System Verification and Validation Plan for Software Engineering

Team #22, TeleHealth Insights
Mitchell Weingust
Parisha Nizam
Promish Kandel
Jasmine Sun-Hu

November 1, 2024

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 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]

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 (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]

The following approaches will be used to verify the SRS:

- **Ad hoc Peer Review**: Informal reviews of the SRS will be conducted after every major revision to the SRS with classmates who will serve as peer reviewers. This will provide new perspectives that can help identify ambiguities, missing requirements and/or areas of improvement.
  - Peer reviewers will submit feedback using GitHub issue tracker for organized task assignment and tracking.
- **Supervisor Review**: Before every major SRS revision submission, the team will send a copy of the SRS to the project supervisor a week before meeting, along with a checklist highlighing priority areas for their feedback. The team will also prepare questions about the requirements related to interfaces and usability which is the supervisor's area of expertise. During the review meeting, the team will first review the supervisor's initial feedback with the supervisor, and then ask the prepared questions.
  - The meeting notes will be documented using GitHub issue tracker.
- **Internal Team Walkthrough**: The team will hold a collaborative session before every major SRS revision submission where the team will discuss each section one by one to verify that all requirements are understood by all members, and that each section is consistent with the project goals/objectives. A checklist will act as a guide to highlight the key concepts the review should focus on, and will help ensure no critical areas are missed in review. The checklist can be found below.
  - Any corrections or modifications that need to be made will be noted in the team meeting's GitHub issue tracker.

The following checklist will be used for the internal team walkthrough:

- ☐ Are all major functions required for the website (interface, backend, recording, analysis, storage) covered?
- ☐ Does each function have clear input and output specifications?
- ☐ Are all requirements written with consistent terminology?
- ☐ Do all requirements avoid conflict with each other?
- ☐ Does each requirement avoid ambiguous language?
- ☐ Are all requirements verifiable and testable?
- ☐ Is each requirement written in a way all teawm members and stakeholders can understand?
- ☐ Are any assumptions about user behavior or system behaviour explicity stated?

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]

[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]

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]

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 Data Processing and Display

This set of test cases will help confirm the system's data retrieval, report generation, and display functionalities to ensure the clinician experience aligns with the project's goals.

- FR-ST-DPD1

Control: Automatic

Initial State: Database is populated with processed assessment data.

Input: Query request for a specific patient's processed assessment data.

Output: The expected result is the successful retrieval of all relevant assessment data, displayed without errors within 10 seconds

Test Case Derivation: Ensures the system can retrieve data efficiently for report generation, meeting retrieval speed and completeness requirements.

How the test will be performed:

1. Query the database for a test patient's processed assessment results.
2. Measure and record retrieval time, ensuring it does not exceed

10 seconds.
3. Verify that all required data is retrieved and matches the stored information.

- FR-ST-DPD2

Control: Automatic

Initial State: Database has assessment data including flagged occurrences, timestamps, and patient performance metrics.

Input: Trigger for report generation based on a retrieved assessment dataset.

Output: The expected result is a generated report containing all required data within 10 seconds.

Test Case Derivation: Confirms that report generation is complete, accurate, and within performance constraints.

How the test will be performed:

1. Retrieve a patient's assessment data from the database.
2. Trigger report generation.
3. Confirm the report includes flagged occurrences, timestamps, and performance metrics.
4. Measure and confirm report generation time does not exceed 10 seconds.

- FR-ST-DPD3

Control: Manual

Initial State: Generated report available in the database.

Input: Clinician dashboard query to display the generated report.

Output: Report displayed in the clinician's dashboard with accurate formatting, charts, and tables, fully loaded within 10 seconds.

Test Case Derivation: Validates the report display function, ensuring usability and speed requirements are met.

How the test will be performed:

1. Query the clinician's dashboard to load the report.
2. Confirm that the report is displayed with correct charts, tables, and formatting.
3. Verify full loading of the report within 10 seconds.

- FR-ST-DPD4

Control: Manual

Initial State: Database has stored reports for previous sessions, each with a unique patient identifier.

Input: Clinician request to access a specific previously generated report.

Output: The expected result is successful retrieval and display of the requested report without errors, within 10 seconds.

Test Case Derivation: Ensures that clinicians can reliably access and view past reports, supporting longitudinal patient assessment.

How the test will be performed:

1. Query the database for a stored report using a unique patient identifier.
2. Verify that the retrieved report is complete and accurate.
3. Confirm that the report is displayed within 10 seconds.

4.1.2 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:

4.1.3 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 Look and Feel Requirements

These test cases ensure that all appearance and style requirements are addressed effectively, covering navigation, user-friendliness, brand consistency, visual appeal, and responsiveness.

- LF-ST-LFR1 (covers LF-AR1, LF-AR2, LF-AR4)

  Type: Dynamic

  Initial State: Platform initialized and navigable on a test device, prepared for user testing with adults and children.

