System Verification and Validation Plan for Software Engineering

Team 1, BANDwidth
Declan Young
Ben Dubois
Nathan Uy
Aidan Mariglia

January 5, 2025

Revision History

Date	Version	Notes
2024-11-03	0.0	Added initial VnV plan
2025-01-02	0.1	Address peer review - use consistent terminologies

Contents

List of Tables

This document outlines the plans in place for the validation and verification of the Battery SOC Algorithm testing tool. It will give details regarding the plans that are in place to verify and validate each stage of the development process of the application, from the requirements to the completion of the software. Additionally, detailed descriptions of the necessary tests for both functional and non-functional requirements to be fulfilled by the system are included, to ensure that the final product is both complete and correct.

1 General Information

1.1 Summary

The Battery SOC Algorithm testing tool will be tested. The tool allows for the submission of user algorithms, attempting to provide battery state of charge estimations. The algorithms will be run against a series of tests, and the results reported back to the user. A leaderboard with various filtering and sorting options will be provided to view the results of different algorithm submissions.

1.2 Objectives

The main objective of this verification and validation plan is to ensure that the battery SOC algorithm testing tool can accurately evaluate the correctness and performance of the submitted algorithms and provide a user-friendly experience. In addition, we must ensure that the system is scalable and can handle high concurrent user submissions.

Due to time constraints, some objectives will be out of scope, namely:

- A comprehensive usability testing - this will require significant time and diverse user demographics. Instead, we will focus on using design best practices and have our primary users provide us with feedback.

We will first prioritize the correctness of the algorithm performance reports. Then, the load-handling capacity will be our second priority and the usability assessment will be the least among our top priorities. Any dependencies and external libraries will be assumed to have already been verified by their implementation team.

1.3 Challenge Level and Extras

The challenge level for this project is general since the technologies and domain knowledge required are simple and the problem itself is not novel enough to constitute an advanced challenge level. As an extra for this project, we will include thorough user guides and walkthroughs as part of the final application.

1.4 Relevant Documentation

- Software Requirements Specification Document
  - The SRS is needed in order to understand the system requirements and verify that the software meets its intended goals
  - https://github.com/AidanMariglia/SOCAlgoTestPlatform/blob/main/docs/SRS/SRS.pdf
- Module Guide and Modular Interface Specification
  - Both design documents are needed to verify the design of the system. Each design element should link to the requirements to ensure that all requirements are accounted for and there are no missing functionalities.
  - https://github.com/AidanMariglia/SOCAlgoTestPlatform/blob/main/docs/Design/SoftArchitecture/MG.pdf
  - https://github.com/AidanMariglia/SOCAlgoTestPlatform/blob/main/docs/Design/SoftDetailedDes/MIS.pdf

2 Plan

This section contains the validation and verification plan for our application. It will outline the team responsible for verification and validation, as well as the plans that this team will carry out to verify the SRS, the design, the verification and validation plan itself, the implementation and the software. The tools that will be used for this verification are also briefly described in this section.

2.1 Verification and Validation Team

Table 1: Verification and Validation Team

Team Member	Roles
Aidan Mariglia	Tester/QA Analyst - develop and execute test plans and review relevant documentation
Nathan Uy	Tester/QA Analyst - develop and execute test plans and review relevant documentation
Ben Bubois	Tester/QA Analyst - develop and execute test plans and review relevant documentation
Declan Young	Tester/QA Analyst - develop and execute test plans and review relevant documentation
Dr. Kollmeyer/Atjen	Both supervisors will oversee the verification process and ensure all requirements are followed and completed.

2.2 SRS Verification Plan

Meeting Plan:
At a weekly check-in, the key requirements related to solving our problem statement will be presented, alongside a brief justification for the requirement. At this point, our supervisor can audit the requirement. For less important non-functional requirements, they can be briefly discussed to see if they are reasonable and well-suited to the problem statement.

Peer Review:
Issues are created by peers with regards to the integrity of the SRS. They will be reviewed by team members, and if they are valid feedback they will be integrated into the SRS.

Requirement Audit Checklist

- Does the requirement satisfy part of the problem statement?
- Does the requirement assume implementation details?
- Is the requirement necessary?

- Is the requirement reasonable?

