System Verification and Validation Plan for Software Engineering

Team #10, Five of a Kind
Omar Abdelhamid
Daniel Maurer
Andrew Bovbel
Olivia Reich
Khalid Farag

October 16, 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

[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

This section outlines the tests for verifying both functional and nonfunctional requirements of the software, ensuring it meets user expectations and performs reliably. This includes tests for code quality, usability, performance, security, and traceability, covering essential aspects of the software's operation and compliance.

4.1 Tests for Functional Requirements

The subsections below outline tests corresponding to functional requirements in the SRS (Author, 2019). Each test is associated with a unique functional

area, helping to confirm that the tool meets the specified requirements. Each functional area has its own subsection for clarity.

The system tests are organized by the major components identified in the SRS: Import Manager and Visualization Manager. These components handle the core functionality of CAD file processing, voxel slicing, 3D model visualization, and user interaction. The tests ensure that all functional requirements (F211-F216 for Import Manager and F221-F227 for Visualization Manager) are properly validated through both automated and manual testing approaches.

4.1.1 Import Manager Tests

This section covers tests for the Import Manager component, which handles CAD file interpretation, voxel slicing, and model manipulation. These tests verify the functional requirements F211-F216 from the SRS, ensuring proper file handling, model processing, and user configuration capabilities. The tests validate both automated processing and user interaction scenarios.

CAD File Import and Project Management Tests

1. test-FR-IM-1 Valid CAD File Import for New Project

   Control: Automated

   Initial State: System is idle, no active project loaded

   Input: A valid CAD file (e.g., .stl, .obj, .ply format) containing a 3D mesh model

   Output: The system successfully creates a new project, imports the CAD file, and initiates the voxel slicing process. The imported model is ready for visualization.

   Test Case Derivation: Confirms that the system correctly processes valid CAD files and creates new projects as specified in F211. This validates the core functionality of starting new projects with imported CAD files.

   How test will be performed: Provide a valid CAD file through the import interface and verify that a new project is created, the file is processed without errors, and the model data is available for subsequent operations.

2. test-FR-IM-2 Past Project File Import

   Control: Automated

   Initial State: System is idle, no active project loaded

   Input: A previously saved project file containing magnetization and material properties

   Output: The system successfully loads the project with all magnetization and material properties preserved, and the model is ready for further editing.

   Test Case Derivation: Ensures that past projects can be reopened with complete data integrity, satisfying F212. This prevents users from losing work and enables iterative design processes.

   How test will be performed: Load a previously saved project file and verify that all voxel properties, magnetization vectors, and material assignments are correctly restored.

3. test-FR-IM-3 Invalid File Format Handling

   Control: Automated

   Initial State: System is idle, no active project loaded

   Input: A file with unsupported format (e.g., .txt, .jpg, .pdf)

   Output: The system rejects the file and displays an appropriate error message indicating unsupported file format, without creating a project or crashing.

   Test Case Derivation: Validates robust error handling for invalid inputs, ensuring system stability when users attempt to import incompatible files.

   How test will be performed: Attempt to import various unsupported file formats and verify that the system gracefully handles the error without system failure.

Model Configuration and Slicing Tests

1. test-FR-IM-4 Voxel Size Configuration

   Control: Manual

   Initial State: A valid CAD model has been imported and is ready for slicing

   Input: User specifies custom voxel dimensions (e.g., 0.1mm x 0.1mm x 0.1mm)

   Output: The system applies the specified voxel dimensions and successfully slices the model into voxels of the requested size.

   Test Case Derivation: Confirms that users can configure voxel dimensions as required by F213, enabling customization of model resolution for different printing requirements.

   How test will be performed: Import a CAD model, configure custom voxel dimensions through the user interface, and verify that the resulting voxel grid matches the specified dimensions.

2. test-FR-IM-5 Model Scaling Functionality

   Control: Manual

   Initial State: A valid CAD model has been imported

   Input: User modifies model dimensions (e.g., scale to 50% of original size)

   Output: The model is successfully scaled to the specified dimensions while maintaining geometric integrity and proportions.

