System Verification and Validation Plan for Software Engineering

Team 21, Alkalytics
Sumanya Gulati
Kate Min
Jennifer Ye
Jason Tran

November 4, 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

This section outlines the Verification and Validation (VnV) responsibilities of each team member and the team supervisor. It presents comprehensive verification plans for the Software Requirements Specification (SRS), design documents (including MIS, MG, and System Design), the VnV plan itself, and the system implementation.

Additionally, the automated testing and verification tools intended for use in the VnV process have been specified. Verification tasks will be carried out by the VnV team in a sequential manner as each phase progresses, with continuous verification of the SRS and design documents as source code is developed. Software validation will be conducted in parallel with other VnV activities, supporting the Proof of Concept (PoC) and final demonstration phases.

3.1 Verification and Validation Team

In this section, Table 1 presents the members of the VnV team designated to carry out the tasks specified in this document. For each team member, their role in the project's verification process is outlined, with key details highlighting their respective responsibilities.

Table 1: Validation and Verification Team Members and Responsibilities

Team Member	**Role and Responsibilities**
Bassel Abdelkader	Advisor and one of the primary reviewers of documentation; contributor to the validation of user-based testing, providing suggestions and feedback to improve software functionality and user experience.
Dr. Charles de Lannoy	Advisor and one of the primary reviewers of documentation; contributor to the validation of user-based testing, providing suggestions and feedback to improve software functionality and user experience.
Jason Tran	Review the work of other team members to uphold high standards, provide suggestions for improvement, and maintain feedback checklists for each work item, with an emphasis on backend code.
Jennifer Ye	Review the work of other team members to uphold high standards, provide suggestions for improvement, and maintain feedback checklists for each work item, with an emphasis on frontend code.
Kate Min	Review the work of other team members to uphold high standards, provide suggestions for improvement, and maintain feedback checklists for each work item, with an emphasis on unit testing.
Sumanya Gulati	Review the work of other team members to uphold high standards, provide suggestions for improvement, and maintain feedback checklists for each work item, with an emphasis on the documentation and VnV Plan.
Other Design Teams	Peer reviewers identify issues and offer feedback and suggestions for improving documentation.

3.2 SRS Verification Plan

The Software Requirements Specification (SRS) is essential for defining the project scope and guiding implementation. To ensure it aligns with project standards and objectives, a rigorous verification plan will be implemented, incorporating iterative feedback from peers and our assigned Teaching Assistant, Chris Schankula.

The verification process will begin with an initial review to evaluate the SRS for clarity, completeness, and alignment with project goals. Feedback will be systematically gathered and categorized to identify areas needing refinement. Exploratory assessment techniques will allow reviewers to interact with application prototypes and design mockups, ensuring requirements are realistically translated into application features.

3.2.1 Functional Testing

Functional testing based on the SRS will validate application behavior, confirming that each function performs according to specified requirements. The feedback will be integrated into the SRS, leading to revisions that enhance clarity and accuracy. A comprehensive final review will ensure all feedback has been effectively addressed, solidifying the SRS as a true reflection of the intended system.

3.2.2 Formal and Ad-hoc Reviews

The verification approach includes formal and ad-hoc feedback methods. Formal reviews will create updated checklists based on existing ones and grading feedback, while review meetings with the supervisor will identify mistakes and suggest improvements. Ad-hoc peer reviews from other design teams will also provide valuable insights. Key verification checks will ensure the adequacy, feasibility, and verifiability of requirements, along with traceability to use cases.

3.2.3 Checklist

An initial verification plan checklist will be maintained and updated over time, focusing on detailed descriptions, relevance, traceability, verifiability,

and feasibility of requirements, as well as closing all reviewer-identified issues. By leveraging diverse insights, the SRS will be refined to facilitate a smooth transition into the development phase, laying a strong foundation for the project's success.

The following checklist has been created as an initial draft that will be iteratively updated as SRS reviews are completed over time:

- ☐ Does each functional requirement have a clear and precise description?
- ☐ Are the rationales for each requirement clearly articulated?
- ☐ Is there a defined fit criterion for each requirement that specifies success criteria?
- ☐ Are all functional requirements complete and unambiguous?
- ☐ Are all requirements consistent with one another, avoiding conflicts?
- ☐ Are all dependencies on external systems or components identified and addressed?

3.3 Design Verification Plan

This section delineates the strategies and procedures the team will employ to verify the correctness and reliability of the design of the Alkalytics application. This plan will serve as a guideline during the testing phase to ensure that the design aligns with the intended requirements and effectively mitigates potential hazards identified by the VnV team.

