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 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

The design verification will primarily focus on the system framework and integrated architecture that details the system implementation. Verification will take place through structured classmate and supervisor reviews. Reviews conducted by classmates will be designed to emphasize the interface design that permits usage of external libraries, internal refactoring and the primary database structure that encapsulates all voxel metadata. Feedback will be created into GitHub issues and discussed amongst the group in order to determine the best course of action to effectively integrate the feedback in subsequent development cycles.

Reviews conducted with our supervisor, Dr. Onaizah, will primarily verify tolerance parameters within the output framework during integration with specialized 3D printing software. Diagrams may be developed to provide visual aids to help facilitate the technical conversations and enhance understanding. These diagrams will default to standard UML diagrams, unless a more simplified diagram is required due to disparity in technical proficiency.

A checklist evaluation has been provided below for reference:

Table 1: Design Verification Checklist

Focus Area	**Verification Tasks**
Core Architecture	• Defined system modularity. • Compliance with standard coding conventions and language-specific best practices. • Software architecture patterns verified.
IDE Integration	• External library integration for voxel slicing validated. • UI/UX patterns verified.
Database Integrity	• Database structure confirmed. • Voxel data management operations verified for accuracy.
External Data Integrity	• Structure of data files confirmed.

3.4 Verification and Validation Plan Verification

The verification validation testing plan will also incorporate peer reviews and certain testing methodologies. It will be evaluated on the following 4 key metric categories: coverage, test effectiveness, testing methodology, and traceability. Peer reviews will be centered around structured checklists that guide meaningful assessment on the scope and completeness of the verification validation plan. Completeness within the scope of the plan should effectively address system scalability in regard to the software's ability to manage growth in project size, complexity, and anticipated functional demands. There will be emphasis placed on ensuring effective coverage of user interactions with 3D renditions and all voxel data management operations. Feedback will be created into GitHub issues and discussed amongst the group in order to determine the best course of action to effectively integrate the feedback in subsequent development cycles.

Feedback will be integrated with an interactive development approach. Established functionality within the system will not be considered complete until complete testing in accordance with the verification validation plan has been completed. During weekly group meetings, updates related to a specific feature will require a corresponding update regarding related test cases that will be established. Tests must also be documented as GitHub issues. Upon closing a test case, the results must be documented within the issue. Results will be evaluated against a pre-defined baseline success metrics.

Within test validation, mutation testing will be integrated to assess the effectiveness of the system. This will include the assertion of defective input files, faulty voxel metadata updates, and unsupported display commands. Our goal with mutation testing is to generate quantifiable observations regarding our system's capabilities of detecting undesirable anomalies.

A checklist evaluation has been provided below for reference:

Table 2: V&V Plan Verification Checklist

Criteria	Verification Tasks
Coverage	• Quantitative evaluation metrics defined. • Established configuration of testing domains. • Established coverage cases. • Quantitative coverage summaries.
Testing Methodology	• Established baseline success metrics. • Testing classification completed.
Traceability	• Test tracking integrated within GitHub issues. • Formalized feedback process.

3.5 Implementation Verification

Implementation verification will ensure that all segments of code strictly adhere to the requirements and constraints outlined in the SRS document as well as ensure adherence to the mandatory programming language's conventions. Unit tests will be used in order to validate the core functionalities with the system. Pytest will likely be the primary testing framework for unit tests relating to the system components' development in Python. In a similar manner, Jest will be used to facilitate unit testing with the UI/UX of our system. Test cases will primarily focus on testing the functionality related to voxel modification requirements identified in F1-F4.

Static analysis will be conducted to ensure accordance with PEP 8, which is the official style guide for writing Python code. There will also be code inspections to verify secure and consistent file handling across all interconnected systems prior to running the code. This is in accordance with SCR1, SCR4, and SCR5.

To ensure high implementation quality throughout all development cycles, code reviews will be on a minimum weekly basis (granted the code was modified throughout that period). These code reviews will incorporate the following:

- effective usage of external Python libraries in accordance with their usage guidelines
- optimized voxel modification processes (F231, F233, F235, F239)
- precise 3D rendition (F221, F222)
- verification of future scalability (NF211)

Testing will also be done to evaluate the performance of implementation. In accordance with thresholds as defined in associated requirements, it will be evaluated as follows:

- data processing across file input and output workflows (NF212, NF242)
- optimized interface responsiveness (F223)
- large database interaction (NF232, SCR3)

Table 3: Implementation Verification Checklist

Category	**Verification Tasks**
Functionality	• Defined test case sets for component requirements (SRS, S.2). • Visual rendering validation. • Voxel modification validation.
Quality	• Structured code walkthrough completed. • Static analysis in accordance with the development environment. • Weekly code reviews conducted.
Performance	• File handling benchmarks. • User interaction processing time benchmarks. • Database interaction benchmarks.
Traceability	• Test tracking integrated within GitHub issues.

3.6 Automated Testing and Verification Tools

Unit Testing Framework (backend): Since the backend of our system will be coded in Python, the primary testing framework used will be PyTest. This framework was chosen due to its flexible testing framework that can be scaled to support complex function testing.

Code coverage tools in PyTest: PyTest offers a variety of tools, including the usage of the coverage.py library (pytest-cov), which allows extensive code coverage. It is able to monitor the program during execution and report what parts of the code have been executed. The plugin pytest-mock also provides a convenient way to mock objects and functions, including those in external libraries. This will be useful for testing an external voxel slicing library.

Unit Testing Framework (frontend): Currently, the primary testing framework that will be used for the front end is Jest, as we will be using the JavaScript library, Three.js, to generate the 3D image.

Code coverage tools in Jest: Jest offers several specialized libraries and techniques that can create image analysis tools for 3D image data evaluation. This includes tools such as snapshot test (jest-image-snapshot), which allows visual comparison. It also offers a mocking tool, which allows testing of specific property image components.

Ideally, the goal is to achieve 50% coverage across both the frontend and backend of the codebases by the end of Rev 0. It is expected the PoC demo product will have significantly reduced coverage. However, coverage will be integrated through an interactive approach. Weekly reports will be generated from all coverage tools to track testing through the project. This will allow us to gain insight into overall test progress, understand general trends within code coverage, and provide clarity on what areas still require additional validation.

3.7 Software Validation

User Test Group
To validate the usability of the system, a specialized test group will be selected based on the current typical users of the 3D printer. Testing will take the form of exploratory and acceptance testing. Users will be provided a list of tasks that they will attempt to navigate through with no prior instruction. The goal is to assess the system on the following aspects: usability, accessibility, discoverability, natural flow of interaction and system behaviour.

The list of tasks will closely relate to essential requirements. The list of tasks will depend on the current status of the project. However, tasks may include:

- create a new project with a given CAS file (F221)
- generate a 3D image with a given voxel size (F213)
- access the bottom face of the object (F223)
- given desired voxel properties of 3 layers, replicate structure on software (F224 - F227, F231, F2313)
  - all 3 layers will have the same voxel properties to see if the user experiments with property replication (F235)
- erase all properties from the third layer
- assign properties to the remaining voxels and export the project

Rev 0
This will include a meeting with Dr. Onaizah to validate the following:

- implementation of SRS requirements
- desired modifications to requirements or changes in user needs
- how the system behaviour compares to specifications

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]

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?