# 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 |
|---|---|---|
| October 27, 2025 | 1.0 | Initial draft by All |

# Contents

# List of Tables

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

# 1 General Information

## 1.1 Summary

[Say what software is being tested. Give its name and a brief overview of its general functions. —SS]

## 1.2 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]

## 1.3 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]

### 1.4 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]

## 2 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

### 2.1 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]

### 2.2 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]

## 2.3 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.<br>• Compliance with standard coding conventions and language-specific best practices.<br>• Software architecture patterns verified. |
| IDE Integration | • External library integration for voxel slicing validated.<br>• UI/UX patterns verified. |
| Database Integrity | • Database structure confirmed.<br>• Voxel data management operations verified for accuracy. |
| External Data Integrity | • Structure of data files confirmed. |

## 2.4 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.<br>• Established configuration of testing domains.<br>• Established coverage cases.<br>• Quantitative coverage summaries. |
| Testing Methodology | • Established baseline success metrics.<br>• Testing classification completed. |
| Traceability | • Test tracking integrated within GitHub issues.<br>• Formalized feedback process. |

## 2.5 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).<br>• Visual rendering validation.<br>• Voxel modification validation. |
| Quality | • Structured code walkthrough completed.<br>• Static analysis in accordance with the development environment.<br>• Weekly code reviews conducted. |
| Performance | • File handling benchmarks.<br>• User interaction processing time benchmarks.<br>• Database interaction benchmarks. |
| Traceability | • Test tracking integrated within GitHub issues. |

## 2.6 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.

## 2.7 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

# 3 System Tests

This section outlines the tests for verifying both functional and nonfunctional requirements of the system. It helps ensure that the system meets stakeholders' expectations. This section includes tests covering the essential aspects of the systems requirements.

## 3.1 Tests for Functional Requirements

The subsections below outline tests corresponding to functional requirements in the SRS. Each test is associated with a unique functional area, helping to confirm that the system meets the specified requirements.

### 3.1.1 Import Manager Tests

**CAD File Import and Project Management Tests**

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

   Type: Functional, Dynamic, Manual

   Covers: F211

   Initial State: No active project is loaded in the system, and the system is ready to create a new project.

   Input: A CAD file (STL file) is provided to the system through the import interface by the user.

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

   Test Case Derivation: The test case is derived from the functional requirement F211, which states that the Import Manager should allow a user to start a new project with the option to import an initial CAD file from their personal device that will be sliced.

   How test will be performed: Create a new project on the system and import a CAD file into the new project to verify that the system will correctly process the file and create a new project. The system should be able to handle the CAD file and initiate the voxel slicing process. This will be tested manually by the developer of the system.

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

   Type: Functional, Dynamic, Manual

   Covers: F212

   Initial State: No active project is loaded in the system, and the system is ready to open a past project.

   Input: A previously saved project file containing magnetization and material properties is provided to the system through the import interface by the user.

Output: The system loads the project containing all of the magnetization and material properties preserved. The model is ready for further editing and visualization.

Test Case Derivation: The test case is derived from the functional requirement F212, which states that the Import Manager should allow a user to import a past project file in order to reopen a project with all magnetization and material properties preserved.

How test will be performed: Open a previously saved project containing magnetization and material properties on the system. Verify that the system correctly loads the project and all of the magnetization and material properties are preserved. The model should be ready for further editing and visualization. This will be tested manually by the developer of the system.

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

Type: Functional, Dynamic, Manual

Covers: F211

Initial State: No active project is loaded in the system, and the system is ready to import a file.

Input: A file with unsupported format (e.g. .txt, .jpg, .pdf) is provided to the system through the import interface by the user.

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

Test Case Derivation: The test case is derived from the functional requirement F211, which states that the Import Manager should allow a user to start a new project with the option to import an initial CAD file from their personal device that will be sliced.

How test will be performed: Try to import a file with an unsupported format (e.g. .txt, .jpg, .pdf) into the system. Verify that the system correctly rejects the file and displays an appropriate error message indicating the file format is unsupported. This will be tested manually by the developer of the system.

**Model Configuration and Slicing Tests**

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

   Type: Functional, Dynamic, Automated

   Covers: F213

   Initial State: A valid CAD model has been imported and the system is ready to configure the voxel dimensions.