   Test Case Derivation: Validates F214 by ensuring users can modify model dimensions before slicing, enabling proper scaling for different printing requirements.

   How test will be performed: Import a CAD model, use the scaling interface to modify dimensions, and verify that the scaled model maintains proper proportions and geometric accuracy.

3. test-FR-IM-6 Partial Voxel Resolution

   Control: Automated

Initial State: A CAD model with complex geometry has been imported

Input: A model containing surfaces that intersect voxel boundaries (creating partial voxels)

Output: The system correctly resolves partial voxels using appropriate algorithms (e.g., majority rule, surface area weighting) and preserves model integrity and accuracy.

Test Case Derivation: Ensures F215 is satisfied by validating that partial voxels are handled correctly, maintaining model accuracy during the slicing process.

How test will be performed: Import a CAD model with complex geometry that creates partial voxels, and verify that the resolution algorithm produces accurate results that preserve the original model's shape and volume.

4. test-FR-IM-7 Model Division for Display

   Control: Automated

   Initial State: A large CAD model has been imported and sliced into voxels

   Input: A model containing more than MAX_DISPLAY voxels

   Output: The system automatically partitions the model into manageable display sections that do not exceed the MAX_DISPLAY threshold, enabling smooth visualization.

   Test Case Derivation: Confirms F216 by ensuring large models are properly divided into manageable sections for optimal display performance and user interaction.

   How test will be performed: Import a large CAD model that exceeds MAX_DISPLAY voxels, and verify that the system creates appropriate partitions that maintain model coherence while staying within display limits.

4.1.2 Visualization Manager Tests

This section covers tests for the Visualization Manager component, which handles 3D model rendering, user interaction, and visual feedback. These

tests verify the functional requirements F221-F227 from the SRS, ensuring proper 3D visualization, navigation, and user interface functionality. The tests validate both rendering capabilities and interactive features.

3D Model Rendering and Display Tests

1. test-FR-VM-1 3D Model Rendition with Clear Voxel Visualization

   Control: Manual

   Initial State: A CAD model has been imported and sliced into voxels by the Import Manager

   Input: Sliced voxel data from Import Manager

   Output: The system successfully renders a 3D model with clear visualization of individual voxels, maintaining geometric accuracy and providing distinct voxel boundaries.

   Test Case Derivation: Confirms F221 by ensuring the Visualization Manager can recreate 3D models with clear voxel visualization, which is essential for property assignment and user understanding of the model structure.

   How test will be performed: Import and slice a CAD model, then verify that the 3D rendition clearly displays individual voxels with proper boundaries and geometric accuracy.

2. test-FR-VM-2 Partition Navigation Functionality

   Control: Manual

   Initial State: A large model has been divided into multiple display partitions

   Input: User navigation commands to switch between model partitions

   Output: The system provides smooth navigation between partitions, allowing users to access all model sections without performance degradation.

   Test Case Derivation: Validates F222 by ensuring users can navigate across all model partitions, supporting the Import Manager's model division functionality.

How test will be performed: Load a large model with multiple partitions and verify that navigation controls allow seamless movement between all sections of the model.

3. test-FR-VM-3 Multi-Perspective 3D Model Interaction

   Control: Manual

   Initial State: A 3D model is rendered and displayed

   Input: User interaction commands (rotate, zoom, pan) to view the model from different perspectives

   Output: The system provides intuitive interface controls that allow seamless navigation across multiple perspectives of the 3D model with responsive interaction.

   Test Case Derivation: Confirms F223 by ensuring users can interact with the 3D model from all perspectives, providing comprehensive visualization capabilities.

   How test will be performed: Use the 3D interaction controls to rotate, zoom, and pan the model, verifying that all perspectives are accessible and the interface remains responsive and intuitive.

