System Verification and Validation Plan for Software Engineering

Team #1, Sanskrit Ciphers
Omar El Aref
Dylan Garner
Muhammad Umar Khan
Aswin Kuganesan
Yousef Shahin

October 27, 2025

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 (**?**) 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]

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

[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]

3.3 Design Verification

[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

[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

[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

[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

4.1 Tests for Functional Requirements

This section details the functional tests derived from the system's high-priority functional requirements defined in sections S.2 and S.4 of the Requirements Standard Plan. Each test validates that the platform performs its intended scholarly and technical functions.

4.1.1 Area of Testing 1 — User Authentication and Access

This set of tests verifies that the authentication system satisfies requirements S.4.1 from the SRS, ensuring secure login, user access control, and proper

error handling. These tests confirm compliance with input validation constraints defined in the data constraints table of the SRS.

1. **Test ID:** test-fr-auth-1
   Control: Automatic
   Initial State: System running; no user session active.
   Input: Valid username and password.
   Expected Output: User successfully authenticated and directed to dashboard.
   Test Case Derivation: Confirms secure credential verification using backend authentication service.
   How test will be performed: Automated UI test verifying successful login and redirection.
   Requirement addressed: The system shall authenticate users through a secure login interface and only allow authorized access.

2. **Test ID:** test-fr-auth-2
   Control: Automatic
   Input: Invalid username and/or password.
   Expected Output: Access denied; error message displayed "Invalid username or password."
   How test will be performed: Automated UI test submitting incorrect credentials.
   Requirement addressed: Invalid credentials must be rejected.

4.1.2 Area of Testing 2 — Fragment Upload and Orientation Normalization

These tests ensure that the system fulfills requirements S.4.2 of the SRS, covering input handling for image uploads, file format constraints, and automated orientation correction. Reference is made to SRS Table 3 (Data Constraints) for permissible image formats and size limits.

1. **Test ID:** test-fr-upload-1
   Control: Automatic
   Initial State: Authenticated session.
   Input: Valid image file (less than 50 MB).

Expected Output: File successfully uploaded and preprocessing begins.
Test Case Derivation: Confirms compliance with supported file formats and upload limits.
How test will be performed: Automated API and UI tests with sample image files.
Requirement addressed: The system shall allow users to upload images (JPG, PNG, TIFF) for processing.

2. **Test ID:** test-fr-upload-2
   Control: Automatic
   Initial State: Uploaded image with random rotation.
   Input: Rotated manuscript image.
   Expected Output: Processed image aligned horizontally; confidence score ≥ 0.9.
   How test will be performed: Compare processed image orientation using OpenCV-based angle detection.
   Requirement addressed: The system shall automatically regularize image orientation within $\pm 5^{\circ}$.

4.1.3 Area of Testing 3 — Fragment Matching and Reconstruction

This section addresses requirements S.4.3 of the SRS, ensuring that the system can identify and reconstruct fragment matches based on edge and content similarity. The tests validate AI model performance, ranking algorithms, and interactive assembly functionality.

1. **Test ID:** test-fr-fragment-matching-1
   Control: Automatic
   Initial State: Database populated with preprocessed fragments.
   Input: Selected fragment ID.
   Expected Output: Ranked list of at least three probable matches with similarity scores.
   Test Case Derivation: Validates integration between AI/ML cluster and backend orchestration service.
   How test will be performed: Automated backend tests with known

fragment pairs; check if top match > 0.8 accuracy.
Requirement addressed: The system shall generate ranked match suggestions for fragments based on edge, damage, and content similarity.

2. **Test ID:** test-fr-fragment-matching-2
Control: Manual
Initial State: Logged-in user with fragments loaded.
Input: Drag-and-drop fragments onto canvas.
Expected Output: Fragments move and snap to nearby edges; arrangement saved to session.
How test will be performed: Manual verification during demonstration; supplemented with UI automation for drag-drop response $<$ 200 ms.
Requirement addressed: The system shall allow scholars to visually arrange fragments in an interactive workspace.

4.1.4 Area of Testing 4 — Script Identification and Transcription Assistance

The following tests verify that the system meets requirements S.4.4 and S.4.5 from the SRS, confirming accurate script classification and OCR-based transcription performance. These functions form the core of the text analysis pipeline and are critical for scholarly accuracy.

1. **Test ID:** test-fr-script-identification-1
Control: Automatic
Initial State: Uploaded fragment with visible text.
Input: Image containing text of known script.
Expected Output: Script classification with $\geq 70\%$ confidence.
Test Case Derivation: Validates accuracy of handwriting classification model.
How test will be performed: Model tested on labeled dataset of fragments; results compared to ground truth.
Requirement addressed: The system shall identify the paleographic script type (e.g., Gupta, Tibetan, Chinese).

2. **Test ID:** test-fr-script-identification-2
   Control: Automatic
   Input: Fragment image with legible text.
   Expected Output: Generated text file with $\geq 70\%$ character accuracy.
   How test will be performed: Compare OCR output to manually transcribed ground truth using Levenshtein distance metric.
   Requirement addressed: The system shall perform OCR transcription tuned for Sanskrit manuscripts.

4.1.5 Area of Testing 5 — Database Search and Exploration

These tests correspond to requirements S.4.6 of the SRS, which specify functionality for database search, filtering, and result export. The tests confirm that the query system performs efficiently and exports accurate data.

1. **Test ID:** test-fr-database-search-1
   Control: Automatic
   Initial State: Database with metadata indexed.
   Input: Query "Gupta script, > 5 lines."
   Expected Output: Filtered result set within 3 s response time.
   Test Case Derivation: Validates search performance and accuracy against known dataset.
   How test will be performed: API testing using Postman and performance profiling with JMeter.
   Requirement addressed: The system shall enable multi-criteria search and filtering (by script type, line count, fragment size).

2. **Test ID:** test-fr-database-search-2
   Control: Manual
   Initial State: Search results displayed.
   Input: Click "Export".
   Expected Output: Downloaded CSV file containing fragment metadata.
   How test will be performed: Manual verification of downloaded file structure.
   Requirement addressed: The system shall support exporting query

results to CSV or PDF.

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

Requirement ID	**Description**	**Test ID(s)**	**Verification Method**
S.4.1	Secure login and authentication	test-fr-auth-1, test-fr-auth-2	Automated UI & API testing
S.4.2	Fragment upload and orientation normalization	test-fr-upload-1, test-fr-upload-2	Automated pipeline testing
S.4.3	Fragment matching and reconstruction	test-fr-fragment-matching-1, test-fr-fragment-matching-2	Automated backend tests + manual UI
S.4.4	Script identification	test-fr-script-identification-1	Model accuracy testing
S.4.5	OCR transcription assistance	test-fr-script-identification-2	Automated comparison to ground truth
S.4.6	Database search and exploration	test-fr-database-search-1, test-fr-database-search-2	API performance testing + manual export check

Table 1: Traceability Between Test Cases and Requirements

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?