2.3 Design Verification Plan

Checklist:
In order for the design of our application to be reviewed by our classmates, we will make a checklist that contains they key considerations and principles that our design should adhere to. The following checklist questions will be used for this step of verification:

- Does the design follow best practices for design patterns?
- Are the specifications of the design abstract?
- Does the design contain all necessary exception handling?

This checklist will be implemented as a GitHub issue template that any person that is verifying the application's design can create. This stage of verification should ideally reveal any more apparent issues with the design, so that they can be resolved before the more exhaustive walkthrough of the design.

Design Document Walkthrough:

In order for the design to be reviewed by our supervisors/stakeholders, we will have a detailed walkthrough of the design of our application and all related documentation. By performing this step of verification we ideally should be able to reveal any errors, missed considerations or ambiguities that were missed in the other stages of verification, such as the checklist. It is most suitable to use a walkthrough with the supervisors rather than other tools, as these groups of people are very knowledgeable about the system being created and therefore should be able to provide the greatest amount of insight and criticism about the design by going through it thoroughly.

This stage of verification should reveal any fundamental issues with the design of the application. Additionally, for each issue that is encountered during the walkthrough, having the supervisors and stakeholders present will be very beneficial to determining the correct solution efficiently.

2.4 Verification and Validation Plan Verification Plan

To ensure that we have an effective Verification and Validation plan, it is essential to verify the plan itself by:

- Conducting an internal review - we as a team will systematically examine the VnV plan to ensure that all our artifacts are verified, and are aligned with the project requirements.
- Peer review - colleagues will review our plan and check for any inconsistencies or anything that is missing in our plan. By having different perspectives, peers might be able to spot more potential issues.
- Supervisor review - At a high level, supervisors will assess the plan to ensure it is sufficient to verify and validate that the system will achieve the project goals.
- Mutation testing - we will modify test cases/requirements and see if the changes are detected. This will ensure that the verification plan can effectively find faults in the system.

Checklist:

- The objectives of the VnV report are clearly defined
- There is a detailed plan on how each artifact is going to be verified. It must be consistent throughout and complete
- All requirements, both functional and non-functional are accounted for.
- There is a traceability matrix for requirements and the verification/validation tests
- Usability survey questions exist and are well-structured.

2.5 Implementation Verification Plan

Refer to section 3 of this document for detailed tests for each requirement and section 4 for the unit tests.

2.6 Automated Testing and Verification Tools

Code Linting:

- Autopep8 - This linter will be run with pre-commit to ensure that all Python code conforms to the PEP8 standard

Unit Testing:

- Pytest - This unit testing framework will be used to unit test all of the application's back-end code.

Automated Integration/End to End Testing:

- Postman - API testing tool
- Selenium - An automated testing framework that will be used to test the application end-to-end, from the front end

Profiling Tool:

- Memray - python memory profiling tool. Will be used to evaluate system memory usage and detect memory leaks.

Code coverage:

- Pytest-cov - Coverage reporting tool which integrates with pytest and provides flexible converge reporting for unit tests.

CI/CD:

- GitHub actions - This CI/CD platform will be used to run workflows in order to test the following aspects of the system (as described above):
  - Linting
  - Unit testing

2.7 Software Validation Plan

Currently there are no formal plans for validation. Informal review of requirements happens on a weekly bases during a 30 minute sync up meeting. Through this process we incrementally discover new requirements, as well as refine existing ones. As the project progresses this process will switch from requirements capture/refinement to validation, where small parts of the system can be observed and validated.

3 System Tests

This section outlines the necessary tests for the application, in order for all requirements (functional and non-functional) to be completely verified. A traceability matrix will also be included to link all system tests to their related requirements.

3.1 Tests for Functional Requirements

This section covers all system tests for the functional requirements. The tests are divided into the following subsections: Algorithm Submission, Results Reporting, Parallel Execution, Account Creation, User Login, Data Segregation, Save Results, Leaderboard Access, Leaderboard Categorization, Sort Leaderboards, Execution Progress, Error Notification. As described in the SRS Section 9, the tests for these different areas of the system should cover all functional requirements of the system.

Algorithm Submission

1. FR1-ST1

   Control: Manual

   Initial State: The algorithm submission page

   Input: A valid algorithm

   Output: Success notification from the system

   Test Case Derivation: The success notification will be output to the screen by the system if the upload of the algorithm was successful and the algorithm file is valid, therefore this notification is expected when a valid file is submitted.

