyCalendar.test.tsx`
- `SettingsPage.test.tsx`

This structure makes it easy to locate each test and ensures that test code remains closely tied to its corresponding component.

One critical reflection is that we did not integrate automated testing as thoroughly as we had hoped during development. Although we did test throughout the entire project, many tests were only added later in the project, which in hindsight left some functionalities untested for longer than we would have liked.

6.1.2 Black-Box Testing Techniques

Our automated tests followed key black-box testing principles. We focused on checking how the app responds to user actions and inputs, without relying on internal implementation details.

- **Boundary Value Analysis:** In `HabitPanel.test.tsx`, we tested date-related edge cases. For example, confirming that users cannot open the progress dialog for future dates. These tests helped ensure the app prevents invalid actions outside the allowed time range.
- **State Transition Testing:** In `ProgressEntry.test.tsx` checks how the UI responds to different habit types. For example, showing numeric input for habits with goals, and hiding input in read-only mode. These tests make sure the right controls appear for each state and that user actions are handled correctly.
- **Use Case Testing:** In `NewHabitModal.test.tsx`, we tested realistic user flows for adding or editing habits. For example, filling in details, switching schedule types, toggling goals, and picking days. These tests help confirm that habit setup works smoothly from start to finish.

By using these techniques, we ensured that key user interactions were tested thoroughly, even in the absence of external testers.

6.1.3 Manual Testing

While automated tests validate most of our components and logic, we supplemented them with manual testing to ensure the application feels correct when being used. Each major functionality (e.g., habit creation, editing, exporting data) was checked on iOS, Android, and the web).

Limits of Manual Testing
Although manual checks can uncover issues that automated tests might miss, such as subtle design issues or platform-specific bugs, this approach is slow and difficult to maintain for every new change, as Crispin and Gregory point out [4]. As a result, we focused our manual testing on critical features and cross-platform validation, rather than using it for every update.

6.1.4 Test Coverage Report

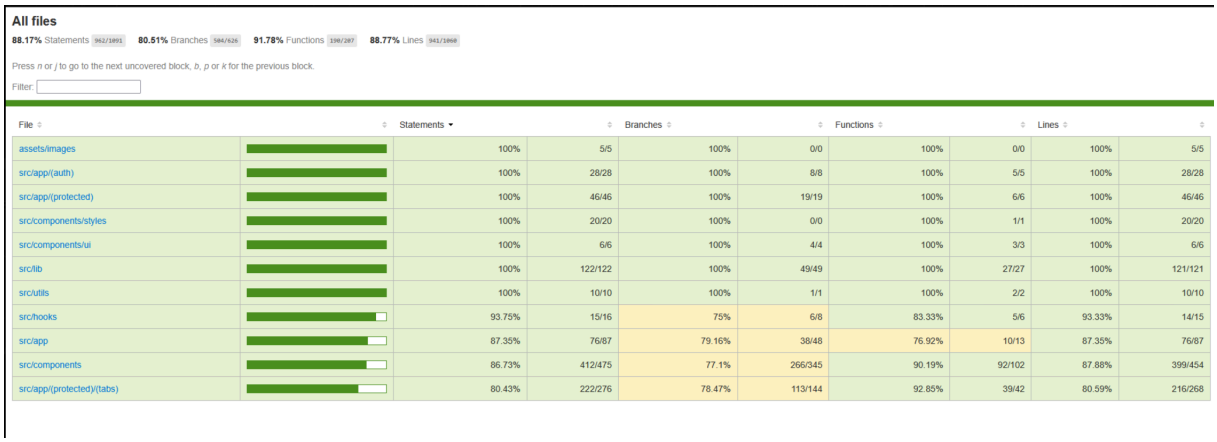


Figure 6.1: Test Coverage Report of our Habit-Tracker Application

Figure 6.1 shows our test coverage report. The source coverage details (HTML pages) are located in the project's coverage/lcov-report folder, particularly in the index.html file.

6.2 Evaluation of testing

6.2.1 Strengths

- **Extensive Automated Coverage**: Our Jest tests are well-integrated, ensuring we can trust most core functionalities. Mocking external dependencies (network, storage, theming) keeps our tests deterministic and easier to maintain.
- **UI Testing**: By using React Native Testing Library, we test not just logic but actual user interactions (tapping buttons, text input, etc.).

- **Cross-Platform Verification**: Manual checks on iOS, Android, and web prevent platform-specific issues and ensures a consistent experience.

6.2.2 Weaknesses and Reflections

- **Fewer Formal Inspections**: We relied heavily on automation due to limited time. This left less room for structured peer review or code inspection, which can catch architecture-level or readability issues that tests may miss.
- **Lack of External Black-Box Testing**: Although we occasionally showed demos to family/friends, we did not organise a broader group of external testers. This might have revealed usability issues or inconsistencies that a developer-focused approach could overlook.

6.2.3 Future Improvements

- **More Peer Reviews**: In future we would review each other's pull requests more often and thoroughly, which could reduce hidden defects early on.
- **Black-Box Testing:** Bringing in more people who haven't seen our code would provide fresh perspectives and more thorough user feedback.
- **Formalising Manual Test Cases**: Storing each manual test case (actions, expected results) in a shared document would add clarity and accountability for each test run.
- **Automated Testing during Development**: We would prioritise the need test more thoroughly throughout the entire project, making sure no functionalities were left untested for longer than necessary.

Overall, our mix of automated tests and manual checks gave us confidence in our software's correctness and stability. However, we recognise that further improvements could be made to our testing to strengthen our quality assurance.

Chapter 7

References

[1] Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. 6th. Newtown Square, PA: Project Management Institute, 2017. ISBN: 978-1628251845 (page 12).

[2] Karl E. Wiegers. *Peer Reviews in Software: A Practical Guide*. Addison-Wesley Professional, 2002. ISBN: 978-0201734850 (page 17).

[3] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Pearson, 2002. ISBN: 978-0135974445 (page 21).

[4] Lisa Crispin and Janet Gregory. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Professional, 2009. ISBN: 978-0321534460 (page 28).