   Input: User specifies custom voxel dimensions (e.g. 0.1mm x 0.1mm x 0.1mm) by entering the dimensions into the system through the import interface by the user.

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

   Test Case Derivation: The test case is derived from the functional requirement F213, which states that the Import Manager should allow a user to configure the voxel dimensions into which the model will be sliced.

   How test will be performed: Configure the voxel dimensions for the imported model by entering the dimensions into the system through the import interface by the user. Verify that the system correctly applies the specified voxel dimensions and successfully slices the model into voxels of the requested dimensions. This will be tested automatically by the developer of the system by running a script that configures the voxel dimensions and verifies that the system correctly slices the model into voxels of the requested dimensions.

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

   Type: Functional, Dynamic, Manual

   Covers: F214

   Initial State: A valid CAD model has been imported and the system is ready to scale the model.

   Input: User modifies the model dimensions to the desired printing size by entering the new dimensions into the system through the import interface by the user.

Output: The model is scaled to the desired printing size, maintaining its geometric integrity and proportions.

Test Case Derivation: The test case is derived from the functional requirement F214, which states that the Import Manager should allow a user to scale the model to the desired printing size before being sliced into voxels.

How test will be performed: Modify the model dimensions to the desired printing size by entering the new dimensions into the system through the import interface by the user. The developer will then manually inspect the rendered 3D model within the application. Verification will involve comparing the visual representation of the sliced model against the original CAD model and the expected scaling of the model to confirm that model integrity and accuracy are preserved as specified in F214

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

Type: Functional, Dynamic, Manual

Covers: F215

Initial State: A CAD model with complex geometry has been imported and the system is ready to resolve the partial voxels.

Input: A model containing surfaces that intersect voxel boundaries is imported by the user through the import interface.

Output: The system correctly resolves partial voxels and preserves model integrity and accuracy.

Test Case Derivation: The test case is derived from the functional requirement F215, which states that the Import Manager shall resolve partial voxels and interpret them in the best way that preserves the integrity and accuracy of the provided model.

How test will be performed: Import a model containing surfaces that intersect voxel boundaries. The developer will then manually inspect the rendered 3D model within the application. Verification will involve comparing the visual representation of the sliced model against the original CAD model and the expected resolution of partial voxels to

confirm that model integrity and accuracy are preserved as specified in F215

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

   Type: Functional, Dynamic, Manual

   Covers: F216

   Initial State: A large CAD model has been imported and the system is ready to divide the model into manageable display sections.

   Input: A model containing more than MAX_DISPLAY voxels is imported by the user through the import interface.

   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: The test case is derived from the functional requirement F216, which states that the Import Manager shall partition the model into user-manageable display sections that do not surpass the MAX_DISPLAY threshold.

   How test will be performed: Import a model containing more than MAX_DISPLAY voxels. The developer will then manually inspect the rendered 3D model within the application. Verification will involve comparing the visual representation of the sliced model against the original CAD model and the expected partitioning of the model to confirm that model integrity and accuracy are preserved as specified in F216

### 3.1.2 Visualization Manager Tests

**3D Model Rendering and Display Tests**

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

   Type: Functional, Dynamic, Manual

   Covers: F221

   Initial State: A CAD model has been imported and sliced into voxels by the Import Manager and the system is ready to render the model.

Input: Sliced voxel data from Import Manager is provided to the Visualization Manager.

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

Test Case Derivation: The test case is derived from the functional requirement F221, which states that the Visualization Manager shall recreate a 3D model with clear visualization of sliced voxels.

How test will be performed: Render a 3D model with clear visualization of sliced voxels by providing the Visualization Manager with sliced voxel data from the Import Manager. Verify that the system correctly renders the model and displays the individual voxels with proper boundaries and geometric accuracy. This will be tested manually by the developer of the system by providing the system with a CAD model and verifying that the system correctly renders the model and displays the individual voxels with proper boundaries and geometric accuracy.

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

   Type: Functional, Dynamic, Manual

   Covers: F222

   Initial State: A large model has been divided into multiple display partitions and the system is ready to navigate between the partitions.

   Input: User navigates between model partitions by clicking on the different display sections of the model.

