System Verification and Validation Plan for ImgBeamer

Joachim de Fourestier

February 27, 2023

1 Revision History

Date	Version	Notes
2023/02/17	1.0	First version
2023/02/20	1.0.1	fix minor issues in T2, T3, T15, and T17
	1.0.1	Minor issues in general information
	1.0.2	Fix grammar in sections 4.2, 4.3, 4.5, and 4.6.
	1.0.3	Fix formatting and grammar in sections 5, 5.1.1, 5.1.2, and 5.1.3.
	1.0.4	Fix grammar in Section 5.2.
	1.0.5	Tweaks to Section 3.2.
	1.0.6	Added missing references to MG and MIS.
2023/02/22	1.0.7	Provide specific values for most of the tests.
	1.0.8	Review the design and VnV verification plans.
2023/02/27	1.0.9	Improve wording for Section 4.5.
	1.0.10	Tweak tests in sections 5.1.2 and 5.1.3.

Contents

List of Tables

2 Symbols, Abbreviations and Acronyms

symbol	description
MIS	Module Interface Specification
MG	Module Guide
MSE	Mean Squared Error
RMSE	Root Mean Square Error
RMSPE	Root Mean Square Percentage Error
NFR	Nonfunctional Requirement
GUI	Graphical User Interface
R	Requirement
ROI	Region of Interest
SRS	Software Requirements Specification
T	Test
U	Unit test
UI	User Interface
VnV	Verification and Validation

3 General Information

This document lays out the Verification and Validation (VnV) plan of ImgBeamer as described by the Software Requirements Specification (SRS) [4]. Testing of the software and its components will be conducted to build confidence in its usability, accuracy, and, ultimately, whether it meets the requirements from the SRS.

3.1 Summary

ImgBeamer software aims to be a demonstration and learning tool of how a scanning electron microscope (SEM) formulates an image. The idea is to help visualize the influence trends of imaging parameters (as defined in the SRS) on image quality (or clarity) and resolution.

3.2 Objectives

The objective is to produce a tool that is easy to use (*usability*). It should feel intuitive to the user and should provide easy-to-understand information. This is so that the user may identify any trends in how the final image is affected with relative ease and confidence. As a secondary goal, the *accuracy* in the trends are important. Due to the subjective and qualitative nature of evaluating image quality, the metric used should be straight-forward and provide some objectivity (for quantitative comparison) and assurance to the users.

3.3 Relevant Documentation

There are multiple design documents that provide the important and intimate details to understand some of the concepts that are being tested. These documents are the following:

- Problem Statement [3]
- SRS [4]
- VnV Report [5]
- MG [1]
- MIS [2]

4 Plan

In this section, multiple plans are described to test and inspect the software with an emphasis on *usability*. Multiple approaches and perspectives will be employed by the VnV team (4.1) to help build confidence in the requirements, to avoid any missed important details, and to deliver on the qualities as mentioned in the objective (3.2).

4.1 Verification and Validation Team

The members of the VnV team as well their individual roles are listed in the following table:

Role	Name
Project Supervisor	Dr. Spencer Smith
Author	Joachim de Fourestier
Domain Expert	Karen Wang
SRS Reviewer	Jason Balaci
VnV Plan Reviewer	Sam Crawford
MG + MIS Reviewer	Lesley Wheat
Expert Consultant	Dr. Nabil Bassim
Expert Consultant	Michael W. Phaneuf

Table 1: Table of the VnV Team Members

[Dr. Bassim is my PhD Supervisor, and Mr. Phaneuf is my co-supervisor (employer/industry), since I am doing an industrial PhD. Much of this project was originally conceptualized with the help and guidance of Mr. Phaneuf. —Author]

4.2 SRS Verification Plan

The SRS will be reviewed by the project supervisor, the SRS reviewer and the author. Some input may be given by the expert consultants if time permits. Most of the feedback has been provided as issues on GitHub, as annotated documents, or by verbal exchange. Throughout the development of the project, the author is expected to make changes needed to the resolve

the issues. The reviewers may refer to the SRS checklist [6]. The key objective is to verify that the software requirements and the documentation are sound and coherent to the intended audience as defined in the SRS.

