System Verification and Validation Plan for Measuring Microstructure Changes During Thermal Treatment

Team #30, ReSprint
Edwin Do
Joseph Braun
Timothy Chen
Abdul Nour Seddiki
Tyler Magarelli

March 8, 2023

Table 1: **Revision History**

Date	**Developer**	**Notes/Changes**
Oct 31, 2022	Timothy Chen	Added to 5.2, 5.3, 7.2
Oct 31, 2022	Edwin Do	Added section 4 for V&V Plan
Nov 1, 2022	Abdul Nour Seddiki	Added to 5.1, 5.3
Nov 2, 2022	Joseph Braun	Added Section 3
Nov 2, 2022	Edwin Do	Added more content to section 4
Mar 4, 2023	Edwin Do	Revised tests for non functional requirements
Mar 4, 2023	Timothy Chen	Revised Usability and Performance NFR test
Mar 6, 2023	Abdul Nour Seddiki	Revised tests for Functional Requirements & fixed formatting
Mar 8, 2023	Edwin Do, Timothy Chen	Further revision of tests for non functional requirements
Mar 8, 2023	Edwin Do	Added unit tests for calculation and validation
Mar 8, 2023	Tyler Magarelli	Updated user experience survey for section 6.2
Mar 8, 2023	Timothy Chen	Added Unit Test Plan for Input Communication, Remote Access, Current State

Contents

1 Symbols, Abbreviations and Acronyms **iv**

2 General Information **1**
- 2.1 Summary . . . 1
- 2.2 Objectives . . . 1
- 2.3 Relevant Documentation . . . 1

3 Plan **1**
- 3.1 Verification and Validation Team . . . 2
- 3.2 SRS Verification Plan . . . 3
- 3.3 Design Verification Plan . . . 4
- 3.4 Implementation Verification Plan . . . 4
- 3.5 Automated Testing and Verification Tools . . . 4
- 3.6 Software Validation Plan . . . 5

4 System Test Description **5**
- 4.1 Tests for Functional Requirements . . . 5
- 4.2 Tests for Nonfunctional Requirements . . . 8
  - 4.2.1 Appearance Test . . . 8
  - 4.2.2 Usability Test . . . 9
  - 4.2.3 Performance Test . . . 11
  - 4.2.4 Operational Test . . . 12
  - 4.2.5 Maintainability/Support Test . . . 13
  - 4.2.6 Security Test . . . 13
  - 4.2.7 Cultural Test . . . 14
  - 4.2.8 Health and Safety Test . . . 14
- 4.3 Traceability Between Test Cases and Requirements . . . 15

5 Unit Test Description **17**
- 5.1 Unit Testing Scope . . . 17
- 5.2 Tests for Functional Requirements . . . 17
  - 5.2.1 Input Communication Module . . . 17
  - 5.2.2 Remote Access Module . . . 19
  - 5.2.3 Current State Module . . . 21
  - 5.2.4 Calculation Module . . . 26
  - 5.2.5 User Input Validation Module . . . 27

5.2.6 Hardware Input Validation Module 29
5.3 Traceability Between Test Cases and Modules 30

6 Appendix **31**
6.1 Symbolic Parameters 31
6.2 Survey Questions 32

List of Tables

1 **Revision History** i
2 Team and Responsibilities 3
3 Requirements Traceability 15

1 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
SRS	Software Requirements Specifications
VS	Visual Studio
UWP	Universal Windows Platform
MIN_USER_ACCEPT_RATE	90% - minimum acceptance rate
TARGET_TIME	60 seconds
INTERACT_TIME	5 seconds
MAX_MISTAKE	2
MAX_SIZE	8GB
MIN_UPTIME	30 minutes
MIN_SAMPLE_RATE	60 samples per second
TIME_ACCEPTED	1 second
ACCEPTED_SIGFIG	3 decimals

[symbols, abbreviations or acronyms – you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

2 General Information

2.1 Summary

The purpose of this document is to provide a detailed plan for the testing of our system. This will include:

- Verification and Validation Plan
- System Test Description
- Unit Test Description

2.2 Objectives

The objectives of testing are to ensure that all functional and non-functional requirements of the system are being met. It is important to include both unit tests as well as system tests, as issues may arise when components are connected together in the system.

2.3 Relevant Documentation

Relevant documentation includes:

- SRS
- MIS
- MG

3 Plan

In this section of planning, it will outline our approaches to cover the requirements outlined in various areas such as the SRS document, Hazard analysis document, our implementation and design. Tools that will be used for automated unit testing and linting will also be introduced.

