Project Title: System Verification and Validation Plan for Software Engineering

Team 14, Reach
Aamina Hussain
David Morontini
Anika Peer
Deep Raj
Alan Scott

November 3, 2023

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

symbol	description
VnV	Verification and Validation
V&V	Verification and Validation
SRS	Software Requirements Specification

2 General Information

This document discusses the verification and validation plan for the REACH project. The main goal of this document is to give all stakeholders in the REACH project confidence that the system will be built based on the correct requirements, and that those requirements will be implemented correctly, ensuring the system is functioning as intended.

2.1 Summary

The software being tested is REACH. REACH is a web application that will simplify the process for both patients, and clinicians working on behalf of their patients, of finding clinical trials that they are eligible for and that would be useful for them.

2.2 Objectives

The main objectives of this document are the following:

- Ensure the system has a high level of usability
- Build confidence in the correctness of the searching/matching capabilities
- Build confidence in the security of user data

These three objectives are the most important qualities of the system, and they are the ones that are in scope for this project.

First, usability is a must since the entire purpose of the application is to simplify the process of finding clinical trials. If the system was difficult to use, this process would not be simplified and the application would be useless. Furthermore, the ability to search/match patients to trials is an integral component of the application, and building confidence in that functionality is crucial for all stakeholders. Finally, ensuring the system is safe and secure when it comes to handling user data is of course extremely important for the users of the application which is why it is being prioritized.

2.3 Relevant Documentation

There are a few documents that are worth noting before jumping in to the verification and validation plan. First, the SRS is the most important and relevant document that accompanies the plan (Reach (2023b)). The majority of the test plan is built on top of the requirements specified in the SRS, since as stated previously, one of the main goals of this document is to build confidence in the implementation of the software requirements. Second, the development plan is relevant as it discusses various tools and technologies that can/will be used to test the software system being built Reach (2023a). Finally, the MIS document which details the modular design of the system is especially relevant to the unit test portion of the document.

3 Plan

This section will delineate the roles of the team members as part of V&V efforts. It will also describe the various verification and validation plans for the project.

3.1 Verification and Validation Team

Table 1: V&V Team

Name	Team	Role
Aamina	Dev Team	As a front-end SME, Aamina will be responsible for handling functional and nonfunctional testing for all front-end system capabilities. In particular, Aamina will manage testing and verification pertaining to JS and TS frameworks, automated cloud deployment, graphical interface and presentation.
David	Dev Team	As a back-end SME, David will be responsible for handling functional and nonfunctional testing for all back-end system capabilities. In particular, David will manage testing and verification pertaining to Python frameworks, database management, and API calls.
Anika	Dev Team	As a front-end SME, Anika will be responsible for handling functional and nonfunctional testing for all front-end system capabilities. In particular, Anika will manage testing and verification pertaining to JS and TS frameworks, automated cloud deployment, graphical interface and presentation.
Deep	Dev Team	As the testing SME, Deep will lead the majority of the testing efforts and ensure that the team is meeting the predetermined standards for testing and verification. Deep will focus on both front- and back-end testing and will lend his expertise to the rest of the dev team when they focus on end-to-end testing.
Alan	Dev Team	As a back-end SME, Alan will be responsible for handling functional and nonfunctional testing for all back-end system capabilities. In particular, Alan will manage testing and verification pertaining to Python frameworks, database management, and API calls.
All Developers	Dev Team	All developers will be responsible for writing and verifying unit, integration and end-to-end tests for the system. Additionally, pipelines testing, Dockerfile testing, automated testing and any other deployment-related testing is shared among all team-members.
Dr. Ho	Supervisor	As a supervisor, Dr.Ho will help the team verify the V&V Plan and will ensure that his vision is being tested properly at a high level.
Dr. Scallan	Supervisor	As a supervisor, Dr.Ho will help the team verify the V&V Plan and will ensure that his vision is being tested properly at a high level

3.2 SRS Verification Plan

Once REACH has been created and all requirements have been implemented, the SRS will be verified by the following methods:

- The supervisors will be given the chance to operate the system using precreated inputs to generate outputs that can be compared to the expected ouput.
- The supervisors will be given the chance to give ad-hoc feedback on the system, and will be asked to complete a survey evaluating how well the development team implemented the requirements. The requirements in the survey will be functional and nonfunctional ones that are easy for those without a software engineering background to understand and evaluate.
- The development team will operate the system as well in order to assess the system's performance on all requirements and how well the system meets expected testing outputs. A survey will be filled out by the development team as well, but it will be more detailed and granular.