   How test will be performed: A valid algorithm will be manually submitted through the submission page of our user interface and we will wait to see if the success notification is displayed on the page. If the success notification is displayed, the test passes.

2. FR1-ST2

Control: Manual

Initial State: The algorithm submission page

Input: An invalid algorithm

Output: Error notification from the system

Test Case Derivation: The error notification will be output to the screen by the system if the upload of the algorithm was successful, but the algorithm file is invalid. Therefore, this notification is expected when submitting an invalid file.

How test will be performed: An invalid algorithm will be manually submitted through the submission page of our user interface and we will wait to see if the error notification is displayed on the page. If the error notification is displayed, the test passes.

Results Reporting

1. FR2-ST1

   Control: Manual

   Initial State: The algorithm submission page

   Input: A valid algorithm

   Output: Success/result data returned from the system

   Test Case Derivation: The results of a successful test will be displayed after the algorithm execution was successful. Therefore, when submitting a successful algorithm we should expect the result of the algorithm execution to be displayed.

   How test will be performed: A tester will use an example algorithm, and submit it through the interface. After an appropriate amount of time has passed if the result is displayed, the test passes. If it does not, it was a failure.

2. FR2-ST2

   Control: Manual

   Initial State: The algorithm submission page

Input: An invalid algorithm

Output: Result data indicating the algorithm's execution has failed

Test Case Derivation: When the system receives an invalid algorithm or algorithm with an error in it, it should return a result to the user indicating that the algorithm's execution as failed, as well as additional details regarding the execution (time etc).

How test will be performed: The tester will use an input algorithm that contains an error, and submit it through the user interface. After a short period of time the execution will fail, and this result will be displayed to the user.

Parallel Execution

1. FR3-ST1

   Control: Automatic

   Initial State: Running system with no jobs queued

   Input: Two valid algorithms

   Output: Both algorithms execute concurrently

   Test Case Derivation: In order to confirm that jobs can be run in parallel, two jobs should be sent in simultaneously. The tests should both be executed independently of each other concurrently.

   How test will be performed: Requests will be made to the API with both algorithms, the id's of the submissions will be recorded. The status of the submissions will both be checked in the database, and they should both be in progress.

Account Creation

1. FR4-ST1

   Control: Manual

   Initial State: Account creation screen

   Input: New Username and password

Output: Success message

Test Case Derivation: When creating a new account, the user provides the new information. As a result, the system for which the account is being created needs only to respond with confirmation that the creation was successful, the username and password should then function for the system.

How test will be performed: A tester will submit an account creation request with their username and password, and wait for a response. If a success message is received, then the test is successful.

2. FR4-ST2

Control: Manual

Initial State: Account creation screen

Input: Existing username and password

Output: Account already exists error

Test Case Derivation: When a user attempts to create an account with credentials for which an account already exists, account creation should fail.

How test will be performed: The person testing will attempt to create an account with the same username as an existing account. The test is successful if the response is that an account with the specific credentials already exists.

User Login

1. FR5-ST1

Control: Automatic

Initial State: Login screen

Input: Valid user credentials

Output: User is logged into their account

Test Case Derivation: If a user provides valid credentials to the system on the log in screen, the system should allow the user access to their account

How test will be performed: An automated testing tool (ex. Selenium) will be used to populate the username/password fields with valid credentials. It will then validate that the user is logged in after attempting to log in with those credentials.

2. FR5-ST2

   Control: Automatic

   Initial State: Login screen

   Input: Invalid user credentials

   Output: Login error returned (invalid username/password)

   Test Case Derivation: If a user provides invalid credentials to the system on the log in screen, the system should throw an error and refuse the user access to that account

   How test will be performed: An automated testing tool (ex. Selenium) will be used to populate the username/password fields with invalid credentials. It will then validate that the user is not logged and that the login error is displayed, after attempting a log in with those credentials.

Data Segregation

1. FR6-ST1

   Control: Automatic

   Initial State: Algorithm submissions/results

   Input: Valid user credentials, and the id of a submission corresponding to that user

   Output: The details of the submission being requested

   Test Case Derivation: Since the submission being requested does belong to the user requesting it, the submission's details should be returned

   How test will be performed: An automated testing tool (ex. Selenium) will be used to login with valid user credentials, and retrieve a past submission for the user that is currently logged in. Since this submission is associated with the user, the submission results should be returned.