4.3 Design Verification Plan

Much of the conceptualization was done after having multiple discussions with the expert consultants and having done literature review for preexisting tools and documentation relevant to the project. Decisions were made with a focus on the usability, meaning little to no setup required. The design and implementation is documented in the MG[1]/MIS[2]. The VnV team, as well as any welcomed volunteers, will provide their input (through GitHub issues). Eventually, the project may be published in a journal where the software ImgBeamer and its accompanying documentation will likely be rigorously reviewed; The expert consultants will review and evaluate the software to establish whether it is fit for public use.

[Not sure what was appropriate to put here. —Author]

4.4 Verification and Validation Plan Verification Plan

The goal is to uncover any mistakes and reveal any risks (such as misconceptions or coverage gaps) through the supervision and review of the VnV team members. Once most of the work has been done, the work and accompanying documentation shall undergo a vetting process: the VnV team will check whether the documented testing plans and verification process have been accomplished and the requirements fulfilled. The author will then review the documents once more against the VnV checklist [7], before a final review by the rest of the VnV team.

4.5 Implementation Verification Plan

Much of the software will be tested manually by users. This will include checking for any inconsistencies, bugs in the graphical user interface (GUI), and unexpected artifacts in images produced. The image metrics will be tested using unit tests (Section 6). For the all the code implemented, linters will be used as mentioned in Section 4.6. As a control for the image metrics, they will be calculated using the ground truth image as both the input and

the reference image (effectively comparing it to itself) to rule out any baseline or unexpected factors in the implementations themselves.

4.6 Automated Testing and Verification Tools

The image quality metric shall be unit tested using pytest for automated testing of any algorithms implemented in Python and QUnit shall be used for those implemented in JavaScript. The unit tests are listed in Section 6. As for linter, flake8 shall be used for Python code and ESLint for JavaScript code. The sewar python package will be used as a reference implementation in Python for the image quality metrics.

4.7 Software Validation Plan

A *usability* survey will be conducted to evaluate the user experience and whether the GUI is intuitive enough to the intended users as described in the SRS [4]. An *accuracy* survey will be conducted to assess the user-perceived image quality. The trends identified in the surveys results will be compared to the calculated image metrics. As a control for the images produced, a manual and an automated test will be conducted to verify if an image identical (or with unperceivable difference) to ground truth image can be reproduced. The compared images shall be in the *accuracy* survey for confirmation. This can be compared as well using the image metrics.

[I think I might be confusing the Software and the Implementation validation plans... —Author]

5 System Test Description

In this section, the system tests that will be conducted are described in detail. These tests will be used to verify the fulfillment of the requirements as listed in the SRS [4]. Most, if not all, of the tests listed here will be manually performed. Automatic tests will be unit tests as described in Section 6.

5.1 Tests for Functional Requirements

The tests present will verify whether the functional requirements (as listed in the SRS [4]) are met and validate that the outputs are as expected.

5.1.1 Image Import and Export

To satisfy R1 from the SRS [4], an input image for the following formats listed below shall be accepted provided they follow the input data constraints. Some preloaded example images shall be included and be available to the user for use as well. As well as for R6, the software should allow for export of the resulting image.

- PNG
- JPG
- BMP

1. **T1**: Test import for PNG/JPG/BMP format

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: A non-corrupt (valid) PNG, JPG, or BMP image file.

   Output: The image should be visible and be displayed as expected as the ground truth image.

   Test Case Derivation: These are some of the most common image file formats and should be compatible with the software.

   How test will be performed: The user will click the "Load image" button and select an image to import.

2. **T2**: Test PNG Image Export

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: User clicks the *Export* button.

   Output: The output image file should look the same as what is displayed in the "Resulting Image" display.

   Test Case Derivation: [Not sure what this means, and how this differs from what's explained Output. —Author]

How test will be performed: The user will click the "Export image" button and choose where to save the image file.