3.3 Design Verification Plan

As stated in the objectives, the main purpose of REACH is to connect patients looking to be involved in a clinical trial with researchers looking for participants for their research studies. This means the usability of the design is one of the top priorities, as people with different abilities will be using REACH. This can include seniors, or people who are not as adept at using technology. For this reason, having a design which focuses on improving the user's interaction with the web application is vital. This can be done by confirming the usability of the design.

The maintainability of the design is also something that needs to be verified. Due to the fact that REACH is in its early stages, it is likely that new improvements and features will be added in the future. To achieve this, other principles such as anticipation of change, separation of concerns, modularity, and reusability should be accounted for.

3.3.1 Usability

Usability refers to how easily typical users of REACH are able to use the web application. The usability of the design is greatly impacted by its user interface, so we will mainly be focusing on verifying and validating the usability of the user interface. The usability of the design also directly depends on the user's capabilities and characteristics, so it is important that a range of users with different capabilities use REACH in order for us to verify how usable it is.

Our design will be reviewed by Dr. Ho and Dr. Scallan, as they are supervisors, stakeholders, and potential users of REACH. They will select a few other users (around three) with different skill sets and characteristics to also review the design. The reviewers will use the checklist below and keep the checklist items in mind during their review of REACH.

Usability Checklist:

- Text, icons, and formatting are consistent
  - Reduces confusion and allows users to pick up and understand meaning behind certain text and icons quicker
- Only relevant information and instructions are displayed without any clutter
  - Ensures user is not distracted or confused by any extra information
- Visible and prompt feedback is provided to the user after they complete an action
  - The system should visibly respond to both successful and unsuccessful actions made by the user and keep the user informed and updated
- Error handling exists
  - Simple details of what the error is, along with straightforward instructions on how the user can fix the error will cause the user to be less confused and panicked

- Simple descriptions which guide users on what to do next are always available or visible to the user
  - Users should not need to spend time thinking about what action they should take next
- User interactions with the system should have constraints
  - Restricting what the user can select, input, or interact with will limit future errors and issues the user experiences

This review will also be aided by the Useability tests listed under Tests for Nonfunctional Requirements, as well as the Usability Survey Questions.

3.3.2 Maintainability

Maintainability refers to how easily the system can be maintained, including the ease at which it can be improved by fixing any bugs or expanded to include new features. Multiple principles affects the maintainability of a system, including modularity, reusability, and whether we design for change. Having modules which implement the separation of concerns principle helps to pinpoint where an issue or bug might be located. This, along with the modules having low coupling (depending less on other modules), makes it easier to add new modules which map to new features or upgrades. The reusability principle can be implemented by having standardized and generic components which are reused multiple times. This will also make it easier to add new features, as we can reuse the components.

The review will involve code inspections, such as a code walkthrough, where the focus is on if there's a valid separation of concerns, and whether this separation clearly exists within the code. This code inspection will occur regularly throughout the development of REACH. Every member of the team will review and verify whether the design is maintainable. Some members will be focusing on modularity. They will ensure the code is decomposed into modules which contain related information. The rest of the members will focus on reusability. They will confirm that we are creating and using generic components whenever possible.

3.4 Verification and Validation Plan Verification Plan

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.5 Implementation Verification Plan

To test the implementation first we have the tests listed below in section 4 and the unit tests listed in section 5. All of these tests should be run before each major update to the program (revision 0, revision 1, etc). Relevant tests should also be run when the affected requirements are modified during minor updates. Also during minor updates the code should be inspected by at least one other developer using a github pull request. We will also be using static analysis, the specific software for this and other testing is listed in section 3.6 below.

