System Verification and Validation Plan for Room8

Team 19
Mohammed Abed
Maged Armanios
Jinal Kasturiarachchi
Jane Klavir
Harshil Patel

2024-10-31

Revision History

Date	Version	Notes
2024-10-31	0.0	Document Created

[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

List of Figures

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (**?**) 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 Challenge Level and Extras

[State the challenge level (advanced, general, basic) for your project. Your challenge level should exactly match what is included in your problem statement. This should be the challenge level agreed on between you and the course instructor. You can use a pull request to update your challenge level (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

[Summarize the extras (if any) that were tackled by this project. Extras can include usability testing, code walkthroughs, user documentation, formal proof, GenderMag personas, Design Thinking, etc. Extras should have already been approved by the course instructor as included in your problem statement. You can use a pull request to update your extras (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

2.4 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. You can create BibTeX entries for your documents and within those entries include a hyperlink to the documents. —SS]

?

[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] The following section outlines the execution plan for verifying project documentation and validating the software. The sections begins outlining the verification and validation team. Following section 3.1, sections 3.2 till 3.4 outline the plans for verifying documentation, while sections 3.5-3.7 discuss verification and validation for the system.

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]

Table 1: **Verification and Validation Team**

Teammate	**Position**	**Project Role**
Mohammed Abed	Developer	Mohammed's role will be focused on automating test cases using online tools such as GitHub actions and any other tools the team chooses to use. Since Mohammed's strengths and past experiences are in web development, he will also provide support in developing tests for the user-facing application.
Maged Armanios	Developer	Maged's role will focus on developing unit and usability tests for the user facing application and it's backend. These tests will be then automated by Mohammed using tools like GitHub Actions.
Jinal Kasturiarachchi	Developer	Jinal will verify usability requirements for the system.
Jane Klavir	Developer	Jane's role will have her verify the cleanliness management system meets the requirements outlined. This includes that it'll be able to detect created messes, cleaned messes, and states of no change.
Harshil Patel	Developer	Harshil will focus on testing the safety and security of the system. This includes ensuring all data is stored securely, access tokens are confidential, and authorization is secure on the user's end.
Dr. R Tharmarasa	Supervisor	Dr. Tharmarasa's role will be to provide the team with support on making decisions related to the cleanliness management algorithm. Additionally, Dr. Tharmarasa will provide his input on the feasibility of requirements related to the cleanliness management system using his research experience in classification and related topics.

Note about project roles: While each member of the team does has specified project roles, all members of the team are expected to be able to support each other when required. The roles defined on the table above are simply the optimal roles for each member based on their expertise and does not restrict the work they are able to undergo to only what's outlined in the table above.

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 (like your primary reviewer), or you may plan for something more rigorous/systematic. —SS]

[If you have a supervisor for the project, you shouldn't just say they will read over the SRS. You should explain your structured approach to the review. Will you have a meeting? What will you present? What questions will you ask? Will you give them instructions for a task-based inspection? Will you use your issue tracker? —SS]

[Maybe create an SRS checklist? —SS]

To verify our SRS, the development team will have a meeting with project supervisor Dr. Tharmarasa. In this meeting, the group will present a summarized version of the SRS which outlines the components, functionality, constraints, and assumptions. This presentation will be accompanied some form of presentation of the use-case scenarios outlined in the SRS. This can be done with either a low-fidelity mock up or a live-demo of the system if sufficient development has been completed. After the presentation, we will ask the supervisor whether they found any use cases unimportant and requirements unverifiable. After meeting with our supervisor, the team will redraft the requirements and use cases according to his feedback.

In the event we are unable to meet with our supervisor for review, an alternative approach would be to seek peer review and feedback from our primary project reviewers. The meeting would have a similar structure to that which would be done with Dr. Tharmarasa.