2. FR6-ST2

   Control: Automatic

   Initial State: Algorithm submissions/result

   Input: Valid user credentials, and a submission corresponding to a different user

   Output: Permission Denied Error

   Test Case Derivation: Since the submission being requested does not belong to the user requesting it, no result should be returned (it does not exist for that user).

   How test will be performed: An automated testing tool (ex. Selenium) will be used to login with valid user credentials, and retrieve a past submission for a different user than the one that is logged in. Since this submission is not associated with the user, no result should be returned.

Save Results

1. FR7-ST1

   Control: Automatic

   Initial State: Algorithm result screen

   Input: The result of an algorithm to save

   Output: The algorithm has been saved successfully

   Test Case Derivation: If a user saves the result of an algorithm's execution, the system should output that the algorithm has been saved successfully.

   How test will be performed: An automatic testing tool will submit an algorithm. Once the execution is completed, it will attempt to save the result. Once saved, the system should respond with that save success message. Additionally, the tool will attempt to retrieve the result and validate the result to ensure that it was saved successfully.

Leaderboard Access

1. FR8-ST1

   Control: Manual

   Initial State: AThe user logged in on the home screen

   Input: The user navigates to the leaderboard screen

   Output: Leaderboard becomes visible with submissions from other users

   Test Case Derivation: The leaderboard should be available to all authenticated users, the system should return the relevant data when requested.

   How test will be performed: A tester will log in and from the home page navigate to the leaderboard page. From there they will confirm that relevant data is being displayed. If the data is present then the test passes, if there is no relevant data present or errors navigating to the page, the test fails.

Leaderboard Categorization

1. FR9-ST1

   Control: Manual

   Initial State: The user logged in on the leaderboard screen

   Input: User attempts to view the leaderboard for algorithms of type SOC

   Output: Leaderboard with only the result for algorithms of type SOC

   Test Case Derivation: When the user attempts to view the leaderboard for only algorithms of type SOC, only SOC algorithms should be displayed, with algorithms of all other types being filtered out.

   How test will be performed: While logged in as a test user, we will navigate to the leaderboard page and apply the filter for algorithms of type SOC. After applyinf this filter we will ensure that only algorithms of type SOC are displayed.

Sort Leaderboards

1. FR10-ST1

   Control: Automatic

   Initial State: A non-empty and unsorted leaderboard exists in the system with at least 2 items.

   Input: A sorting criteria from the user

   Output: A sorted leaderboard based on the sorting criteria submitted

   Test Case Derivation: The system will sort the leaderboard according to the sort criteria submitted.

   How test will be performed: Initialize a mock database with a table that has unsorted data. The data can include properties that are the same among different entries. Then, call the sorting algorithm with different sort criteria. Compare the output with the expected sorted table.

Execution Progress

1. FR11-ST1

   Control: Manual

   Initial State: The algorithm submission page

   Input: A valid/invalid algorithm is submitted

   Output: Display the progress of execution giving updates at each stage.

   Test Case Derivation: The system will display the progress of the executing algorithm, which includes all the important steps during the algorithm's runtime.

   How test will be performed: An algorithm is submitted manually through the submission page of our user interface and throughout the algorithm's execution, we will monitor the output display for progress updates. The test passes if all the expected progress messages are displayed in the correct order and are being output at around the same time it is expected to be displayed.

Error Notification

1. FR12-ST1

   Control: Manual

   Initial State: The algorithm submission page

   Input: An algorithm that will throw an exception

   Output: Failure notification from the system

   Test Case Derivation: The error notification will be output to the screen by the system if the execution of the algorithm is unsuccessful, therefore this notification is expected when a faulty algorithm is submitted.

   How test will be performed: An algorithm that will throw an exception will be manually submitted through the submission page of our user interface. We will wait to see if the error notification is displayed on the page. If the error notification is displayed, the test passes.

3.2 Tests for Nonfunctional Requirements

This section covers all system tests for the non-functional requirements. The tests are divided into the following subsections: Look and Feel, Learning, Usability, Speed and Latency, Authentication, Database Integrity, Robustness or Fault Tolerance, Capacity, Scalability/Extensibility, and Adaptability. As described in sections 10-17 of the SRS, the tests for these different areas of the system should cover all non-functional requirements of the system.