5.1.2 Spot Profile and Imaging Parameters

To satisfy R2 and R5 from the SRS [4], the software should accept input from the user to change the width, height, and rotation which, define an ellipsoid shape. This shape is then used as the spot to sample the ground truth image. To satisfy R3, the software should accept user input for a real positive number for the pixel size. To satisfy R4, the user should be able to specify a subregion (or ROI) for processing.

1. **T3**: Spot Width and Height - Exact-sampling

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: Both width and height are set to "100%".

   Output: The spot layout should reflect these changes and display an updated arrangement with the given shape. The resulting image should exhibit the expected **exact-sampling** case as documented by the SRS figures [4].

   Test Case Derivation: [not needed in this case? —Author]

   How test will be performed: Either by scrolling in the visual spot shape UI or by number input in the GUI.

2. **T4**: Spot Width and Height - Under-sampling

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: Both width and height are set to "10%".

   Output: The spot layout should reflect these changes and display an updated arrangement with the given shape. The resulting image should exhibit the expected **under-sampling** case as documented by the SRS figures [4].

   Test Case Derivation: [not needed in this case? —Author]

How test will be performed: Either by scrolling in the visual spot shape UI or by number input in the GUI.

3. **T5**: Spot Width and Height - Over-sampling

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: Both width and height are set to "500%".

   Output: The spot layout should reflect these changes and display an updated arrangement with the given shape. The resulting image should exhibit the expected **over-sampling** case as documented by the SRS figures [4].

   Test Case Derivation: [not needed in this case? —Author]

   How test will be performed: Either by scrolling in the visual spot shape UI or by number input in the GUI.

4. **T6**: Spot Rotation - Astigmatism

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The angle is set to "45 degrees", the width is set "60%", and the height is set to "500%".

   Output: The spot layout should reflect the updated arrangement with the rotated spots. The resulting image should exhibit the expected **astigmatism** effects as documented in the SRS figures [4].

   Test Case Derivation: [see output? —Author]

   How test will be performed: By dragging the rotation node in the visual spot shape UI.

5. **T7**: Raster Grid / Pixel Size

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The rows and columns are set to "8" for the raster grid.

Output: The resulting image resolution (not image size) should be an "8" by "8" grid image. The "8" rows and "8" columns should be represented in the spot layout display.

Test Case Derivation: [see output? —Author]

How test will be performed: The user shall input the number of rows and columns.

6. **T8**: Subregion / ROI

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The magnification is set to "2x" and the raster grid is set to "8 x 8".

   Output: Both the spot layout and the resulting subregion displays should have "8 x 8" resolution (rows and columns). The resulting image preview should have "16 x 16" resolution (rows and columns).

   Test Case Derivation: [see output? —Author]

   How test will be performed: The user scroll/zoom in the Subregion (ROI) View display until a magnification of 2x is reached.

5.1.3 Image Quality Metric

To satisfy R7, the user should be able to see a larger number for a resulting image that is closer to ground truth and a smaller number otherwise.

1. **T9**: High metric value (approximate to exact-sampling)

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The Spot size is set to "120%" with a perfectly circular spot shape.

   Output: The score should be greater than "0.8650".

   Test Case Derivation: The resulting image should look somewhat sharp as shown in the SRS figures [4].

2. **T10**: Low metric value (under-sampling)

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The Spot size is set to "10%" with a perfectly circular spot shape.

   Output: The score should be less than "0.8601".

   Test Case Derivation: The resulting image should look somewhat overly "sharp" or pixelated as shown in the SRS figures [4].

3. **T11**: Low metric value (over-sampling)

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: The Spot size is set to "500%" with a perfectly circular spot shape.

   Output: The score should be less than "0.8601".

   Test Case Derivation: The resulting image should look somewhat "blurry" or "defocused" as shown in the SRS figures [4].

4. **T12**: Control metric value

   Control: Manual

   Initial State: ImgBeamer loaded and idle.

   Input: Spot size of a 100%, the same pixel size (image resolution) as the ground truth image, and perfectly circular spot shape.

   Output: A number that is near the best or maximum quality metric value. The resulting image should nearly identical to the ground truth image. The score should be greater than "0.9800".

   Test Case Derivation: The resulting image should be identical or have unperceivable differences to the naked eye.