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

   Test Case Derivation: The test case is derived from the functional requirement F222, which states that the Visualization Manager shall provide users with simplistic navigation across user-manageable display sections to ensure access to all model partitions.

   How test will be performed: Navigate between model partitions by clicking on the different display sections of the model. Verify that the

system correctly provides smooth navigation between partitions, allowing users to access all model sections without performance degradation. This will be tested manually by the developer of the system.

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

   Type: Functional, Dynamic, Manual

   Covers: F223

   Initial State: A 3D model is rendered and displayed and the system is ready to interact with the model from different perspectives.

   Input: User rotates and zooms the model to view it from different perspectives using intuitive interface controls.

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

   Test Case Derivation: The test case is derived from the functional requirement F223, which states that the Visualization Manager shall provide users with an intuitive interface that permits seamless navigation across multiple perspectives of the 3D model.

   How test will be performed: Rotate and zoom the model to view it from different perspectives using intuitive interface controls. Verify that the system correctly provides intuitive interface controls that allow seamless navigation across multiple perspectives of the 3D model with responsive interaction. This will be tested manually by the developer of the system.

**Layer Focus and Voxel Selection Tests**

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

   Type: Functional, Dynamic, Manual

   Covers: F224

   Initial State: A 3D model with multiple layers is displayed and the system is ready to isolate a specific layer.

Input: User selects a specific layer to focus on by clicking on the layer in the display.

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

Test Case Derivation: The test case is derived from the functional requirement F224, which states that the Visualization Manager shall allow users to isolate a specific layer to facilitate property assignments, rendering all other voxels irrelevant while that layer is in focus.

How test will be performed: Select a specific layer to focus on by clicking on the layer in the display. Verify that the system correctly isolates the selected layer and renders all other voxels irrelevant while that layer is in focus. This will be tested manually by the developer of the system.

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

   Type: Functional, Dynamic, Manual

   Covers: F225

   Initial State: A specific layer is in focus and displayed and the system is ready to highlight the selected voxels.

   Input: User selects individual voxels or groups of voxels within the focused layer by clicking on the voxels in the display.

   Output: The system correctly provides clear visual feedback showing which voxels are currently selected using distinct highlighting.

   Test Case Derivation: The test case is derived from the functional requirement F225, which states that the Visualization Manager shall provide users with visualization that showcases which voxels are currently selected within a specific layer.

   How test will be performed: Select individual voxels or groups of voxels within the focused layer by clicking on the voxels in the display. Verify that the system correctly provides clear visual feedback showing which voxels are currently selected using distinct highlighting. This will be tested manually by the developer of the system.

**Material and Magnetization Tracking Tests**

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

   Type: Functional, Dynamic, Manual

   Covers: F226

   Initial State: A model with unassigned materials is displayed and the system is ready to track the material assignment.

   Input: User assigns material IDs to various voxels by clicking on the voxels in the display and assigning the material IDs.

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

   Test Case Derivation: The test case is derived from the functional requirement F226, which states that the Visualization Manager shall integrate easy tracking of voxels that have been assigned material IDs by adjusting the colour of the voxel to indicate assignment completeness.

   How test will be performed: Assign material IDs to various voxels by clicking on the voxels in the display and assigning the material IDs. Verify that the system correctly updates the voxel colors to indicate material assignment completeness, providing clear visual feedback about assignment status. This will be tested manually by the developer of the system.

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

   Type: Functional, Dynamic, Manual

   Covers: F227

   Initial State: A model with unassigned magnetization vectors is displayed and the system is ready to track the magnetization assignment.

   Input: User assigns magnetization vectors to various voxels by clicking on the voxels in the display and assigning the magnetization vectors.

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

Test Case Derivation: The test case is derived from the functional requirement F227, which states that the Visualization Manager shall integrate easy tracking of voxels that have been assigned magnetization vectors by adjusting the colour of the voxel to indicate assignment completeness.

How test will be performed: Assign magnetization vectors to various voxels by clicking on the voxels in the display and assigning the magnetization vectors. Verify that the system correctly updates the voxel colors to indicate magnetization assignment completeness, providing clear visual feedback about the assignment status. This will be tested manually by the developer of the system.