Look and Feel

1. NFR1-ST1

   Type: Manual

   Initial State: The website is open in a web browser

   Input/Condition: The user navigates the entire website

   Output/Result: Summary of appearance, styles and designs that resemble the website Kaggle and those that do not.

How test will be performed: A user will load both our website and Kaggle in the same browser. He/she will then compare our website's look and feel with Kaggle's and list down all elements that are similar and more importantly, elements that are not similar.

Learning

1. NFR2-ST1

   Type: Manual

   Initial State: The user has access to the documentation of the website and the website itself

   Input/Condition: The user checks for the documentation of specific functionality.

   Output/Result: A detailed documentation exists for the specified functionality and the user confirms that it is complete and clear.

   How test will be performed: A new user will use the documentation to search how to use a specific core functionality of the program. The documentation should be sufficient for the user to start using the functionality without needing further assistance.

2. NFR2-ST2

   Type: Manual

   Initial State: The user is logged in to the website

   Input/Condition: User accesses a feature for the first time

   Output/Result: Tooltips/hints are shown to the user on how to use the feature and the user can proceed with his/her task.

   How test will be performed: A user of the website will attempt to use a feature he/she has not used before and confirm that the hints allow them to proceed with their task without needing further assistance.

Usability

1. NFR3-ST1

   Type: Manual

   Initial State: The website is open in a web browser

   Input/Condition: The user navigates the entire website

   Output/Result: The user will list down all unfamiliar symbols encountered on the website or any ambiguous symbols that can cause confusion

   How test will be performed: A user will explore the entire website and evaluate the clarity of the symbols/icons encountered.

2. NFR3-ST2

   Type: Manual

   Initial State: The website is open in a web browser

   Input/Condition: The user skims through all the terminologies used in the website.

   Output/Result: The user notes all inconsistencies in the terminologies used.

   How test will be performed: A user will go through different parts of the website, noting any inconsistencies in terminology.

3. NFR3-ST3

   Type: Manual

   Initial State: The user is logged in to the website

   Input/Condition: The user performs an action on the website (like submit an algorithm)

   Output/Result: The user either sees a message indicating the action was successful/unsuccessful or at least a loading icon if there is latency.

   How test will be performed: A user will submit an algorithm through the algorithm submission page and confirm that a success/failure message was displayed on the screen after submission. In the case of a delay, a loading icon is displayed.

4. NFR3-ST4

   Type: Manual

   Initial State: The user is logged in to the website

   Input/Condition: The user encounters a complex use case

   Output/Result: A walkthrough guide exists for that use case and the user is able to complete his/her task using only the walkthrough successfully.

   How test will be performed: A user will try to follow the walkthrough for a complex use case and confirm that the use case can be completed without needing further assistance.

Speed and Latency

1. NFR4-ST1

   Type: Functional

   Initial State: The user is logged in to the website

   Input/Condition: A request for the leaderboard (sorted/filtered and unsorted/unfiltered)/al

   Output/Result: The result is returned in less than 3 seconds

   How test will be performed: Use a Postman collection to make the specified requests. Use P

Authentication

1. NFR5-ST1

   Type: Functional/Manual

   Initial State: The login page is open

   Input/Condition: Valid credentials/Invalid credentials

   Output/Result: If valid credentials are passed in, then the website's main page is displayed and access is granted. Otherwise, access is denied and an error message is displayed indicating an invalid username/password was provided.

How test will be performed: Attempt logging in using valid credentials and verify that the user logs in to the account that corresponds to the provided credentials. Also, attempt to log in with an invalid credential and verify that an error message is displayed and the user is not granted access.

2. NFR5-ST2

   Type: Manual

   Initial State: The user is logged in

   Input/Condition: A user attempts to view another user's email address

   Output/Result: If the user has an admin role, the other user's email is displayed when viewing their account. If the user does not have an admin role, the other user's email is not displayed when viewing their account.

   How test will be performed: Log in as an admin user and check if users' emails can be viewed. Then, log in as a non-admin user and attempt to view users' emails and ensure they are not visible.

Database Integrity

1. NFR6-ST1

   Type: Functional

   Initial State: The database is initialized

   Input/Condition: Invalid data (data that do not follow the proper schema)

   Output/Result: The database rejects the input and returns an error message that states invalid data.

