System Verification and Validation Plan for Software Engineering

Team #16, The Chill Guys
Hamzah Rawasia
Sarim Zia
Aidan Froggatt
Swesan Pathmanathan
Burhanuddin Kharodawala

October 27, 2025

Revision History

Date	Developer	Notes
October 24th	Swesan Pathmanathan and Aidan Froggatt	Completed System Tests section
October 25th	Hamzah Rawasia	Completed Plan sections 2.2, 2.3, 2.4, 2.6
October 26th	Sarim Zia	Added Plan sections 2.1, 2.5, 2.7
October 26th	Burhan Kharodawala	Added Sections 1 and 5
October 27th	All	Added Reflections

Contents

1 General Information

1.1 Summary

The Verification and Validation plan sets a strong basis for documenting the testing strategies to test key features of this application. It aims to discuss an approach to test the correctness, validity, and reliability of the application and its features. Hitchly is a rideshare application which aims to help McMaster students, staff and faculty members to get an easy, convenient and trustworthy platform to carpool to and from McMaster University. Some of the key functions of this application are to match riders using the user's timetable, location, and user preferences. Another important feature that makes the application more trustworthy is the McMaster official email verification system. This ensures that each user accessing this application is truly associated with the university. The application also aims to provide a cost calculator for drivers to estimate their total cost of carpooling which they can use to set their rideshare prices. Lastly, it includes the payment integration feature which allows for a smooth transaction between drivers and riders.

1.2 Objectives

The Verification and Validation plan's main objectives are as follows:

- **Software Correctness:** Build confidence in the software's correctness by verifying and validating its critical requirements, ensuring that the application does what it claims to do.
- **Features Validation:** Validate the implemented features to ensure they provide a smooth and comfortable user experience.
- **Confidence in Stakeholders:** Build confidence among the stakeholders by providing proof of validation for all software requirements for the project.
- **Validate Components:** This plan aims to verify and confirm the proper implementation and integration of critical components like the matching algorithm, cost calculator, and payment interface.
- **Large Scale Concurrent Testing:**

- **Comprehensive Data Testing:**

The out of scope objectives are as follows:

- **Large Scale Concurrent Testing:** Due to the limited time and resource, performing large scale testing with users accessing the system concurrently to ensure server capabilities may be valuable but avoided.
- **Comprehensive Data Testing:** The validation of secured data encryption may just be limited to functional testing due to the limited resources and time for this project. It will just be limited to the verification of user authentication.

1.3 Challenge Level and Extras

There will be two extras for this project. Both documents play an important role in making the application as user friendly as possible.

- **User Manual:** This will help users navigate the application and provide a brief description of the key features to allow for a smooth and comfortable user experience.
- **Usability Testing Report:** This will ensure that users have a positive and smooth interaction with the application by observing the users while using the application. Moreover, opinions and feedback from the user will be considered and documented to improve the usability of this application. This will help evaluate the user-friendliness of the key features of the application.

1.4 Relevant Documentation

- **Problem Statement and Goals:** This document is very important in providing a layout for documenting the main problem that the application aims to solve and the main goals it plans to achieve. This provides a starting point for testing and highlights the desired results through the identification of goals in the document **?**.
- **Development Plan:** This document provides a timeline for the development of the project. It ensures that the project is on track with its development and testing. It also provides a timeline for the tests based on the timeline for the developement of the application **?**.

- **Software Requirements Specification:** This document aims to breakdown the application's key features and functionality into two-part, functional and non-functional requirements. This plays a key role in listing all key functionalities and planning requirements-based testing **?**.
- **Hazard Analysis:** This document identifies the key hazards involved with the development of this application. It helps provide a list of vulnerabilities that must be addressed and ruled out through testing **?**.
- **Module Guide:** This document provides a description of the division of the application into modules. This is important as it provides a clear basis for planing and running module-based and component-based testing **?**.
- **Module Interface Specification:** This document dives deep into exploring the interface specifications between modules. This would be important to plan testing functionality and interactions between two modules **?**.
- **Usability Testing Report:** This documents the feedback and observation from usability testing. This is an additional test which ensures the user compatibility of the application **?**.

2 Plan

This section discusses the team's plan for verification and validation of process implementation. It covers the teams and their roles, the SRS verification plan, the design verification plan, the verification and validation verification plan, the implementation verification plan, automated testing and verification tools, and the software validation plan.

2.1 Verification and Validation Team

Dr. Spencer Smith:

- Advisor and primary documentation reviewer. Provides guidance for project direction, documentation structure, and technical feedback throughout the software development lifecycle.

