Project Title: System Verification and Validation Plan for Software Engineering

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

November 2, 2023

Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

[Say what software is being tested. Give its name and a brief overview of its general functions. —SS]

2.2 Objectives

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: "build confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

2.3 Relevant Documentation

[Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. —SS]

Author (2019)

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

3.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may include ad hoc feedback from reviewers, like your classmates, or you may plan for something more rigorous/systematic. —SS]

[Maybe create an SRS checklist? —SS]

3.3 Design Verification Plan

[Plans for design verification —SS]

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

[Create a checklists? —SS]

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

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

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

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Authentication

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

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: Mnaually create an account and sign in as a guest, and verify the user is prompted.

4.1.2 Area of Testing2

...

4.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

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, 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. 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

Title for Test

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 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).

4.2.4 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

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

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

Author Author. System requirements specification. https://github.com/..., 2019.

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?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

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

Appendix — Reflection

[This section is not required for CAS 741 —SS]

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

1. 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 etc. You should look to identify at least one item for each team member.

2. 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?