   How test will be performed: Try to insert to the tables in the database some data that do not follow the tables' schema. Verify that for each incorrect data input, there is an error message returned and the table does not save any of the incorrect data.

2. NFR6-ST2

   Type: Manual

   Initial State: The database is populated.

   Input/Condition: The database has scheduled backups

   Output/Result: The backup files are created and stored and can be accessed

   How test will be performed: After a backup has been scheduled for the database, ensure that the backup files are complete by attempting to restore from the backup and there are no data loss.

Robustness or Fault Tolerance

1. NFR7-ST1

   Type: Manual

   Initial State: Logged into the website for two different users

   Input/Condition: Submit an algorithm that causes an error for one of the users

   Output/Result: The error is only displayed for the user it was created for, but is not for the other user that is logged in

   How test will be performed: Log in to the accounts for two separate users. Submit an algorithm for one of the logged-in users. Ensure that the error that occurs/is displayed is only for the user that submitted the algorithm that causes it. The state of the other account should not be impacted by the error at all.

Capacity

1. NFR8-ST1

   Type: Functional

   Initial State: Logged into the website for 500 users

   Input/Condition: Submit an algorithm for each of the 500 users

Output/Result: The execution of all of the algorithms should be executed concurrently immediately after submission

How test will be performed: Use selenium to log into the accounts for 500 users. Submit an algorithm for all 500 users at the same time. The status of the execution for all 500 submissions should be validated to be in progress after submission.

Scalability or Extensibility

1. NFR9-ST1

   Type: Functinal

   Initial State: Logged into the website for 500 users

   Input/Condition: Submit an algorithm for each of the 500 users

   Output/Result: The number of resources used of the backend should be 50 times that of when 1 job is submitted (Resources should scale every 10 jobs)

   How test will be performed: Log into the accounts for 500 users. Submit an algorithm for all 500 users at the same time. Monitor the number of resources used in the backend to ensure it has scaled resources correctly.

Adaptability

1. NFR10-ST1

   Type: Functional

   Initial State: N/A

   Input/Condition: Run all existing tests, for Safari, Microsoft Edge, Google Chrome and Firefox.

   Output/Result: All tests should pass.

   How test will be performed: Use Selenium to run all automated tests for each of the compatible browsers for our application

3.3 Traceability Between Test Cases and Requirements

Table 2: Traceability Between Test Cases and Requirements

Test Case	**Requirements**
Algorithm Submission	FR-1
Results Reporting	FR-2
Parallel Execution	FR-3
Account Creation	FR-4
User Login	FR-5
Data Segregation	FR-6
Save Results	FR-7
Leaderboard Access	FR-8
Leaderboard Categorization	FR-9
Sort Leaderboard	FR-10
Execution Progress	FR-11
Error Notification	FR-12
Look and Feel	LFR-1, SR-1
Learning	LR-1, LR-2
Usability	UPR-1, UPR-2, UPR-3, UPR-4
Speed and Latency	SLR-1, SLR-2, SLR-3
Authentication	ACR-1,ACR-2
Database Integrity	IR-1, IR-2
Robustness or Fault Tolerance	RR-1
Capacity	CR-1
Scalability or Extensibility	SR-1
Adaptability	ADR-1

References

4 Appendix

4.1 Usability Survey Questions?

General Usability: Rank from 1-5 (1 being the lowest and 5 being the highest)

- How easy was it to navigate the website?
- How would you rate the overall design and layout of the website?
- How satisfied are you with the leaderboard search and filtering options?

Provide specific responses to each

- What was your biggest pain point navigating the website?
- Was there any task you were unable to do, even after hints were provided by the website and after looking through the documentation/walkthroughs provided?

List some items

- Could you list any inconsistent terminologies used on the website?
- Could you list any unfamiliar symbols encountered on the website?

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?
2. What pain points did you experience during this deliverable, and how did you resolve them?
3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.
4. 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? Of the identified approaches, which will each team member pursue, and why did they make this choice?

Nathan Uy

1. Writing the tests for functional requirements went well because we have very well-defined functional requirements and it is clearly defined what is expected from the system, so coming up with tests and ensuring completeness of tests were straightforward.

2. Writing the tests for non-functional requirements was challenging. Firstly, we had many NFRs and had to trim it down first to include only the important ones. In addition, some of our NFRs are very hard to measure quantitatively like usability and maintainability. So, it was quite challenging to come up with metrics to assess them.