Additionally, the development team will review the SRS document to verify each requirement:

- Is free of implementation details
- Does not include acceptance criteria
- Does not conflict with any other requirement

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]
The Verification and Validation Plan will be read by each member of the team to confirm the following:

- Each teammember referenced has clear, defined roles with a clause for role switching and overlapping roles
- Each test satisfies the following:
  - The test corresponds to a requirement in the SRS
  - Functional requirements have test cases specific enough that a test can be built with the data. Including clearly defined inputs and outputs
  - Functional test cases are deterministic
  - Nonfunctional test cases are specific and provide enough information that someone can run the test once the product is built.

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]

[The final class presentation in CAS 741 could be used as a code walkthrough. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

The system implementation will be verified using a plethora of tools and techniques. First, the implementation of the system will be verified using the system tests and unit tests outlined in sections 4 and 5. Additionally, the development team will create integration tests to see if the interactions between different modules is expected. Any tests that can be used during CI/CD using will be done with them to ensure changes to the source code do not create a loss functionality. Finally the team will perform an acceptance test, running the application and acting out the key use-case scenarios outlined the SRS.

During development, the team will use static analysis tools to help minimize common development mistakes such as referencing non-existent variables, incorrect type operations, etc.

Additional tests that can be executed if required are:

- Code walkthroughs with the development team
- Review sessions with our team's primary reviewers

3.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

At the time of writing, the programming languages have not been finalized. Nonetheless, this section outlines the expected tools that will be utilized for the expected programming languages and frameworks. Currently, the expected programming languages and frameworks are expected to be JavaScript/TypeScript for the frontend application, utilizing the React framework, and Python for the cleanliness management system.

TypeScript:

- Jest: The preferred framework for testing React-based applications
- Puppeteer: Provides high-level APIs to control Chrome and allows for automating UI testing, form submissions, and keyboard inputs easily
- ESLint: A code linter that helps catch common programming mistakes and can be enhanced to enforce code styling. Automates the detection of potential vulnerabilities and life-cycle methods for React based projects.

Python:

- Pylance: A language server, providing tools features like intelliSense, logical linting, code actions, code navigation, and semantic colourization. These features can assist developers in catching errors while developing.
- Flake8: A command-line utility for enforcing style consistency across Python projects. Includes lint checks.
- Pytest and PyUnit: Testing frameworks.

CI/CD Tools:

- GitHub Actions: Allows for the automation of builds and tests

API Testing Tools:

- Postman: Can run API functional and performance tests to determine average and median API response times. Can also be used to ensure the correctness of an API's response

3.7 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

4 System Tests

[There should be text between all headings, even if it is just a roadmap of the contents of the subsections. —SS]

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 Area of Testing1

[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. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs. Output is not how you are going to return the results of the test. The output is the expected result. —SS]

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

   How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   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:

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]

[For some nonfunctional tests, you won't be setting a target threshold for passing the test, but rather describing the experiment you will do to measure the quality for different inputs. For instance, you could measure speed versus the problem size. The output of the test isn't pass/fail, but rather a summary table or graph. —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]

[If you introduce static tests in your plan, you need to provide details. How will they be done? In cases like code (or document) walkthroughs, who will be involved? Be specific. —SS]

4.2.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2.2 Area of Testing: Authentication and House Management

This section covers encryption standards for authentication and house management data of the application, as presented in SRS, Section S.2.1.

Encryption of Authentication Data

1. test-encryption-01

   Type: Automatic.

   Initial State: Application open with user ready to login.

   Input/Condition: User inputted data in login elements.

   Output/Result: All data in transit and stored is encrypted.

   How test will be performed:

   1. Capture data in transit between client and server.
   2. Verify data in transit encrypted using network monitoring tools.
   3. Verify database is using encrypted storage type.

2. test-security-01

   Type: Manual.

   Initial State: User has application open on login screen.

   Input: Incorrect login credentials.

   Output: Error message that avoids revealing sensitive information.

   How test will be performed:

   1. Attempt to login with incorrect password but correct email.
   2. Observe error message displayed.

...

4.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

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]

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

[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.

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?