3.6 Automated Testing and Verification Tools

For the backend (i.e., python service(s)), we will be using pytest as our unit testing framework. Additionally, we will be using tavern (an API testing framework built on top of pytest) to test our API. These API tests will ideally run at scheduled intervals (i.e., nightly, weekly... whatever is reasonable for the resources available), which can be done with the use of GitHub actions. Furthermore, we will be using several static analyzers to ensure code quality and maintainability. These static analyzers include flake8 (which ensures the code will follow the pep8 coding standard), mypy (which ensures strict typing in python, and the use of comments+docstrings), and autoflake (which ensures there are no unused packages/imports in a python file). Finally, for analyzing code coverage in our python service(s), we will use coverage.py, which produces useful coverage reports after running tests using pytest.

For the frontend, nothing has changed from the development plan (i.e., using jest for unit testing + code coverage, and ESLint for linting).

3.7 Software Validation Plan

To validate our software system/requirements, there are a few steps that will be taken. First, continuing to meet with our clients (supervisors) while developing the early parts of the application will help ensure the system is on the right track. Furthermore, upon completing of the revision 0 demo, we will demo the system to our clients, enabling them to ask questions, provide feedback, and ensure the system has the correct requirements. At this time we will also encourage our clients to use the platform directly (since it will be deployed, they will be able to easily access it) and give additional feedback post revision 0 demo, which will hopefully bring light to new and improved requirements that can be implemented for revision 1.

4 System Test Description

4.1 Tests for Functional Requirements

The following subsections are split based on the core aspects of the application. These include authentication, which enables a user to sign in to their account, user data, which enables users to save their data for future uses of the application, the emailing system, which will notify users of new trials and give them a template for reaching out to trial clinicians, searching for trials (the main purpose of the app), and general program/system information (i.e., the FAQ).

4.1.1 Authentication

1. AT-1

   Control: Automatic

   Initial State: User is not signed in to the application, and is about to create an account.

   Input: A valid, unique email, and a valid password satisfying the NIST guidelines

   Output: The user's account should be created successfully, and a 201 CREATED response should be returned.

Test Case Derivation: As stated in FR-1 and FR-2, the system must only allow a user to create an account if the email is unique (i.e., not used by some other user yet), and if their password satisfies the NIST guidelines. Since the inputs to this test case satisfy these requirements, the response should be a success.

How test will be performed: Pre define a valid username and password. Automated test to pass in these values and check the return value/output.

2. AT-2

   Control: Automatic

   Initial State: User is not signed in to the application, and is about to create an account.

   Input: An email that is already attached to another account, and a valid password.

   Output: The user's account should not be created and a response stating this failure should be returned.

   Test Case Derivation: From FR-2, the email must be unique. Since it is not unique in this test case, the account should not be created.

   How test will be performed: Pre define an invalid email. Automated test to pass in these values and check the return value/output.

3. AT-3

   Control: Automatic

   Initial State: User is not signed in to the application, and is about to create an account.

   Input: A valid, unique, email, and a password that does not satisfy the NIST guidelines.

   Output: The user's account should not be created and a response stating this failure should be returned.

   Test Case Derivation: From FR-1, the password must abide by the NIST guidelines. Since it does not in this test case, the account should not be created.

How test will be performed: Pre define an invalid password. Automated test to pass in these values and check the return value/output.

4. AT-4

   Control: Automatic

   Initial State: User is not signed in to the application, and is about to log in to the application.

   Input: A valid, unique, email, and a password that satisfies the NIST guidelines, which are tied to an account.

   Output: The user should be logged in to the application.

   Test Case Derivation: From FR-3, if the credentials are correct, a user should be logged in to the application.

   How test will be performed: Pre define a valid username and password tied to an account. Automated test to pass in these values and check the return value/output.

5. AT-5

   Control: Automatic

   Initial State: User is not signed in to the application, and is about to log in to the application.

   Input: An email that is not tied to an account, with a password that satisfies NIST guidelines (but can be any valid password).

   Output: The user should not be logged in to the application.

   Test Case Derivation: From FR-3, a user is only logged in to an account if the credentials are correct. Since the email is not valid, they should not be logged in.