Lucas Dutton:

- Advisor and primary documentation reviewer. Provides guidance for project direction, documentation structure, and technical feedback throughout the software development lifecycle.

Sarim Zia:

- Responsible for reviewing team members' pull requests to suggest improvements and/or fixes to code and documentation. Ensure smooth integration of reviewed features. Lead internal beta testing throughout the development phase and lead public beta testing to collect and review all relevant feedback.

Aidan Froggatt:

- Responsible for reviewing team members' pull requests to suggest improvements and/or fixes to code and documentation. Oversee smooth integration of reviewed features through creating rulesets. Conduct internal beta testing throughout the development phase.

Swesan Pathmanathan:

- Responsible for reviewing team members' pull requests to suggest improvements and/or fixes to code and documentation. Ensure smooth integration of reviewed features. Conduct internal beta testing throughout the development phase and focus emphasis on user interface responsiveness.

Hamzah Rawasia:

- Responsible for reviewing team members' pull requests to suggest improvements and/or fixes to code and documentation. Ensure smooth integration of reviewed features. Conduct internal beta testing throughout the development phase and focus emphasis on functional feature validation.

Burhanuddin Kharodawala:

- Responsible for reviewing team members' pull requests to suggest improvements and/or fixes to code and documentation. Ensure smooth integration of reviewed features. Conduct internal beta testing throughout the development phase and focus emphasis on non-functional feature validation.

2.2 SRS Verification

We want to ensure that the SRS document is complete, accurate, and consistent, which will be verified using the following approaches:

Peer Review: The team will rely on feedback from classmates that are from other capstone teams for peer evaluation. It is expected that they will provide feedback on the clarity and completeness of each section, going through and finding any inconsistencies and flaws. Based on their findings, GitHub issues will be opened to provide specific suggestions for improvement and any aspects that can be improved.

Stakeholder Feedback: The primary stakeholder(s) of our project are McMaster University students and faculty that commute to campus. As such, we will share the SRS document with a sample of chosen stakeholders. This is important to ensure that the SRS reflects actual stakeholder needs and expectations rather than internal assumptions. The selected users will be able to give specific feedback on which features, requirements, and workflows meet their expectations of the functionality of the system. Any conflicts that may arise between user expectations and written requirements will trigger a revision of the corresponding sections. As for testing the requirements, users will be given an opportunity to use the system and perform basic testing of functionality with set cases given to each user, where they will be able to see the workflow of the application first-hand and provide feedback on the functionality and whether it meets the requirements stated in the SRS.

Team Validation: The team will also meet internally to discuss the SRS and ensure the content is consistent with the intended functionlity and capabilities of the application, and is consistent throughout. Additionally, the team will also conduct our own testing using the system, especially for non-functional requirements to ensure that the application meets the performance requirements, as well as the overall functionality of each major component of the system.

SRS Verification Checklist

- [ ] Are all of the core features (Matchmaking, scheduling, safety, payments, etc.) clearly described?

[ ] Is every requirement unique and traceable?

[ ] Has feedback from the classmate peer review been addressed?

[ ] Are the requirements clearly defined and measurable?

[ ] Are the inputs and outputs for important use cases clearly defined?

[ ] Are there any conflicting requirements and/or assumptions?

2.3 Design Verification

To ensure the design is functional and up to standards, the team will conduct several reviews with all developers to thoroughly go over the technical soundness of the design and major artifacts, such as the system's architecture, database schema, API specifications, and the UI/UX design. For the UI/UX design, it is especially important that we receive feedback from stakeholders to validate the user experience and flow. We will provide users with key user stories and the corresponding interface which is used to complete them so that they are able to get a feel of the functionality of the design. It is important that the team documents all feedback and any confusion from the users to validate the usability of the design. There will also be feedback received from peers on other capstone teams to verify design components, especially written components and diagrams.

Design Verification Checklist:

[ ] Does the database schema correctly model necessary fields for required data, and appropriate relationships?

[ ] Does the API design include secure, authenticated endpoints for core components and the functionality?

[ ] Does the system architecture clearly define how components will interact and how key features such as the matchmaking engine will work?

[ ] Do the UI wireframes clearly and intuitively display all core features and workflows?

