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 (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 presents the overall strategy for Verification and Validation (V&V) of the project. It outlines how we will organize people and activities, how key artifacts will be reviewed and tested, and how results will inform subsequent work. The goal is to provide a clear, practical roadmap for building confidence in the system from requirements through delivery.

3.1 Verification and Validation Team

This subsection defines the project roles responsible for planning, executing, and reviewing V&V activities. Each role has clear primary responsibilities and an assigned independent Reviewer to ensure separation of duties and objective appraisal. Roles may rotate by phase (requirements, design, implementation) to match expertise and workload, but accountability remains with the named owner for the current milestone. The table below summarizes the assignments.

Table 1: V&V Roles and Responsibilities

Role	Primary Responsibilities	Person	Reviewer
V&V Lead	Manages the development and execution of the V&V plan, oversees test protocols, ensures compliance, and leads root cause analysis for issues found during testing	Dylan Garner	Aswin Kuganesan
V&V Manager	Defines the overall V&V strategy, coordinates activities across the project, manages schedules and resources, and forecasts workloads	Yousef Shahin	Umar Khan
Test Architect	Builds Req↔Test traceability; drafts system and NFR tests	Omar El-Aref	Yousef Shahin
V&V Engineer	Develops and executes test plans and procedures, designs test cases, documents results, and troubleshoots issues.	Umar Khan	Dylan Garner
Quality Assurance	Often works closely with the V&V team to ensure the product meets quality standards throughout the development lifecycle	Aswin Kuganesan	Omar El-Aref
Supervisor	External reviewer; approves SRS baselining and risk mitigations	Dr. Shayne	

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]

This subsection defines how we will verify that the SRS is correct, complete, consistent, measurable, and feasible. We use three complementary approaches: guided reviews by TAs/peers, a supervisor-facing survey and interview that validate fundamentals in non-technical terms (translated into technical requirements), and continuous conformance checks during design/implementation against SRS use cases and fit criteria. Each approach includes instruments,

process, and success metrics.

TA/Peer Guided Review (Structured Feedback)

Instrument. A checklist-driven review by peers and TAs to ensure we are staying in line with the fundemntals of our requirements. Issues are filed in GitHub as feedback from TAs and peers as well as feedback on Avenue which is more detailed from the TA

Process.

1. After handing in the SRS document our peers go through the document and highlight areas to improve on.
2. Once our peers have given us feedback we are to address the feedback
3. The TA will also go through the SRS document and provide more in depth feedback to help with the document overall
4. Once we get the feedback for the TA we create an Issue in GitHub and address the feedback given to us

Success metrics.

- **Measurability coverage:** $\geq 100\%$ of feedback has been addressed
- **Ambiguity count:** ≤ 2 open ambiguity items after rework.
- **Defect turn-around time (TAT):** All feedback has been addressed within 72 hours of being given the feedback (with a grace period of 48 hours for the entire team)
- **Link/xref errors:** 0 build warnings for missing references.

Supervisor Validation via Survey plus Interview (Non-technical → Technical)

Instrument. A 10-question survey focused on *fundamental product expectations*, followed by a 30 min interview. Example survey items:

1. "The SRS describes how fragments will be *prepared* (orientation/cleaning) before analysis." (1–5)

2. "The SRS explains how *candidate matches* will be presented for scholarly judgment." (1–5)

3. "The SRS addresses *trust signals* (e.g., confidence scores, overlays) for suggested matches." (1–5)

4. "Privacy and access for *unpublished images* are clearly described." (1–5)

5. Open: "What essential research task is missing or under-specified?"

Process.

1. Send SRS and survey 48 hours in advance; collect responses.

2. Hold interview; capture clarifications and priorities (top 3 must-haves).

3. *Translation step:* Requirements Analyst converts each non-technical concern into one or more technical requirements (G/S with Fit Criteria), files issues, updates traceability.

Success metrics.

- **Coverage score:** average score ≥ 4.0 on survey.
- **Gap closure:** 100% of supervisor "missing/under-specified" items mapped to new/updated requirements within 1 week.
- **Traceability updates:** new items appear in Req→Test matrix with an assigned test type and owner.

C. Ongoing Conformance During Build (Design/Code → SRS)

Instrument. A checklist that is shared with the team and has each module mapped to a requirement.
Process.

1. For each S.4 scenario in the SRS, create at least one system test; link the implementing issue/PR.

2. On each PR, the author selects related requirement from the SRS document.

3. Monthly conformance audit: Each month the team will highlight what percentage of the requirements is done or being completed.

4. The team will also hihglight each month if they encountered a requirement that the SRS missed and make sure it is added within 48 hours of finding the missing requirement.

5. If code conflicts with SRS, open change request: either update SRS (with rationale) or refactor code.

Success metrics.

- **Traceability coverage:** $\geq$ 95% of modules or requirments can be linked to code module.
- **Use-case alignment:** 100% of G.5 use cases have a corresponding S.4 detailed scenario and at least one test case.

3.3 Design Verification

[Plans for design verification —SS]
[The review will include reviews by your classmates —SS]
[Create a checklists? —SS]

This subsection defines how we will verify that the *design* (architecture, detailed design, and model design) is **sound, implementable, and consistent** with the SRS. We use three complementary techniques with objective success criteria.

DV-1: UML Design $\leftrightarrow$ Code Modules

Approach. Maintain a living UML class diagram for core modules (front-end, backend services, AI/ML components). Verify code conformance on each milestone via static review (e.g., presence of classes/interfaces, key method signatures, dependency directions).