   How test will be performed: Pre define a random email/password. Automated test to pass in these values and check the return value/output.

6. AT-6

   Control: Manual

Initial State: User is not signed in to the application, and would like to reset their password.

Input: A valid email address.

Output: The user should receive an email, containing a link which will allow them to reset their password.

Test Case Derivation: From FR-5, a user must receive an email when attempting to reset their password.

How test will be performed: Manually go on to the login page, enter an email address, and click reset password. Then, verify that the email received is correct, and that the link takes the user to the reset password page.

7. AT-7

   Control: Automatic

   Initial State: User is not signed in to the application, and is about to reset their password.

   Input: A password that is different from the one currently tied to their account, satisfying the NIST guidelines.

   Output: The user's password should be reset, and they should be logged in to the application.

   Test Case Derivation: From FR-5, a user must only be able to reset their password if the password they enter is new, and if it satisfies the NIST guidelines.

   How test will be performed: Pre define a random email/password. Automated test to pass in these values and check the return value/output.

8. AT-8

   Control: Manual

   Initial State: User is signed in to the application and is about to sign out of the application.

   Input: N/A

   Output: The user should be logged out of the application.

Test Case Derivation: From FR-6, a user should be able to log out of the application, with really no constraints.

How test will be performed: Manually sign out of the application, and verify that it works.

9. AT-9

   Control: Manual

   Initial State: User has created an account for the first time, or is logging in as a guest.

   Input: N/A

   Output: The user is prompted to enter their personal data.

   Test Case Derivation: From FR-7, a user is prompted to enter personal information when creating an account or logging in as a guest.

   How test will be performed: Manually create an account and sign in as a guest, and verify the user is prompted.

4.1.2 User Data

1. DT-1

   Control: Manual

   Initial State: User is not signed in, and does not have an account.

   Input: New user data (medical conditions, personal attributes).

   Output: Updated user database displayed on the user profile page.

   Test Case Derivation: From FR-8, the system shall store the information entered by the user in the database.

   How test will be performed: Manually create a new account, and add in some user data and medical conditions. Log back in and verify the data is still present on the user profile.

2. DT-2

   Control: Manual

Initial State: User is signed in with their personal account.

Input: Updated user data (medical conditions, personal attributes).

Output: Updated user database displayed on the user profile page.

Test Case Derivation: From FR-9, a user shall be able to update their personal information.

How test will be performed: Manually update the user profile with new user information, then log out. Log back in and check that the new data is present.

3. DT-3

Control: Manual

Initial State: Tester is not logged in.

Input: Database query.

Output: List of non-unique unique database elements.

Test Case Derivation: From FR-8, the system shall store the information entered by the user in the database.

How test will be performed: Manually query the database for columns that are supposed to contain unique values, namely keys, such as user ID, email address, and etc. Verify that the length of the returned set is 0.

Rationale: To ensure that key constraints are upheld, we must ensure that the values are non-unique, otherwise querying for certain users may return multiple sets of data.

4.1.3 Email

1. ET-1

Control: Manual

Initial State: The user has searched for trials, and has been given a list of trials.

Input: N/A

Output: A customized email template.

Test Case Derivation: From FR-10, the system shall create an email template for the user.

How test will be performed: Manually select a given trial result, and select the option to generate a template. Verify that it is displayed to the user.

2. ET-1

Control: Automatic

Initial State: The user is not logged in to the system.

Input: N/A

Output: Email notification notifying user of available trial.

Test Case Derivation: From FR-11, the system shall notify the user when new medical trials matching their specifications are available.

How test will be performed: A test user will be created with their own email account, with common medical conditions. The email will be checked periodically for notifications.

4.1.4 Program Information

1. PT-1

Control: Manual

Initial State: The user is at the home screen.

Input: N/A

Output: FAQ menu.

Test Case Derivation: From FR-12, the system shall include frequently asked questions and trial information.

How test will be performed: Manually select the FAQ menu, and verify that it displays the FAQ.

4.1.5 Searching

1. ST-1

   Control: Manual

   Initial State: The user is signed is, either as themselves or as a guest.