3.3.1 Document Review

After the completion of the initial draft of the design documentation (including the Management Guide (MG), Management Information System (MIS), and System Design documents), each member of the testing team will conduct a comprehensive review before submission. The objectives of this review process are to:

- Ensure that the system design aligns with all functional and non-functional requirements.

- Assess the accuracy of the documentation in describing the intended functionality and behavior of the system.
- Record and report any design elements that deviate from specified requirements for further discussion.

3.3.2 Review Meetings

A structured review meeting will be conducted with the supervisors once the design documents are completed. Additionally, peer reviews from classmates will provide critical suggestions for improvement.

3.3.3 Code Conformity Verification

The team will verify that the code adheres to SOLID design principles, ensuring modular and maintainable code structures.

3.3.4 Formal Team Review

A formal review with team members will occur after initial document creation, allowing for reflection on the design prior to final review. Checklists will be utilized to compare the design documents against the Software Requirements Specification (SRS) post-verification.

3.3.5 Checklist

This checklist will be updated as reviews progress to ensure comprehensive verification of the design documents:

- ☐ Are all requirements (functional and non-functional) traceable to at least one implementing module in the MG?
- ☐ Have all issues raised by reviewers been addressed and resolved?
- ☐ Do all modules and components conform to the SOLID design principles?
- ☐ Are all modules assigned unambiguous tasks with well-defined inputs and outputs?
- ☐ Is the design reflective of the database structure and functionality required for the application?

3.4 Verification and Validation Plan Verification Plan

The Verification and Validation (VnV) plan, i.e., this document, must also have a verification plan to ensure its correctness, completeness, and feasibility. The following methods will be applied to verify the document:

- **Internal document review**: All team members review each section to ensure quality and provide feedback/suggestions for improvement.
- **Peer review by classmates**: Another team reviews the contents of the V&V plan and provide feedback/suggestions for improvement.
- **Feedback integration**: The team refines the VnV plan appropriately after reviewing feedback received from internal reviews, peer reviews, and the grading Teaching Assistant (TA).
- **Mutation testing**: The team performs mutation testing by injecting mutations (i.e., faults) into the tests to evaluate whether the tests can detect the mutant and verify if its behaviour is the expected outcome.

The following checklist serves as a guide for verifying the VnV plan:

- ☐ Each verification plan is complete, feasible, and unambiguous.
- ☐ Roles and responsibilities for verification are explicitly and clearly defined.
- ☐ All requirements are covered by the test cases.
- ☐ Each test specifies clear inputs, expected outputs, and criteria for pass/fail.
- ☐ Test procedures are well-described.
- ☐ A traceability matrix is provided, mapping each test case to their relevant requirement(s).

3.5 Implementation Verification Plan

Both static and dynamic techniques will be employed to verify the implementation for the Alkalytics project, as outlined below:

- **Static Analyzers**: Linting tools will be used for static code analysis to identify bugs or stylistic errors, maintain code readability, and enforce coding standards. Specific tools to be used are listed in Section 3.6.

- **Code Inspection/Walkthroughs**: New features or significant code changes will be reviewed by at least two other team members before merging into the main branch. The primary developer may walk the reviewers through the code. Code reviews will be conducted through pull requests on GitHub; only reviewed and approved code may be merged into the main branch. This process ensures code quality, readability, functionality, and allow for mutual understanding of all system components among the team members.

- **System Testing**: System testing will verify that the application meets both functional and non-functional requirements as specified. Details regarding system testing procedures and planned test cases are outlined in Section **??** of this document.

- **Unit Testing**: Unit testing will verify the behaviour of the application's individual components to ensure they perform as expected. The unit testing plans in Section 5 will be developed upon completion of the design specification.

3.6 Automated Testing and Verification Tools

Sections 10 and 11 of the Development Plan discuss the tools to be used and coding standards the team will adhere to. Below is an expansion on the referenced sections:

- **Linters**: Flake8 (for Python) and ESLint (for JavaScript) are the linting tools the team will use, as outlined in Section 10 of the Development Plan. These tools will help maintain code quality by automatically detecting syntax errors, code inconsistencies, and potential bugs.

- **Unit Testing Frameworks**: The team will use Jest for front-end unit testing and Pytest for back-end unit testing.

- **Continuous Integration (CI)**: CI will be implemented using GitHub Actions, automating the building and testing processes throughout development. The team currently has a workflow that runs on all LaTeX

documentation, and additional workflows for automated testing will be added as development progresses.

- **Code Coverage Metrics**: Code coverage will be assessed using the appropriate plugins/flags for the respective unit testing frameworks mentioned above (e.g., `pytest-cov` for Pytest, `--coverage` in Jest).