5.2 Tests for Nonfunctional Requirements

The following tests will check if the nonfunctional requirements, as defined in the SRS [4], are met. The emphasis is on the *usability* (NFR2) of the software. In the case of ImgBeamer, simplicity is more important than the actual complexity or correctness. To satisfy the *accuracy* (NFR1) requirement, an image quality and clarity survey shall be conducted to establish what is perceived by the user and the image metric of what a "better" image is. To satisfy the *maintainability* (NFR3) requirement, the code should follow a consistent style and reasonably pass static code analysis as described below. As for *portability* (NFR4), the user should be run the software on their platform and environment of choice.

5.2.1 Usability Testing

1. **T13**: Usability Survey

   Type: Manual

   Initial State: None

   Input/Condition: A group of intended users, the SRS [4], and the survey questions on topics related to *usability*.

   Output/Result: The completed user surveys (see 8.1).

   How test will be performed: The users will be given a series of questions to evaluate the ease-of-use, whether the GUI is intuitive or confusing, etc.

5.2.2 Accuracy Testing

1. **T14**: Image Metric Survey

   Type: Manual

   Initial State: None

   Input/Condition: A group of intended users, the SRS [4], a list of images of varying quality, and the survey questions on topics related to the image quality and clarity.

   Output/Result: The completed user surveys (see 8.2).

How test will be performed: The users will be given a series of questions to evaluate a given list of images, rank them based on detail and information content, etc.

5.2.3 Maintainability Testing

1. **T15**: Static code analysis

   Type: Manual

   Initial State: None

   Input/Condition: The software code will be tested through the use **pylint** for any Python code used and **ESLint** for any JavaScript code used.

   Output/Result: The code linters should list little to no warnings (less than 10 unique warnings, on the default strictness setting), and zero errors.

   How test will be performed: Execute the **pylint** tool on all written Python code and the **ESLint** tool on all written JavaScript code.

2. **T16**: Code Review

   Type: Manual

   Initial State: None

   Input/Condition: The software source code and the code review checklist (see 8.3)

   Output/Result: The code should respect the defined code review checklist.

   How test will be performed: The author with walkthrough the code manually and check for qualities such as consistent styling, loose variables, variable scoping, unreachable code, sufficient commenting, etc.

5.2.4 Portability Testing

1. **T17**: Various Platforms and Environments

Type: Manual

Initial State: None

Input/Condition: Given any of the supported platforms, the user should be able to set up and run the software with relative ease.

Output/Result: The software runs on the platform of choice with little to no setup including installation, meaning less than 4 clicks and under 3 minutes set up time.

How test will be performed: The user will download and run the software on their platform of choice.

6 Unit Test Description

This section cannot not be filled or completed until the MIS (detailed design document) has been completed. [I did do some image metrics testing that can at least be partially documented here... —Author]

6.1 Unit Testing Scope

[Other than the image metrics, I am not sure what else could be unit tested as of now? —Author]

[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]

6.2 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]

6.2.1 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. ...

6.3 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]

6.3.1 Module: Image Quality Metric

[I will likely use the "reverse" / "inverse" value as the image metric: $Metric = 1 - RMSPE$, where the "RMSPE" is RMSE normalized to a value between 0 and 1. —Author]

1. **U1**: Lowest/Highest MSE, RMSE and RMSPE scores

   Type: Automatic

   Initial State: [Not running? —Author]

   Input/Condition: The ground truth images and multiple resampled images of varying and known quality.

   Output/Result: The higher quality image should return the lowest error (i.e. lowest MSE, RMSE, RMSPE) scores.

   How test will be performed: Executing a suite of tests for the multiple images.

2. **U2**: Control testing

   Type: Automatic

   Initial State: [Not running? —Author]

   Input: The ground truth image.

   Output: an error score of zero [or the lowest error possible depending on the given metric —Author].

   How test will be performed: Executing a suite of tests using the different metric to compare the ground truth image to itself.

3. test-id#

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output:

   How test will be performed:

6.3.2 Module ?

...