2.4 Verification and Validation Plan Verification

To ensure that the Verification and Validation plan is complete, correct, and effective, the team will hold an internal review meeting to ensure that the document is complete and feasible. We will also get peer feedback from classmates where they will identify areas of the plan which are unclear, incomplete, or needs improvement. To keep track of the feedback from peers, GitHub issues will be used so that peers can clearly communicate their thoughts. Additionally, the team will ensure that the tests have complete coverage of requirements, and will also make use of mutation testing to verify the quality and effectiveness of the unit tests in the plan. This involves introducing small faults into the code, such as changes in the matchmaking algorithm to modify efficiency and/or effectiveness and running the test suite against it to validate that it is robust.

Verification and Validation Plan Verification:

[ ] Do the requirements in the SRS have at least one corresponding test case?

[ ] Are all test cases clear, specific, and testable, with defined expected outcomes?

[ ] Is the plan for mutation testing feasible?

[ ] Has feedback from the peer review been addressed and closed?

2.5 Implementation Verification

- **Code Walkthrough/Inspection:** New Code PR's will be reviewed by at least two team members before the change is merged into the main branch. The code will be reviewed via pull requests on GitHub and will include thoughtful titles and comments to ensure the reviewers are able to understand the changes. If any further clarification is required, the developer will conduct a code walkthrough. All feedback will be conveyed via GitHub, ensuring a standardized review procedure. Only approved code will be merged into the main branch.

- **Unit Testing:** Unit tests will be used to verify the expected behavior of the application. Each module will have a corresponding unit test suite. Unit test plans are outlined in Section 5 of this document.

- **System Testing:** System testing will ensure all functional and non-functional requirements are met by the application. Functional testing will verify key workflows, and non-functional testing will include performance, usability, and security checks. System test plans are outlined in Section 4 of this document.

- **Static Analyzers:** Static analysis tools will be used to analyze the code for formatting and good coding practices. Reports generated from static analysis will be reviewed periodically to ensure high code quality. Specific tools are listed under Section 3.6.

2.6 Automated Testing and Verification Tools

To ensure high quality, correct, and maintainable codebase, the team will use the following tools for automated testing and code verification, which were also briefly outlined in the Development Plan document.

Unit Testing Framework: Jest will be the primary framework for running unit tests for both the React Native frontend and Node.js backend logic. We will write unit tests for important business logic such as the cost-sharing calculation, parsing user timetables, and individual React components.

Backend Integration Testing: Supertest will be used to conduct integration testing on our backend API by simulating HTTP requests and verifying the responses. We will create test suites to verify that all API endpoints are functioning as expected, which includes testing for correct data responses and error handling.

Static Analysis and Formatting: ESLint and Prettier will ensure the codebase remains consistent, readable, and without common errors. ESLint will be configured to analyze our code for potential bugs and help enforce best practices, and Prettier will format code to maintain a consistent style across the codebase.

Jest will also be used for code coverage by utilizing the integrated coverage reporter to measure the percentage of our codebase that is executed by our unit and integration tests. Finally, we will also use Github Actions for Continuous Integration to automate the testing and verification process.

2.7 Software Validation

Our validation plan for the software includes extensive user testing and review sessions with all relevant stakeholders to ensure that all functional and non-functional requirements are met.

- **User Testing:** Hitchly relies on extensive external user testing to validate that the system functions correctly under real-world conditions. Beta testing will begin early with a focus on core functionality. Relevant feedback from beta users will be incorporated throughout the development phase. Internal beta testing will also be performed to validate feature functionality and consistency before releasing it for public testing.
- **Stakeholder Review Sessions:** Our stakeholder review sessions will be conducted regularly with our assigned TA to evaluate the system's progress against expected targets. These meetings will be used to refine the system's design and ensure stakeholder expectations are met.

3 System Tests

Purpose: Verify that Hitchly meets its defined functional and non-functional requirements.

Structure:

- Section 3.1 tests core functional requirements (verification and matching).
- Section 3.2 tests key non-functional qualities (reliability and usability).
- Section 3.3 maps each test to its source requirement for traceability.

Reference: Section 1.4 (G.4) *Functionality Overview* defines the functional and non-functional requirements.

3.1 Tests for Functional Requirements

Objective: Confirm correct operation of primary system functions—user verification and ride matching.

Justification: These address Hitchly's two most critical risks trust and core value creation.

Sub-areas:

- 3.1.1 User Verification Tests
- 3.1.2 Ride Matching Tests

3.1.1 Area of Testing 1 – User Verification

Purpose: Ensure only McMaster-affiliated users and verified drivers access the system. Covers: FR-1 (User verification) from Section 1.4 (G.4).