   Input: Medical conditions and other user attributes used to search.

   Output: Displayed search results.

   Test Case Derivation: From FR-13, FR-14, a user shall be able to search for medical studies based on their health conditions, and the system shall display to the users the results of their trial search.

   How test will be performed: Manually enter the a set of medical conditions and physical attributes into the search function. Submit, and verify the results are returned.

2. ST-2

   Control: Manual

   Initial State: The user is signed into their personal account, with medical information saved to their account.

   Input: N/A

   Output: Displayed search results.

   Test Case Derivation: From FR-15, the system shall allow a signed-in user to automatically enter their conditions into the search parameters.

   How test will be performed: Manually navigate to the search function. Verify that their conditions and information is automatically inputted into the search fields.

3. ST-3

   Control: Manual

   Initial State: The user is logged in, either as themselves or as a guest, and is at the search function.

   Input: N/A

Output: N/A

Test Case Derivation: From FR-15, the system shall allow a signed-in user to automatically enter their conditions into the search parameters.

How test will be performed: Attempt to search without any fields filled in. Verify that the search fails.

4.2 Tests for Nonfunctional Requirements

4.2.1 Useability

1. UT-1

   Control: Dynamic, Manual

   Initial State: The user starts on the home screen.

   Test Case Derivation: From NFR-3, NFR-4, and NFR-5, regarding the learnability, simplicity, and legibility of the user interface.

   How test will be performed: Have the user sign in, search, and browse results without outside help. 95% of users should be able to accomplish this.

2. UT-2

   Control: Dynamic, Manual

   Initial State: The user starts on the home screen.

   Test Case Derivation: From NFR-3, NFR-4, and NFR-5, regarding the learnability, simplicity, and legibility of the user interface.

   How test will be performed: Have the user sign in, search, and browse results without outside help. Have the user then complete a survey(Usability Survey Questions)

3. UT-3

   Control: Dynamic, Manual

   Initial State: The user starts on the home screen. The selected language should be set to the default (English).

   Test Case Derivation: From NFR-17, regarding accessibility of the application users who do not speak an official language.

How test will be performed: Users of the next five most spoken languages other than English and French should be selected. They should be instructed to change to their preferred language and then should login, search, and browse results. 95% of users should not report any issues navigating the interface.

4.2.2 Performance

1. PT-1

   Type: Dynamic, Manual

   Initial State: User has entered their information for their and is about to search for eligible trials.

   Input/Condition: All possible information filled in.

   Output/Result: The system should load the trials and display them to the user in less than 5 seconds.

   How test will be performed: In an existing account with all information categories filled in by the user, manually search for eligible trials, and keep track of how long it takes for the trials to load.

2. PT-2

   Type: Dynamic, Manual

   Initial State: N/A

   Input/Condition: API key available to authenticate against the API.

   Output/Result: API calls should respond in less than 1 second.

   How test will be performed: Using locust, simulate 1000 users concurrently searching for trials, and manually track the max time recorded for a response from the API.

4.2.3 Operational and Environmental

1. OT-1

   Type: Automatic

Initial State: N/A

Input/Condition: Database is populated with running trials

Output/Result: The system should show the number of trials within an order of magnitude to the trials on ClinicalTrials.gov

How test will be performed: Count the number of running trials in the database and compare to the number of running trials on ClinicalTrials.gov

2. OT-2

Type: Dynamic, Manual

Initial State: User is on home screen, on a variety of devices

Output/Result: The user should report satisfactory performance on all devices (¿7)

How test will be performed: Give the user multiple devices from the last 5 years (ex. iphone 15, windows laptop, chromebook, etc) and ask them to rate the performance of the application compared to a desktop pc from 1-10.

4.2.4 Maintainability

1. MT-1

Type: Static, Manual

How test will be performed: Manually run linters on code before merging a PR.

Rationale: This will ensure code is following the proper standards, which will help increase the maintainability of the system. Also, this will be performed manually for the reason of conserving resources (i.e., could setup some workflow to automatically lint code every time some code is pushed to the GitHub Repo, however this could be costly).