### 3.1.3 Integration Tests Between Import and Visualization Managers

**Data Flow and Communication Tests**

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

   Type: Functional, Dynamic, Automated

   Covers: F211, F221

   Initial State: No active project is loaded in the system, and the system is ready to import a file.

   Input: A valid CAD file is imported by the user through the import interface.

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

   Test Case Derivation: The test case is derived from the functional requirement F211 and F221, which states that the Import Manager should allow a user to import a CAD file from their personal device that will be sliced and the Visualization Manager should successfully render the 3D model without data loss or corruption.

   How test will be performed: Import a valid CAD file. Verify that the system correctly imports the CAD file and the Visualization Manager successfully renders the 3D model without data loss or corruption. This will be tested automatically by the developer of the system by running

a script that imports a CAD file and verifies that the system correctly imports the CAD file and the Visualization Manager successfully renders the 3D model without data loss or corruption.

## 3.2 Tests for Nonfunctional Requirements

### 3.2.1 Usability and Look & Feel

1. test-US-1 Design Acceptance & Intuitivity Survey

   Type: Non-Functional, Dynamic, Manual, Survey-based

   Covers: G4-HNF-1

   Initial State: Application open

   Input/Condition: User interacts with the application

   Output/Result: Recorded observations, feedback, and survey response that indicates whether the design works for the user.

   How test will be performed: Users will be exposed to the application with minimal direction and asked to perform some basic tasks. Observations such as where users get confused, when they need to consult the manual, and any comments made during use will be recorded. Afterwards, users will fill out a survey relating to their satisfaction with the interface, as well as how they would rank how intuitive functionalities are on a 5-point scale. Observations and survey responses will then be analysed, with changes implemented where necessary to improve UI design in the case that expectations are not reached. A draft of the survey is included in the appendix of this document.

   Quantifiable Metric: 90% of survey responses must rank 4 or above for intuitiveness of each feature.

2. test-US-2 Accessible Colours Check

   Type: Static, Manual

   Covers: NF223, GEN1

   Initial State: Default colours for materials and application chosen

   Input/Condition: Constrast analyser run on colours

Output/Result: Compliance level meets WCAG standards

How test will be performed: A web-based contrast checking tool such as WebAIM will be utilised with both the set of colours chosen for material IDs, as well as colours used in the UI (e.g. text, buttons, sidebars).

Acceptance Criteria: All ratios must meet 3:1 contrast ratio for WCAG 2.0 compliance.

3. test-US-3 Design Ease of Use Test

   Type: Non-Functional, Dynamic, Manual, Survey-based

   Covers: NF231, GEN2

   Initial State: Application open

   Input: User completes core tasks (import file, material/magnetization assignment, export file)

   Output: 90% of users tested complete tasks in less than MAX_INTERACTIONS.

   How test will be performed: After a short explanation of core tasks, users will be asked to perform core tasks. Clicks per task are recorded and later analysed to verify that expectations are satisfied.

4. test-US-4 View Model and Layer Focus within application

   Type: Non-Functional, Dynamic, Manual

   Covers: S3

   Initial State: Application open with an example project file pre-loaded

   Input/Condition: Developer attempts to view an individual layer of the model

   Output/Result: Application displays layer and full model separately, alongside interface to edit the layer.

   How test will be performed: Tester will select a layer and attempt to view the layer separately, as well as the model as a whole, as if they intend to edit voxels within the layer. This confirms that the primary design notion of being able to focus on a specific layer when editing voxels is satisfied.

5. test-US-5 Clear and Interpretable Warnings and Errors

   Type: Survey-based

   Covers: S4.1-7

   Initial State: Application open

   Input/Condition: User performs interactions that are known to result in error

   Output/Result: 90% of respondents agree that errors are informative

   How test will be performed: The user will be given direction in how to trigger different errors, and be exposed to all warning and error messages the system can generate. Users will explain whether they find it easy to understand or not (e.g. pass/fail). Investigation in how to reword prompts will take place for messages that under 90% of users do not understand.

   Quantifiable Metric: For all messages, 90% of users 'pass'.

6. test-US-6 Concise and Interpretable User Manual

   Type: Survey-based

   Covers: GEN3

   Initial State: Manual is completed

   Input/Condition: User reads the full user manual

   Output/Result: User agrees the manual is helpful with installation, general usage, and interpretation of common errors.