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

This section covers all tests for different areas of the system.

4.1 Tests for Functional Requirements

The subsections below covers each major functional of the application, from uploading data to the various outputs of post query analysis. Many of the functions come from the same user flow. By testing the subsections below guarintees that all possible user flows involving the main application functioanlities are working as expected.

4.1.1 Data Input and Storage

The tests below provide a way to evaluate the correctness of data input and storagee for the following functional requirements:

- FR-1
- FR-2
- FR-3
- FR-4

1. FR-ST1

   Control: Manual

   Initial State: Database is running and ready to intake data

   Input: Dataset to be stored, in .CSV format

   Output: Data that is inputted is sent into the system, then labelled and stored successfully

   Test Case Derivation: When the data is sent to the system, the data should be stored somewhere and labelled properly before it can be queried.

   How test will be performed: The test can be performed by sending a test sample of varying sizes to the storage in the system.

4.1.2 Data Querying and Results

Th tests below provide a way to evaluate the data querying and visualization process of the system for the following functional requirements:

- FR-5
- FR-6
- FR-7
- FR-8
- FR-9

1. FR-ST2

   Control: Manual

   Initial State: The system is not running any jobs, and the user interface is cleared.

   Input: A selection of different combinations of parameters and datasets to be queried on

   Output: A human-readable and customizable visualization of correct results corresponding to the selected paramters from the input

   Test Case Derivation: The expected output of the system is based on the query parameters selected. A user expects the data analysis to match with what they asked for, and the user is allowed to customize the visuallized data.

   How test will be performed: The database will be queried using multiple combinations of parameters, and the results will be compared against the expected output. The outputted visualization will then be tested for customizability.

4.1.3 Data analysis

The tests below provide a way to evaluate the data analysis in the application for the following functional requirements:

- FR-10
- FR-11

1. FR-ST3

   Control: Manual

   Initial State: The application's cleared user interface which has not yet been used to query data, with no graph showing yet.

   Input: A combination of parameters to query on for a selected dataset

   Output: A small written human-readable paragraph explaining the input data.

   Test Case Derivation: To be able to understand the returned data in ways other than through a graph, a written response gives the user a variety of choices.

How test will be performed: The website interface will allow the user to pick a written analysis response. The application will look for patterns and trends in the data and will output the findings.

4.1.4 Data Hygiene

The tests below provide a way to evaluate how the application maintains the data hygiene of the datasets related to the following functional requirements:

- FR-12
- FR-13

1. FR-ST4

   Control: Manual

   Initial State: The application's cleared user interface which has not yet been used to query data, with no graph showing yet.

   Input: Dataset to be stored, in .CSV format

   Output: A log file documenting errors found in the input data and/or removals of missing data.

   Test Case Derivation: This is to ensure the efficency of the querying and ensuring the database is only as big as it needs to be. This will also ensure that errors are dealt with by the application and are recorded to document any inconsistancies to increase traceability.

   How test will be performed: After loading in a CSV file with some error in the data. A log file will be generated documenting the error after the application attempts to fix it.

4.1.5 User Access

This tests below provide a way to evaluate how the application allows for user login related to the following functional requirements:

- FR-14

1. FR-ST5

   Control: Manual

Initial State: User interface shows a login page, with no login credentials currently used

Input: Sample user credentials

Output: The page redirects to the page designated after login

Test Case Derivation: The system must be able to authenticate users properly, and when authenticated, they should be given access to the application

How test will be performed: Sample credentials with different combinations of characters will be used to log in to ensure the fields handle credentials correctly.

4.1.6 Data Export

The tests below provide a way to evaluate the export of query reports after a session for the following functional requirement:

- FR-15

1. FR-ST6 Control: Manual

   Initial State: User interface after multiple usages of data queries

   Input: User clicking the button for saving or downloading

   Output: Query report will be downloaded to the user's device

   Test Case Derivation: The user needs to be able to get a system generated report of the queries from their session, and be able to save or download that report as needed.

   How Test Will Be Performed: After making multiple queries on the data, the save/download button will be pressed to test functionality.

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 Testing2

...

4.3 Traceability Between Test Cases and Requirements

Req. ID	System Test ID
FR-1 FR-2 FR-3 FR-4	FR-SLN1
FR-5 FR-6 FR-7 FR-8 FR-9	FR-SLN2
FR-10 FR-11	FR-SL3
FR-12 FR-13	FR-SL4
FR-14	FR-SL5
FR-15	FR-SL6

[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.

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?