1. **Test Case 1 – Valid Sign-Up**
   Control: Automatic
   Initial State: No existing account.
   Input: Valid McMaster email + password + driver license (upload for drivers).
   Expected Output: Verification email sent → account activated after confirmation.
   Test Case Derivation: Based on credential and email-domain constraints defined in Section 1.4 (G.4).
   How test will be performed: Simulate sign-up, check database entry and email receipt.

2. **Test Case 2 – Invalid Verification**
   Control: Automatic
   Initial State: No existing account.
   Input: Non-McMaster email or invalid license format.
   Expected Output: Account creation rejected with error message.
   Test Case Derivation: Derived from input constraints and driver-license validation rules in Section 1.4 (G.4).
   How test will be performed: Input invalid data, verify system blocks progress.

3.1.2 Area of Testing 2 – Ride Matching

Purpose: Confirm that the matching algorithm correctly proposes compatible rider–driver pairs. Covers: FR-3 (Ride matching) and FR-4 (Cost estimate) from Section 1.4 (G.4).

1. **Test Case 1 – Successful Match**
   Control: Automatic
   Initial State: Verified users with valid profiles and schedules.
   Input: Rider and driver with overlapping routes/times.
   Expected Output: Match proposal generated with computed cost share.
   Test Case Derivation: Based on matching criteria and cost-sharing logic specified in Section 1.4 (G.4).
   How test will be performed: Run matching routine, check match records and cost calculation.

2. **Test Case 2 – No Compatible Match**
   Control: Automatic
   Input: Rider and driver with non-overlapping schedules.
   Expected Output: "No match found" message.
   Test Case Derivation: From matching constraints outlined in Section 1.4 (G.4).
   How test will be performed: Submit non-overlapping profiles and observe system response.

3.2 Tests for Nonfunctional Requirements

Objective: Evaluate system quality beyond correctness—reliability and usability. Approach: Use performance measurement and user feedback methods rather than simple pass/fail criteria.

Sub-areas:

- 3.2.1 Reliability (Matching Correctness)
- 3.2.2 Usability (Low-Friction Flows)

3.2.1 Area of Testing 1 – Reliability (Matching Correctness)

Purpose: Ensure consistent and repeatable ride matching results. Covers: NFR-1 (Reliability) from Section 1.4 (G.4).

1. **Test Case 1 – Repeatability Check**
   Type: Automatic / Dynamic
   Initial State: Stable dataset for verified users.
   Input: Run matching inputs multiple times.
   Expected Result: Identical pairings each run.
   How test will be performed: Automated script logs output and compares hash results.

2. **Test Case 2 – Stress Test**
   Type: Dynamic Performance Test
   Input: Simulated peak load ($\geq$ 1000 simultaneous match requests).
   Expected Result: $\geq$ 95% successful matches within response-time threshold.
   How test will be performed: Load-testing tool records failures and latency for reliability assessment.

3.2.2 Area of Testing 2 – Usability (Low-Friction Flows)

Purpose: Validate ease of use for key flows (sign-up, ride offer/request, confirmation). Covers: NFR-2 (Usability) from Section 1.4 (G.4).

1. **Test Case 1 – First-Time User Scenario**
   Type: Manual / Survey-Based
   Initial State: New participants with no training.
   Input: Observe users completing sign-up and ride requests.
   Expected Result: $\geq$ 90% complete tasks without help in $\leq$ 3 minutes.
   How test will be performed: Usability study plus survey metrics (Appendix).

2. **Test Case 2 – Interface Clarity**
   Type: Static Inspection / Heuristic Evaluation

Input: UI screens and labels.
Expected Result: No ambiguous labels or layout clutter violations.
How test will be performed: Walkthrough by UX reviewers using standard heuristics.

3.3 Traceability Between Test Cases and Requirements

Requirement ID	Requirement Description (from 1.4 G.4)	Test Case ID(s)
FR-1	User Verification	3.1.1-1, 3.1.1-2
FR-3	Ride Matching	3.1.2-1, 3.1.2-2
FR-4	Cost Estimate	3.1.2-1
NFR-1	Reliability (Matching Correctness)	3.2.1-1, 3.2.1-2
NFR-2	Usability (Low-Friction Flows)	3.2.2-1, 3.2.2-2

Table 1: Traceability between test cases and requirements as defined in Section 1.4 (G.4) Functionality Overview.

4 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

4.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on

verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

4.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

4.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

3. ...

4.2.2 Module 2

...

4.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

4.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

How test will be performed:

4.3.2 Module ?

...