   How test will be performed: User accesses the manual and reviews it. Afterwards, they give feedback on each section of the manual, pointing out areas of confusion or notes that should be added. Feedback is then incorporated upon revision of the manual.

   Quantifiable Metric: Considered an initial success if areas of confusion for each section is less than one.

### 3.2.2 Performance & Scalability

1. test-PS-1 Scalability Validation for Import and Visualization

   Type: Non-Functional, Dynamic, Automated

   Covers: NF211, NF222, implicitly tests NF212

   Initial State: Application is open with a set of CAD files of varying sizes prepared.

   Input/Condition: Sequentially, each CAD file is imported and the resulting voxel model is viewed within the application.

   Output/Result: Application completed import within expected times given MIN_RATE and the estimated number of voxels in the resulting project file from each input. Resulting voxel models render correctly within the application

   How test will be performed: Import will be initiated on each CAD file (five files will be prepared, ranging from small (estimated 100 voxels) to very large (1.5 * MAX_VOXELS)). Timing will be started upon import, and stopped when the program reports the file was successfully imported and opens the project file. It will also be recorded whether the resulting project file renders without error. If recorded times are not what is expected given MIN_RATE and the size of the resulting model, the import module will be investigated for possible sources of optimization. If the resulting file is not openable or does not render, error messages will be investigated for possible sources of refactoring.

2. test-PS-2 Performance Validation for Import, Visualization, Editing, and Export

   Type: Non-Functional, Dynamic, Automated

   Covers: NF212, NF221, NF232, NF242

   Initial State: Application open with a set of CAD files of average sizes prepared.

   Input/Condition: Sequentially, each file is imported, core operations completed, and exported

Output/Result: Latencies for import/view change/edit/export matches those defined in the above non-functional requirements.

How test will be performed: Import will be initiated on each CAD file (three files will be prepared, ranging from small (estimated 100 voxels) to semi-large (100,000 voxels)). Timing and latency will be recorded for importing the CAD file, changing the orientation of the project file, opening layer focus, editing properties of voxel selections of varying sizes (1 voxel, 100, and a full layer), and exporting the completed project file. These times will then be compared against the defined minimum latencies and expected times given minimum processing rates. Should performance fall short of these expectations, specific sections that fail will have their code reviewed to identify areas that can be optimized.

3. test-PS-3 Resource Usage Validation Test

   Type: Non-Functional, Dynamic, Automated

   Covers: E3.4

   Initial State: Application open, with a CAD file of average size prepared.

   Input/Condition: File is imported, core operations completed, and exported

   Output/Result: Recorded CPU and GPU usage over the course of the test does not exceed thresholds.

   How test will be performed: Import will be initiated on the CAD file, and core operations will be performed (see test-PS-2's description). Throughout the test, CPU and GPU usage of the application on the host computer will be recorded. Should usage exceed defined thresholds, the operation that lead to increased usage will be investigated for precisely what part of the operation triggered it, with refactoring of the unsatisfactory module.

### 3.2.3 Compliance

1. test-CO-1 Windows and Linux Compatibility Test

Type: Non-Functional, Dynamic, Manual

Covers: E3.1

Initial State: Logged in to Windows and Linux and application is not installed

Input/Condition: User installs the application on both Windows and Linux

Output/Result: Application is successfully installed and functions as intended.

How test will be performed: The user will download the installer on each operating system, run it, and attempt to open the application. If the application opens successfully on each OS and can perform basic functionalities, the test is ruled a success.

2. test-CO-2 Import and Export Compatibility Test

   Type: Non-Functional, Dynamic, Manual

   Covers: E3.2, E3.3

   Initial State: Application open, with a CAD file of average size prepared.

   Input/Condition: File is imported, core operations completed, and exported

   Output/Result: Application converts input file correctly, exported file is correctly formatted

   How test will be performed: The tester selects the CAD file to be imported, and verifies that the resulting project file is correctly generated (e.g. resulting voxel model is an accurate approximation of the original model). The tester then assigns dummy data to each layer and exports the project file to the 3D-printer ready format. The resulting file is then checked for completeness and correct formatting.