Declan Young

1. During this deliverable, writing the system tests went well. I think that because we have already spent a significant amount of time creating and refining the requirements, it was easy to determine how we will test the system to ensure that it fulfills all of them. Additionally, since all the tests we made were directly traced to requirements, it was fairly clear what tests were needed to test the entirety of the system.

2. One of the pain points during this deliverable was writing the verification and validation plans for the different aspects of the project. I think this was the case because verifying many aspects of the project, such as the SRS or the VnV itself is not very straightforward, and is very difficult to do in a complete manner. We resolved this pain point by considering all of the different suitable methods of verification for each aspect of the project, and picking those that we thought were most appropriate based on their effectiveness and feasibility. Another pain point was determining what testing tools to use. Since there is such a large number of different testing tools, each one being suitable for different applications. To resolve this pain point, we did research for tools that useful in the specific applications we will be doing, and narrowed the available options down based on experience we had with them and ease of use in order to select the best options.

Aidan Mariglia

1. During the work on this deliverable, collaborating as a team went very well. Now that we have a few deliverables under our belt, the team was very effective at applying working strategies such as meeting early, dividing work into discrete chunks, discussing early the levels of importance for different sections and key points which the team wanted to

make. These strategies have been refined through working on previous deliverables, and now result in a more efficient working process.

2. Effectively choosing an area to focus on for this deliverable caused difficulty. A result of complete past deliverables, the team realized that putting an even effort across the board was not effective due to the size and time constraints associated with the deliverable. As a result we decided to single out what we thought was the most important part of the deliverable, and put an increased effort there, even if it meant a reduced effort in other areas. It was difficult however to agree on what was the most important area of this deliverable. In the end we decided it was the system tests.

Benjamin Dubois

1. During this deliverable creating the checklist to ensure the effectiveness of our verification and validation plan went well and we were able to quickly agree on the items and create this list. I believe this went well as we had previously discussed how we were going to ensure the effectiveness of tests and already had a clear idea of what we needed to complete to ensure this.

2. The largest pain point when completing this deliverable was creating the non-functional requirement tests. This is because we had some non-functional requirements like "The appearance of the system should look similar to Kaggle" that are hard to test as they can be very subjective from person to person. Unlike the functional requirements that only had one correct output that could be tested, it was much harder to find tests that ensured these non-functional requirements were fulfilled.

Team

3. It will be necessary for our team to acquire the following skills in order to complete the verification and validation of our project:

    - Proficiency in the testing technologies mentioned in this document (pytest, selenium)

- Knowledge of different types of testing specifically: unit testing, integration testing, usability testing, performance testing and load testing.
- Skills in conducting surveys, interviews and feedback from users to improve the design/usability of the software.
- Project management skills will be necessary to manage our limited time, effectively track results/issues and allocate tasks.

4. Two approaches for acquiring the skills mentioned above are:

   - For proficiency in testing technologies, one way to be proficient is by registering for online courses/tutorials and doing hands-on exercises to apply the knowledge. Another is by going through the documentation for the technologies and making use of community forums like Stack Overflow.
     - The team decided to pursue making use of the documentation available online as it is more efficient and most of the tools are already well documented.
   - For acquiring knowledge of different types of testing, one way to master the skill is by watching online tutorials for each testing type. There are many free resources online and this way, each member can study at their own pace. Another way to master the skill is by discussing amongst ourselves and sharing our knowledge about the subject. This way, our team cohesion and collaboration is enhanced further.
     - The team decided to pursue the latter option since it would likely take less time compared to individually spending time to learn from a course. It also allows sharing of knowledge among the team, peer support, and team problem-solving, ultimately strengthening the team's cohesion.
   - One way to learn how to be effective at writing and conducting surveys is by watching videos on how professionals do surveys. Another is by actually conducting surveys ourselves among our peers and reflecting on what went right and what we could improve on.
     - The team decided to pursue the latter approach as it is more engaging and hands-on learning.

- One way to gain project management skills is by attending courses that go over the fundamentals of project management and methodologies like Agile, Scrum, etc. Another is by gaining experience working on a project.
  - The team decided to pursue using our experience from our internships to discuss and learn from each other about effective project management strategies that we have observed in the workplace and try to implement those in our own projects due to the time constraints of this project.