4.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

5 Appendix

5.1 Usability Testing Plan

Usability testing will be conducted at the end of the first stage of development of the application. This will be used to assess and judge the user's compatibility with the application. This section will include a plan for conducting this test.
The goal is to conduct usability testing right after the development of the MVP of Hitchly. The MVP (minimal viable product) includes the ride matching algorithm, and user authentication. The plan is to find a group of students and staff to navigate through the application on their own and ask them for any feedback they may have. Secondly, ask the users a set of questions and observe their expressions; the time it takes them to complete a given task and features that may confuse them. Below is a list of possible questions/tasks a user must complete to test the user compatibility of the application:

- Launch the application and register on the application.
- Navigate to the user profile and add your timetable to the application. Update some information on your profile.
- Navigate to the list of matched drivers/riders in the application.
- Select your profile type, Drivers or Riders.
- For Riders, click on the matched driver and access their user information.
- For Drivers, click on the matched rider and access their user information.
- Log out of the application and re-login to the application with the registered credentials in a given time constraint.

Here are the main objectives of the test:

- Users find the process of creating an account intuitive and easy.
- Users are able to verify their emails quickly.

- Users are able to navigate through the various sections of the app without any problem.
- Users address any discomfort they have while using the application.
- Users address any errors they face while using the application.

This brief test plan provides a structured approach for analyzing user's interaction with the application and understanding the key areas of improvement. Users will be monitored using a time metric to check if a user is able to complete a task within a given time constraint, ensuring efficiency. Moreover, users will be asked for feedback after each task they perform to get as much feedback as possible to improve the user experience.

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. **What went well while writing this deliverable?**
   (Swesan) The structure of the V&V plan came together smoothly once I reviewed the example templates and understood what was expected. I was able to clearly link each test to the functional and non-functional requirements from Section 1.4 (G.4), which made the writing process more organized. Formatting, tone consistency, and version control all went well, and I'm happy with how cohesive the final document feels.

   (Burhanuddin) This document was very helpful in setting the basis for identifying and documenting the testing approaches for our capstone project. Specifically, the list of documentation was a very critical section that allowed me to understand the importance of each document and their connection to the specific tests. Overall, we had an in-depth discussion of the testing approaches we wanted to consider and documented them with full clarity.

   (Hamzah) A few things that went well were that this document tied back to previous documents we already completed, so being able to see everything come together under the project was nice. I worked on the

Plan section, so having a clear project scope and defined features made it easier to brainstorm what needed to be verified and validated for each part of the project. It was also nice to have some reference/example documents to look at to get a better idea of the expectations in some sections.

(Sarim) Working on one section with another team member meant we had to ensure we were fairly dividing the section. It also meant that some parts would be dependent on the other team members to finish before they could be started. We were able to deal with this well by discussing our schedules early on and setting key deadlines so neither of us suffered from delays by the other.

2. **What pain points did you experience during this deliverable, and how did you resolve them?**
(Swesan) The main difficulty was determining how detailed each test case should be and distinguishing between verification and validation tasks. At first, I wrote overly detailed descriptions, but after reviewing previous examples and clarifying expectations from the course materials, I refined the structure to focus on clarity and traceability. Once I aligned the test coverage with the proof-of-concept scope of Hitchly, the process became much more straightforward.

(Burhanuddin) Initially, the documentation had a few areas/sections that were very ambiguous in what they wanted. They lacked clarity in the description of the sections. Some of the descriptions were also outdated, which caused some confusion. Example, section 2 had a subsection for challenges which is part of an old rubric. Through the TA meetings, we were able to clarify what the description meant.

(Hamzah) At first it was a bit hard to understand what was expected in the Plan sections, in terms of what can be actionable, and there seemed to be a lot of overlap between sections. Also, since our team does not have a supervisor for the project, it felt like we were missing out on a big aspect of the verification process. After speaking with the TA, I was able to get some more clarity on what parts can overlap

between sections, and also how we can incorporate stakeholders into the plan instead of a supervisor.

(Sarim) Our team had numerous questions planned for the TA meeting, however, due to traffic delays, most of us were unable to reach on time. We were able to communicate this early to the team members who were present and get our deliverable questions across even with the inability to attend on time. For me, these questions helped clear up ambiguity in the Plan section and ensured I was able to have a clear understanding while writing the section.

3. **What knowledge and skills will you need to acquire to successfully complete the verification and validation of your project?**
(Swesan) I need to deepen my understanding of dynamic testing and automation. Specifically, I want to improve at writing reproducible automated test scripts for functional verification and gain familiarity with load testing tools to assess performance under peak usage conditions. These skills will help ensure Hitchly's matching and verification features are both reliable and scalable.