7 Traceability Between Test Cases and Requirements

Traceability matrices are used to simplify the process of identifying what needs to be changed if a component is modified. An "X" is used to indicate links between items in the table. When a component is changed, the elements marked with an "X" might need to be updated as well.

	R1	R2	R3	R4	R5	R6	R7	NFR1	NFR2	NFR3	NFR4
T1	X										
T2						X					
T3		X			X						
T4		X			X						
T5		X			X						
T6		X			X						
T7			X		X						
T8				X	X						
T9							X				
T10							X				
T11							X				
T12							X				
T13									X		
T14								X			
T15										X	
T16										X	
T17											X
U1							X				X
U2							X				X

Table 2: Traceability Matrix Showing the Connections Between the Tests and Requirements

References

[1] J. de Fourestier. Module guide for imgbeamer, 2023. URL https://github.com/joedf/CAS741_w23/blob/main/docs/Design/SoftArchitecture/MG.pdf.

[2] J. de Fourestier. Module interface specification for imgbeamer, 2023. URL https://github.com/joedf/CAS741_w23/blob/main/docs/Design/SoftDetailedDes/MIS.pdf.

[3] J. de Fourestier. Problem statement and goals: Imgbeamer, 2023. URL https://github.com/joedf/CAS741_w23/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf.

[4] J. de Fourestier. Software requirements specification for imgbeamer: Scanning electron microscope image formation, 2023. URL https://github.com/joedf/CAS741_w23/blob/main/docs/SRS/SRS.pdf.

[5] J. de Fourestier. Verification and validation report: Imgbeamer, 2023. URL https://github.com/joedf/CAS741_w23/blob/main/docs/VnVReport/VnVReport.pdf.

[6] Dr.S̃. Smith. Srs and ca checklist, 2022. URL https://github.com/smiths/capTemplate/blob/main/docs/Checklists/SRS-Checklist.pdf.

[7] Dr.S̃. Smith. System verification and validation plan checklist, 2022. URL https://github.com/smiths/capTemplate/blob/main/docs/Checklists/VnV-Checklist.pdf.

8 Appendix

This section contains the *usability* and image quality (metric *accuracy*) user survey questions, as well as a code review checklist.

8.1 Usability Survey Questions

- What web browser do you use and what version is it?
- What operating system do you use?
- What is your screen resolution?
- Are you using a mouse and keyboard or a touchscreen?
- How much time (in seconds or minutes) did it take you to find the function you were looking for?
- What was the most useful feature for you?
- On a scale from 1 to 10, how easy was it to set up and start the software?
- On a scale from 1 to 10, how easy was it to use the software?
- What was the least enjoyable feature to use?
- Was everything legible and clearly displayed? If not, what was not?
- Would you consider using the software as a learning or teaching tool?
- Do you have any extra comments or thoughts you'd like to share?

8.2 Image Quality Survey Questions

Given multiple images that are numbered and a ground truth image, the user shall answer the following questions?

- Which image looks the seems to be the clearest or is of the high quality in your opinion?
- Which image provide the most information or detail?

- Which image is the most similar to or the most representative of the ground truth image?
- Which image is the least pleasant in your opinion?
- Rank the images from the lowest quality (least clear) to the highest quality (most clear) in your opinion?
- Rank the images from least to most informative (or has most of the details of the ground truth image)?
- The user is shown the ground truth image and the reproduced control image (generated using the same image resolution and exact-sampling). Do these images look identical to you? If not, on a scale of 1 to 10, how similar are these images?

8.3 Code Review Checklist

- Does the code follow a consistent style?
- Are there any variables that should not be global?
- Are function names not too long (i.e. less than 40 characters)?
- Are most function well named and clear in what they do?
- Are there any long lines of that can be split into multiple lines?
- Are global variables and functions documented?
- Is each function not too long and sufficiently broken down to accomplish a single task?
- Are comments plentiful, but not noisy?
- Is the code sorted into separate files where reasonable?
- Is there any duplicate that code be avoided?
- Is there any use of jargon, domain-specific, or unclear terms?
- Is all of the code reachable?

- Could programmer other than the author read the code and briefly understand it?
- Is there any leftover commented code that is not useful?