Layer Focus and Voxel Selection Tests

1. test-FR-VM-4 Layer Isolation Functionality

   Control: Manual

   Initial State: A 3D model with multiple layers is displayed

   Input: User selects a specific layer to focus on

   Output: The system isolates the selected layer, rendering all other voxels irrelevant or transparent, facilitating property assignment on the focused layer.

   Test Case Derivation: Validates F224 by ensuring users can isolate specific layers for focused interaction, which is essential for efficient property assignment.

How test will be performed: Select different layers of a multi-layer model and verify that only the selected layer remains visible and interactive while other layers are appropriately dimmed or hidden.

2. test-FR-VM-5 Voxel Selection Highlighting

   Control: Manual

   Initial State: A specific layer is in focus and displayed

   Input: User selects individual voxels or groups of voxels within the focused layer

   Output: The system provides clear visual feedback showing which voxels are currently selected, using distinct highlighting or color changes.

   Test Case Derivation: Confirms F225 by ensuring users receive clear visual feedback about their current selections, improving usability and reducing errors.

   How test will be performed: Select various voxels within a focused layer and verify that the selection is clearly highlighted with distinct visual indicators.

Material and Magnetization Tracking Tests

1. test-FR-VM-6 Material Assignment Visual Tracking

   Control: Manual

   Initial State: A model with unassigned voxels is displayed

   Input: User assigns material IDs to various voxels

   Output: The system updates voxel colors to indicate material assignment completeness, providing clear visual feedback about assignment status.

   Test Case Derivation: Validates F226 by ensuring users can easily track which voxels have been assigned materials through visual color changes.

   How test will be performed: Assign materials to various voxels and verify that the color changes provide clear indication of assignment status and remaining work.

2. test-FR-VM-7 Magnetization Assignment Visual Tracking

   Control: Manual

   Initial State: A model with some magnetized and some unmagnetized voxels

   Input: User toggles the magnetization visualization display

   Output: The system provides the option to toggle color display of all magnetized voxels, allowing users to easily identify magnetization status.

   Test Case Derivation: Confirms F227 by ensuring users can track magnetization assignments through visual toggles, improving workflow efficiency.

   How test will be performed: Toggle the magnetization visualization and verify that magnetized voxels are clearly distinguished from unmagnetized ones through appropriate color coding.

4.1.3 Integration Tests Between Import and Visualization Managers

This section covers tests that validate the interaction and data flow between the Import Manager and Visualization Manager components. These tests ensure that the two managers work together seamlessly to provide a complete user experience from CAD file import to 3D visualization.

Data Flow and Communication Tests

1. test-FR-INT-1 Import to Visualization Data Transfer

   Control: Automated

   Initial State: System is idle, no active project

   Input: A valid CAD file is imported and processed by the Import Manager

   Output: The Visualization Manager receives complete voxel data and successfully renders the 3D model without data loss or corruption.

Test Case Derivation: Ensures proper data flow between managers, validating that voxel slicing results are correctly transmitted to the visualization system.

How test will be performed: Import a CAD file and verify that the resulting 3D visualization accurately represents the sliced voxel data from the Import Manager.

2. test-FR-INT-2 Model Division Integration

   Control: Automated

   Initial State: A large CAD model has been imported

   Input: Import Manager divides the model into display partitions

   Output: Visualization Manager correctly displays and allows navigation between all partitions created by the Import Manager.

   Test Case Derivation: Validates that model division from Import Manager is properly handled by Visualization Manager, ensuring seamless user experience with large models.

   How test will be performed: Import a large model that requires division and verify that all partitions are accessible through the visualization interface.

3. test-FR-INT-3 Configuration Parameter Synchronization

   Control: Manual

   Initial State: A model is ready for slicing

   Input: User configures voxel size and model scaling parameters

   Output: Both Import Manager (for slicing) and Visualization Manager (for display) correctly apply the configuration parameters and maintain consistency.

   Test Case Derivation: Ensures that user configuration changes are properly synchronized between managers, maintaining data consistency throughout the system.

   How test will be performed: Modify configuration parameters and verify that both slicing results and visualization accurately reflect the changes.

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?