(Burhanuddin) As I am fairly new to app development, I need to learn all types of web development testing frameworks. I also need to understand them in Typescript, a language we will be using for the development of the application.

(Hamzah) An important skill the team will need to learn is Jest, which will be the primary unit testing framework we use to test our codebase. Writing unit tests with good coverage and effectiveness is crucial for the project, so it's important to be familiar with the tools that will be used.

(Sarim) I think when it comes to verification, knowing the tool is important, however it is also important to know theory about what type of tests we can and should perform alongside any additional theory that could help provide complete verification coverage. However, the last time our team members took a class related to verification testing

was years ago, so it may be an aspect we need a refresher on.

(Swesan) The structure of the V&V plan came together smoothly once I reviewed the example templates and understood what was expected. I was able to clearly link each test to the functional and non-functional requirements from Section 1.4 (G.4), which made the writing process more organized. Formatting, tone consistency, and version control all went well, and I'm happy with how cohesive the final document feels.

(Aidan) I found that structuring the V&V plan around clear verification tiers (SRS, design, implementation, and validation) made the writing process straightforward. Aligning test cases directly with functional and non-functional requirements improved traceability and reduced redundancy. Integrating automation tools like Jest and GitHub Actions into the plan also felt natural since these align closely with our existing workflow. Overall, this deliverable came together efficiently once the structure and testing philosophy were defined.

4. **What pain points did you experience during this deliverable, and how did you resolve them?**
   (Swesan) The main difficulty was determining how detailed each test case should be and distinguishing between verification and validation tasks. At first, I wrote overly detailed descriptions, but after reviewing previous examples and clarifying expectations from the course materials, I refined the structure to focus on clarity and traceability. Once I aligned the test coverage with the proof-of-concept scope of Hitchly, the process became much more straightforward.
   (Aidan) One challenge was balancing feasibility with completeness — the temptation was to include every possible validation step, but time and scope required focusing on what could realistically be tested this term. I also found it tricky to document test coverage without finalizing the detailed design. To resolve this, I focused on defining testing strategies conceptually and left module-specific details as placeholders for post-design updates.

5. **What knowledge and skills will you need to acquire to successfully complete the verification and validation of your project?**
   (Swesan) I need to deepen my understanding of dynamic testing and

automation. Specifically, I want to improve at writing reproducible automated test scripts for functional verification and gain familiarity with load testing tools to assess performance under peak usage conditions. These skills will help ensure Hitchly's matching and verification features are both reliable and scalable.
(Aidan) I want to strengthen my understanding of end-to-end testing workflows and CI/CD validation. Specifically, I need to improve at automating test pipelines that integrate unit, integration, and mutation testing with coverage reporting. I also plan to deepen my knowledge of usability testing methodologies and how to quantify user satisfaction during beta testing.

6. **For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Which will you pursue, and why?**
(Swesan) To build my dynamic testing and automation skills, I can:
(1) Review documentation and tutorials for testing frameworks and testing utilities.
(2) Develop small-scale automated test cases directly within the Hitchly repository to gain hands-on experience.

I will prioritize the second approach since applying testing concepts directly in code helps uncover integration issues early and strengthens confidence in both the testing process and implementation reliability.
(Burhanuddin) I will first look at language documentation and understand the basics of technology. Secondly, I will go through some video tutorials and get some hands-on practice with the desired testing tools.
(Hamzah) One approach is to learn the important principles and strategies of good unit tests and writing test suites for a project, as having the fundamental understanding is the most important aspect. Once we are comfortable with that, we can follow video tutorials and look at examples of writing unit tests using Jest to get familiar with the framework.
(Sarim) The best approach would be to review any relevant course material from past years. This is a good approach because it would simply be a quick refresher in a format, we are familiar with, rather than learning something with a new format/approach. However, another option

is to follow any online sources such as tech blogs that show how to provide complete system coverage.

(Aidan) For CI/CD and automated test coverage, I can: (1) Experiment with GitHub Actions to create incremental test pipelines that validate each pull request automatically. (2) Study testing patterns from large-scale open-source monorepos to understand practical approaches to test orchestration. For usability validation, I can: (1) Research UX testing frameworks and heuristics to design structured surveys and A/B test flows. (2) Conduct real-world pilot tests with early users to collect actionable feedback. I plan to prioritize hands-on experimentation with automated pipelines since this will directly improve code reliability and streamline team workflows.