3. test-CO-3 Project File Fail-safe

   Type: Non-Functional, Dynamic, Manual

   Covers: NF241

Initial State: Application open, with a completed project file prepared.

Input/Condition: File is open, export is initiated and interrupted

Output/Result: Project file remains unaltered

How test will be performed: The tester begins the export process on the loaded project file, before intentionally ending the program in the middle of the export. The project file is then checked against a copy of itself made before the test to see if there are any differences; if there are no differences, the test is considered a success.

4. test-CO-4 PEP8 Standard Compliance

   Type: Static Analysis

   Covers: DP.10

   Initial State: Application is fully implemented

   Input/Condition: Codebase is scanned with a linter (e.g. Pylint)

   Output/Result: Code complies with PEP8 Standard

   How test will be performed: Developer inputs the entirety of the application's backend code into a linting program and takes note of all warnings. All areas of note are then addressed, with possible refactorings or small fixes resulting.

### 3.2.4 Maintainability

1. test-MA-1 Maintainable Codebase

   Type: Static Analysis

   Covers: DP.10, GEN4

   Initial State: Application is fully implemented, and related documentation is complete

   Input/Condition: Developers review all code and documentation

   Output/Result: Code is sufficiently modular, and fully commented; associated documentation is easy to understand

How test will be performed: Reviewers inspect all code, evaluating based on modularity (functionality is sufficiently separated, archicture supports editing one section not mandating updates throughout entire codebase), comments (both function/class headers and notes within functions, explaining complex or unclear code), and naming conventions (names of functions, classes, variables are consistent and clear). Associated documentation is reviewed separately, evaluating based on completeness (all modules are detailed) and readability (jargon is clearly defined; non-team members can understand the content).

### 3.2.5 Security

Regarding the requirements listed in the Hazard Analysis document, they will not be explicitly tested for; examining each, we see that they are implicitly tested for by the following system tests and/or unit tests.

- SCR1: Implicitly tested for by test-FR-IM-3.
- SCR2: Infeasible to test at runtime as this would require access to hardware we have assumed we will not be working with within the SRS. Will be verified by unit tests that simulate a system with less memory.
- SCR3: Will be verified by unit tests related to the Editing manager that verify latency.
- SCR4: Implicitly tested for by test-FR-XM-1 and test-FR-XM-2.
- SCR5: Implicitly tested for by test FR-XM-3.
- SCR6: Implicitly tested for by test-FR-EM-6.
- SCR7: Implicitly tested for by test-FR-EM-7.
- SCR8: Will be verified by unit tests related to both the Visualization and Editing managers that verify limits of voxel selection.
- SCR9: Implicitly tested for by test-FR-XM-5.

## 3.3 Traceability Between Test Cases and Requirements

Table 4: Import Manager Tests and Corresponding Requirements

| Test ID (test-) | Requirement and/or Relevant Documentation |
|---|---|
| FR-IM-1 | F211 |
| FR-IM-2 | F212 |
| FR-IM-3 | F211 |
| FR-IM-4 | F213 |
| FR-IM-5 | F214 |
| FR-IM-6 | F215 |
| FR-IM-7 | F216 |

Table 5: Visualization Manager Tests and Corresponding Requirements

| Test ID (test-) | Requirement and/or Relevant Documentation |
|---|---|
| FR-VM-1 | F221 |
| FR-VM-2 | F222 |
| FR-VM-3 | F223 |
| FR-VM-4 | F224 |
| FR-VM-5 | F225 |
| FR-VM-6 | F226 |
| FR-VM-7 | F227 |
| FR-INT-1 | F211, F221 |

Table 6: Non-Functional Tests and Corresponding Requirements

| Test ID (test-) | Requirement and/or Relevant Documentation |
|---|---|
| US-1 | G4-HNF-1 |
| US-2 | NF223, GEN1 |
| US-3 | NF231, GEN2 |
| US-4 | S3 |
| US-5 | S4.1-7 |
| US-6 | GEN3 |
| PS-1 | NF211, NF222, NF212 |
| PS-2 | NF212, NF221, NF232, NF242 |
| PS-3 | E3.4 |
| CO-1 | E3.1 |
| CO-2 | E3.2, E3.3 |
| CO-3 | NF241 |
| CO-4 | DP.10 |
| MA-1 | DP.10, GEN4 |

# 4 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]

## 4.1 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]

## 4.2 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]

### 4.2.1 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. ...

#### 4.2.2 Module 2

...