4.2.5 Security

1. ST-1

2. ST-2

4.2.6 Legal

1. LT-1

   Control: Dynamic, Manual

   Initial State: The user is at the search screen.

   Input: User search criteria.

   Output: Option prompting user for consent.

   Test Case Derivation: From NFR-18, regarding the collection of user consent before using or storing user data.

   How test will be performed: Users are asked to input their search criteria, and should see a message prompting them to provide consent for the collection of data when they try to search.

4.3 Traceability Between Test Cases and Requirements

Table 2: Test case Traceability

Test case	Requirement(s)
AT-1	FR-1, FR-2
AT-2	FR-2
AT-3	FR-1
AT-4	FR-3
AT-5	FR-3
AT-6	FR-5
AT-7	FR-5
AT-8	FR-6
AT-9	FR-7
DT-1	FR-8
DT-2	FR-9
DT-3	FR-8
ET-1	FR-10
ET-2	FR-11
PT-1	FR-12
ST-1	FR-13, FR-14
ST-2	FR-15
ST-3	FR-15
UT-1	NFR-3, NFR-4, NFR-5
UT-2	NFR-3, NFR-4, NFR-5
UT-3	NFR-17
PT-1	NFR-6
PT-2	NFR-7
OT-1	NFR-11
OT-2	NFR-12
MT-1	NFR-13
ST-1	NFR-15
ST-2	NFR-16
LT-1	NFR-18

5 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]

5.1 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]

5.2 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]

5.2.1 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. ...

5.2.2 Module 2

...

5.3 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]

5.3.1 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:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

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

References

Reach. Development plan. https://github.com/davimang/REACH/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf, 2023a.

Reach. System requirements specification. https://github.com/davimang/REACH/blob/main/docs/SRS/SRS.pdf, 2023b.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions

Usability Survey:

How easy was it to navigate to the search page (1 is hard, 10 is easy)?
How easy was it to understand each search term (1 is hard, 10 is easy)?
Did you find any part of the process frustrating(Y/N)?
If yes, which part did you find frustrating and why?
Was all text easy to read while using the website(Y/N)?

Appendix — Reflection

Below are skills that the team will need to develop in order to successfully carry out the verification and validation process, in addition to how the team plans to further our knowledge in the given areas.

- Automated unit testing via integrations
  - Free tutorials on Youtube or other free video sharing platforms.
  - Documentation on the given cloud platform's integrations.
- Verification and validation reporting standards
  - Lecture material and other course material, from both capstone and other courses.
  - Documentation and templates from IEEE.

For the automated unit testing, the team has decided to pursue video tutorials, as these tend to be easier to approach, and provide practical examples. For the VnV reporting, the team has decided to use course materials, as this is readily accessible and easy to understand.

Additionally, below is a skill that each member will need to develop, along with two ways they can learn each skill.

- Alan - GitHub CI/CD pipelines
  - YouTube tutorials or video tutorials from other sites.
  - GitHub documentation.
- David - Jest testing framework
  - Jest documentation
  - YouTube videos/tutorials
- Deep - Pytest testing framework
  - Course website
  - YouTube videos/tutorials
- Anika - Working with automated testing frameworks in pipelines

  - Pytest and Jest documentation
  - Pluralsight tutorials
- Aamina - Pytest testing framework
  - Pytest documentation
  - YouTube videos/tutorials

Below is which method each member intends to use, and why.

- Alan - YouTube tutorials. This method is the most approachable, and provides easy to follow along examples.
- David - Jest documentation. I will use the documentation since I have used several other testing frameworks in previous projects, and I anticipate the documentation should provide sufficient information to get up and running with the framework.
- Deep - Course Website. This method will allow me to learn about pytest is a more formalized scenario. This will help with writing tests according to industry standards.
- Anika - Pluralsight tutorials. I will use this method because it will help me learn about the frameworks in a more structured way with in-depth practice and examples.
- Aamina - YouTube videos/tutorials. Since I am new to Pytest and have previously only worked with Java and the Junit testing framework, it would be best to watch videos with simple explanations and get a walkthrough of different examples.