- **Procedure:**

    1. Identify *must-exist* types and interfaces in the UML for the current milestone.

2. Map each UML element to its code module (file/path, namespace, class/interface name).
3. Keep a checklist of the modules that have been mapped and the ones that still need too get created.
4. As the design starts getting implemented we can start adjusting the UML diagram as needed.

- **Success metrics:**
  - *Coverage:* $\geq 95\%$ of *must-exist* UML elements mapped to code modules.
  - *Directional integrity:* 0 violations of designated dependency rules.

DV-2: Structured Walkthroughs (Code/Design)

We will use lightweight, structured walkthroughs for code and design implementations

Author

- **What they do:** Presents the change (PR), states intent, scope, and known risks; answers questions.
- **How it works:** In a meeting they will share their screen and present the changes they made as well as the logic behind their changes
- **Why it rotates:** The author is simply whoever made the change for that task; authorship necessarily varies by feature or document section.

Reviewer

- **What they do:** Performs the detailed critique using a checklist (correctness, testability, traceability, compliance).
- **How it works:** Reviews asynchronously before the session; in the session, walks through findings; files issues or suggests concrete edits.
- **Why it rotates:** Review expertise should match the change (e.g., ML for matching model code, accessibility for UI text), so the most relevant teammate reviews.

Walkthrough Lead

- **What they do:** Facilitates the session, keeps time, ensures the checklist is followed, resolves scope creep, and states clear outcomes.
- **How it works:** Opens with goals and exit criteria; keeps discussion to evidence; confirms owner and due date for each action.
- **Why it rotates:** Lead alternates between V&V Lead and Test Architect depending on topic (documents vs. tests/code), ensuring domain-appropriate facilitation.

Audience / Notetaker

- **What they do:** Attends for situational awareness; the designated notetaker records decisions, action items, and deferments.
- **How it works:** Notes are captured and action items become GitHub issues (`blocker/major/minor`) with assignees and deadlines.
- **Why it rotates:** Anyone can take notes; rotating spreads context and avoids overloading a single person.

Process (applies to both code and document walkthroughs).

1. **Prep (T − 24h):** Author posts link(s) to PR , a 1–2 paragraph summary, and relevant checklists.
2. **Asynchronous review:** Reviewer annotates items and runs applicable checks.
3. **Session (30 minutes):** Lead states goals & exit criteria; Author presents; Reviewer walks through findings; decisions and actions are recorded.
4. **Exit criteria:** All *blocker/major* items assigned with due dates; PR updated or issues filed; next steps acknowledged by owners.
5. **Follow-up (72h):** Author closes items; Reviewer verifies fixes; Lead updates the review log.

Why roles change per task. Changes originate in different parts of the system (UI, backend, AI/ML). The *author* is the contributor for that change; the *reviewer* is selected for the best subject-matter fit; the *lead* alternates to match topic; the *notetaker* rotates to share context. This rotation ensures expertise, fairness in workload, and higher defect detection while remaining lightweight and feasible.

DV-3: ML Design Soundness (Regularization/Robustness)

Approach. Verify that AI/ML components are designed to generalize: regularization and robustness are evaluated with stress tests.

- **Controls implemented:** dropout (or equivalent), data augmentation, early stopping, and (where applicable) ensembles.
- **Ablation plan:** compare baseline vs. dropout, augmentation, ensemble; report mean±sd over k-fold Cross Validation.
- **Robustness:** evaluate on mildly perturbed inputs (rotation, illumination, small occlusion); report metric deltas.
- **Success metrics:**
  - Will mainly be measured by if we found a better more robust model. Regardless of the outcome it will be a success since we have tested the models ability to generalize.
- **Design outcome:** if criteria fail, document change in an ADR (e.g., add augmentation X, adjust dropout p, switch loss) and retest.

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]

The V&V plan is itself an artifact that must be verified and improved over time. We will use peer/TA reviews, a structured test-plan walkthrough & retrospective focused on functional and nonfunctional tests

PV-1: Classmate and TA Reviews

What: External review by classmates and a course TA to assess clarity, completeness, and feasibility.
How:

- Have a completed version of the V&V plan submitted.
- Class peers review the plan and give feedback in an Issue on the GitHub.
- TA reviews the plan and confirms scope, grading-aligned expectations, and missing evidence.
- Each team memeber will create an Issue on the GitHub and fix the comments they were given on their respective parts.

Success: $\geq 95\%$ of actions from peer/TA reviews closed within 7 days.

PV-2: Test-Plan Walkthrough & Retrospective (Functional & Nonfunctional)

What: A structured meeting to walk through current system tests (functional) and NFR tests (accessibility, privacy, performance, usability), emphasizing what has worked, what has not, and how to adapt. This will be quite similar to the walkthrough mentioned ion susection 3.1.
Inputs: Current test catalog, pass/fail history, known pain points, and a walkthrough checklist covering: specificity of steps/data, pass/fail criteria, traceability to SRS Req IDs, reusability, and stakeholder relevance.
How:

1. Lead presents the V&V Plan document and walks through the sections asking if the team has any doubts with the currecnt section.
2. If a team member does have doubts about a section they would then specify. (If not the Lead moves on to the next section).
3. The team comes up with a solution to address the doubt and clear up any misunderstadning.
4. Team records adjustments in an Issue on the GitHub with owner and due date.

Success:

- Executability: 100% of sampled tests can be executed by a third party without clarification after fixes.
- Specificity: Each test specifies input data, exact steps, expected outputs, and pass thresholds.
- Adaptation throughput: $\geq 95\%$ of improvement actions closed by next walkthrough.

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

[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 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?