### 4.3 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]

#### 4.3.1 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:

### 4.3.2 Module ?

...

## 4.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

# 5 Appendix

This is where you can place additional information.

## 5.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

- **OPTIMAL_VOXEL_SIZE** = 5.5nm

  This was derived using the optimal voxel size specified by the stakeholder.

- **MAX_VOXELS** = 13,996,800,000 voxels

  This considers the maximum number of voxels contained within the entire model.

- **MAX_LAYER_DISPLAY** = 518,400 voxels

  This was derived assuming 960 voxels x 540 voxels per layer. It considers the maximum number of voxels contained within a single layer for a single display window.

- **MAX_DISPLAY** = 103,680,000 voxels

  This was derived assuming 960 voxels x 540 voxels per layer. It considers the maximum number of voxels that will be displayed at a given time.

- **MIN_RATE** = 500,000 voxels per second

  This considers how fast voxel data will be generated during file import and export.

- **MAX_EDIT_LATENCY** = 1 second

  This considers how quickly the system should respond to model modifications performed by the user.

- **MAX_VIEW_LATENCY** = 500 ms

  This considers how quickly the system should respond to changes in model view orientation by the user.

- **MAX_INTERACTIONS** = 5

  This considers the maximum number of steps (e.g. clicks, enter details into a text box) a user should take to complete a part of a task.

## 5.2 Usability Survey Questions?

Each of the following questions is to be answered via a 5-point scale, with 1 being the least and 5 being the most except where otherwise listed. This list may change or be updated as functionalities are added or removed.

1. How difficult was it for you to navigate the overall interface? (5 - very easily, 1 - with difficulty)
2. Do you consider the current method of moving the model to inspect it intuitive?
3. How natural do you find the layer select functionality?
4. How intuitive did you find the method of selecting voxels?
5. How easy was it for you to assign a magnetization vector to a set of voxels?
6. How easy was it for you to assign a material to a set of voxels?
7. Do you find the current signifiers for completed voxels helpful? (5 - very helpful, 1 - not very helpful)
8. How intuitive did you find the file importing process?
9. How intuitive did you find the file exporting process?

# Appendix — Reflection

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?

   Overall this deliverable went well. We were able to write up all the parts of minimal issues. The peer feedback we received during the TA review session was also helpful in ensuring that we were on the right track. One problem that we had was the time constraints, since all the members were busy with other assignments and midterms, so we had to work on this deliverable in a timely manner. Another thing that went well was our ability to communicate with each other, and ensuring that everyone was on the same page.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   **Omar:**
   **Daniel:**

   **Andrew:**

   **Olivia:** One of the challenges that I came across was trying to decide on the best code coverage tool for the frontend of the system. The process of 3D rendering is fairly complex, which means it will likely be

fairly difficult to test. Without any prior experience in the development of 3D rendering, I found it hard to understand which tools would be able to provide enough coverage given the needs of our code. It helped a little once we decided what language we were going to use to develop our frontend. In the end, it still required a fair amount of research, and I decided to just choose one that I think right now would be a good choice. I've come to realize that as the project progresses, it's likely that things will change. This might mean that the chosen coverage tool might change. It doesn't have to be perfect because adapting to changing needs is part of the development process.

**Khalid:** I was primarily responsible for writing the system tests for the Import and Visualization Managers. The main challenges that I had when writing up this section was ensuring that the tests were comprehensive and covered all the functional requirements that were listed for both manaers on the SRS. Additionally, I had to ensure that the tests were easy to understand and follow, and that they were able to catch all the potential errors and edge cases.

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.

   There are five major knowledges/skills our team will need to acquire:

   - **Knowledge of PyTest Unit testing framework.** With PyTest being the main testing framework for the application's backend, all team members will need to know its conventions.
   - **Knowledge of Jest Unit testing framework.** Similar to above reasoning, Jest is the identified framework for testing the application's frontend.
   - **Organizing Usability tests.** Beyond developing the test with the users themselves, knowing the right observations to take note of is important to integrating feedback correctly.
   - **Mutation testing skills.** Mutation testing is only as good as the variety of mutants produced.