  Input/Condition: Conduct user tests with participants performing core tasks like starting an assessment, navigating menus, and viewing results.

  Output/Results: Expected results include:

  - No more than three levels of navigation depth, and each screen presents no more than six main options.
  - 100% of users complete core tasks within five clicks.
  - 95% of children aged 6-12 can complete a sample assessment independently.

  How the test will be performed:

1. Observe and record the number of clicks taken by each user to complete key tasks.
2. Inspect the navigation structure to ensure it meets depth and option constraints.
3. Test with children to verify completion of assessments independently.

- LF-ST-LFR1 (covers LF-AR3, LF-AR5, LF-SR1, LF-SR2)

Type: Static and Dynamic

Initial State: Platform with finalized color schemes, fonts, and brand assets loaded and available for inspection and user interaction tests.

Input/Condition: Perform visual inspection and feedback collection, along with response-time measurements for interactive elements.

Output/Results: Expected results include:

- Compliance with brand guidelines, style guide, and use of no more than three calming, neutral/pastel tones.
- Positive feedback from child participants indicating a calm, non-stressful environment.
- 100% consistency in design across all pages.
- 100% of user interactions provide immediate feedback within 0.5 seconds.

How the test will be performed:

1. Compare platform's design elements against the client's brand and style guidelines.
2. Collect feedback from child participants about the visual atmosphere.
3. Test and measure feedback response times for various interactions.

4.2.2 Maintainability and Support Requirements

These test cases ensure the platform meets its maintenance, support, and adaptability requirements effectively.

- MS-ST-MSA1 (covers MS-MR1, MS-SR1)

  Type: Static and Dynamic

  Initial State: Modular platform architecture with access to the component's source code. A direct link to GitHub is also available on the platform.

  Input/Condition: Perform updates to individual components and simulate user feedback submissions via the GitHub repository.

  Output/Results: Expected results include:

  - Each component update does not exceed 10 lines of code edited outside the updated module.
  - Users can submit issues and feature requests directly to GitHub, categorized as issues, feature requests, or feedback.

  How the test will be performed:

  1. Perform code updates on isolated components and verify changes are contained within 10 lines outside the component.
  2. Test submission flow to GitHub, verifying links are accessible and that issues and requests are categorized correctly.

- MS-ST-MSA2 (covers MS-SR2)

  Type: Dynamic

  Initial State: Platform initialized with a tutorial accessible from the homepage.

  Input/Condition: New user group follows the tutorial to complete primary tasks (e.g., starting an assessment).

  Output/Results: Expected results include:

  - 90% of users can complete core tasks correctly after following

the tutorial.

How the test will be performed:

1. Guide users through the tutorial and observe task completion rates.
2. Collect feedback on tutorial clarity and assess if 90% of users can independently complete tasks.

- MS-ST-MSA3 (covers MS-AR1)

Type: Dynamic

Initial State: Platform accessible across various devices and screen sizes, from mobile (4") to desktop (27").

Input/Condition: Load and navigate the platform across multiple devices to evaluate responsive design and functionality.

Output/Results: Expected results include:

- 100% of essential features are fully functional and readable across all screen sizes tested.

How the test will be performed:

1. Access the platform on various screen sizes and test for display, layout, and functionality.
2. Verify that all features are usable and adapt responsively without readability or functionality loss.

4.2.3 Cultural Requirements

These tests ensure that the platform respects cultural sensitivities and provides full bilingual support, enhancing inclusivity and accessibility for diverse user groups.

- CU-ST-CUR1

Type: Static and Dynamic

Initial State: Platform content (language and imagery) is finalized and presented for review.

Input/Condition: A cultural consultant reviews all language and imagery, and user acceptance testing gathers feedback from a diverse set of users.

Output/Results: Expected results include:

- 100% of reviewed content is confirmed as culturally sensitive with no instances of offensive language or imagery.

How the test will be performed:

1. A cultural consultant examines all text and imagery for potential cultural insensitivity.
2. Conduct user acceptance testing with a diverse group and gather feedback on the platform's cultural sensitivity.
3. Validate that all feedback confirms no culturally insensitive content.

- CU-ST-CUR2

Type: Static

Initial State: Platform is available in both English and Mandarin, with all interface elements and assessments translated.

Input/Condition: Switch between language settings to review each text, instruction, and assessment in both languages.

Output/Results: Expected results include:

- 100% of content is fully translated and functional in both English and Mandarin with no untranslated elements.

How the test will be performed:

1. Navigate through the platform in both English and Mandarin settings, verifying translations for each section.
2. Confirm that all assessments, instructions, and interface elements are accurately translated and free from language discrepancies.

4.2.4 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:

4.2.5 Area of Testing2

...

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]

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

[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?