- **How to conduct a code review.** Given the potential scale of the codebase and the limited time of team members, effective, time-efficient code reviews will be essential.

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?

    - **Knowledge of PyTest:** Approaches include reviewing existing online tutorials and documentation with the goal of achieving a broad understanding of the framework, with an alternative option of experimenting with the tool with a toy example program to gain practical experience.
    - **Knowledge of Jest:** Similar to the above, a first option is to review documentation as needed, and a second to experiment creating tests in the framework to gain experience.
    - **Usability Tests:** A first approach is to review course content from Human-Computer Interfaces, a course that covers how to observe users and derive feedback from not just what they say, but how they behave with the interface, as well as non-obvious prompting methods. Another option is to review online material about how to conduct user testing that focuses on usability, and to possibly conduct a trial run with peers.
    - **Mutation testing:** Approaches include a reviewing content from Software Testing, a course all team members have taken that not only covers how to come up with test cases, but mutation testing as well, and how to develop varying mutants. Another option is to review academic papers detailing mutatation testing strategies, some of which can be found in the course materials of Software Testing.
    - **Code Reviews:** Software Testing also included content detailing how code reviews are conducted in formal contexts, which could be a helpful resource; ideas could be adapted to this project and provide a standard to consult. Another option involves running code reviews in other projects team members are involved in through-

out term to gain experience, which would benefit both this project and those involved.

**Omar:**
**Daniel:**
**Andrew:**
**Olivia:**
*Knowledge of PyTest:* Given that I have minimal experience working with PyTest, I will likely integrate a mixture of the two approaches. To ensure I have a solid foundation of knowledge regarding the framework, I'll start with reviewing online documents. However, once I have that foundational knowledge, I feel that I will learn much more through engaging with the tool.
*Knowledge of Jest:* Similar to my answer above, I will probably combine the approaches. Since I expect the testing of 3D rendition to be much more complex, I feel like I will have to spend more time reviewing the documentation to adequately understand the testing information. However, once I have a good idea of the basics of Jest testing, I feel that active engagement with the tool is better for me, as I tend to have a higher retention rate for what I learn.
*Usability tests:* Due to the fact that this system will only be used by a fairly small user group on a regular basis, I feel that it is more beneficial to focus my learning on how to conduct real user testing trial runs. While generalized fundamentals are important in creating a usable design, it might better suit the scope of the project to focus on usability in specific regards to the select group of current users.
*Mutation testing:* I believe that I will use a combination of the two approaches. When looking to recall general mutation testing information at the beginning stages, reviewing course material will be helpful due to the familiarity with the content structure. However, when deriving the specific mutations for these test cases, I think that engaging with papers and online resources will allow me to better tailor mutation cases to the specific scope of this project.
*Code reviews:* I personally would like to focus on doing code reviews with other team members, as I find the fresh perspective can be helpful. When I spend several hours on a project, I find that it is easy for me to grow complacent during review due to loss of motivation. Therefore, I believe code reviews would be beneficial to actively re-engage myself

with the project and take the opportunity to learn from others.

**Khalid:**
*Knowledge of PyTest:* I will use the online tutorials and documentation to help me gain a broading understanding of the testing framework. I think this will be a good starting point, as it will help me understand the framework theoretically, and then I can apply it to the project to gain practical experience.

*Knowledge of Jest:* Similar to the above, I will also use the online tutorials and documentation to help me gain a broading understanding of the testing framework. I will also experiment with the tool with a toy example program to gain practical experience.

*Usability tests:* For this skill, I will look over my notes from the 4HC3 (Human-Computer Interfaces) course, which covers how to observe users and derive feedback from both their responses. This was I can recall the content and apply it to the project.

*Mutation testing:* Similar to the above, I will look over my notes from the Software Testing course that covers not just test case development but also mutation testing strategies and how to develop varying mutants. I will also experiement with a simple program to gain practical experience.

*Code reviews:* I will look over my notes from the Software Testing course that covers how to conduct a code review. I will also experiment using some old code from other projects just to get a feel for the process.