The following topics will be covered:

- The verification and validation team along with their respective responsibilities
- Our approach towards the SRS Verification plan
- Our approach towards the Design Verification plan
- Implementation verification plan
- Any testing and verification tools we plan to use

3.1 Verification and Validation Team

Below is a table outlining the members of the Verification and Validation team along with their respective responsibilities. Note that the listed responsibilities are only used as a guideline, responsibilities can shift between team members on a as-per-needed basis.

Table 2: Team and Responsibilities

Team Member	Role Name	Responsibilities
Edwin Do	Software and SRS Tester	Ensures that all requirements are valid and verified under the scope of software capabilities in this project and the SRS
Timothy Chen	Software Tester	Ensures that all requirements are met and verified under the scope of software capabilities in this project
Tyler Magarelli	Software Tester	Ensures that all requirements are met and verified under the scope of software capabilities in this project
Joseph Braun	Hardware Tester	Ensures that all requirements are met and verified under the scope of hardware capabilities in this project
Abdul Nour Seddiki	Hardware Tester	Ensures that all requirements are met and verified under the scope of hardware capabilities in this project
Dr. Hatem Zurob	Supervisor	Ensures that all requirements are valid and meets the expected result

3.2 SRS Verification Plan

To verify our SRS, our team intends on revisiting the SRS document on a bi-weekly basis to verify that the requirements are up to date and in sync with the project goal. This will also allow us to cover any newly discover risks or hazards which will also be reflected in the Hazard Analysis document. Any new changes within the two week window will be noted and discussed at the end to see if additional changes to the SRS document would be necessary. At each bi-weekly review, the team also plans on using the SRS checklist as a guideline throughout the meeting.

In addition, the team will use ad hoc feedback from reviewers such as classmates from our 4GA6 Capstone class as well as instructor, supervisor and

teaching assistants. This will act as a supplementary addition if any element of the SRS is out of date or missing.

3.3 Design Verification Plan

Our plans to verify our design document includes using the MIS checklist and reviews from our classmates. The MIS checklist will help our team ensure that the design logic helps us meet the requirements specified in the SRS document and cover any hazards or risks outlined in the Hazard analysis. The team will conduct an internal design review, by going over all the outlined requirements, risks and hazards and verifiying that the design does not contain any logical flaws to the best of our abilities.

In addition, the team will use the feedback from reviewers such as classmates from our 4G06 Capstone class as well as instructor, supervisor and teaching assistants. This will help further assist our team to ensure that the design is free from any logical flaws and is able to help meet the outlined requirements.

3.4 Implementation Verification Plan

Throughout the development phase of this project, the team will use GitHub Issues and pull requests to implement various features. Each pull request will require at least two other team members to inspect and review the code ensuring that it meets the requirement and design discussed. A pipeline can also be implemented in GitHub to ensure that the build in the main branch is always stable. Unit tests will also be used to ensure that the implementation of the product is verified.

3.5 Automated Testing and Verification Tools

The final product will be an Universal Windows Platform (UWP) application built using Microsoft Visual Studio (VS) in C# and XAML.

Majority of the project's testing will be conducted within VS. For unit testing, the team will use the built-in features of Unit Test Applications in VS to create unit tests projects and units tests.

Code coverage tests is also covered within the suite of tools available within VS. The team will be able to create testing suites and unit tests for the VS project. The team also plans on using the results of the code coverage tests to identify which portion of the project's code not covered by our tests. Uncovered blocks of code can be colour-coded to signify to the developer that no current test covers that block of code. Other metrics such as number of lines covered, % of a block covered will be summarized per code coverage project to help indicate where more testing efforts are needed.

Using Visual Studio's IDE extensions, the team plans on using SonarLint and CSharpier as its linter and formatting tool for C# respectively.

3.6 Software Validation Plan

To verify that the software will work as intended and designed, the team will use the sample data provided by Dr. Zurob and see if the results are within a reasonable margin of error. This sample data is obtained from a collection of existing materials with known results. If the actual results are outside a reasonable range, then the team shall conclude that the results are not valid.

4 System Test Description

4.1 Tests for Functional Requirements

FR-T1. Control: Dynamic & Manual
Initial State: Entire system is set up and the application is running.
Input: Developers will initialize the current source with supply parameters, initialize the nanovoltmeter and capture measurements, and set up a sample and demonstrate the main function of the application.
Output: The current source shall be reset and the application shall input the parameters of the current supply and set them to the current source appropriately, then enabling and disabling the current. The current level shall be displayed appropriately. The nanovoltmeter shall be reset and the application shall display the voltage levels appropriately when capture is started. When stopping capture, the application shall pause the measurements and keep the last batch of data displayed.The application shall calculate and display the resistivity of the metallic

sample based on values obtained from the measurement devices.
Test Case Derivation: Since the measurement devices are connected via a high-speed communication bus (GPIB) to the control computer, calculations are expected to be synchronized to the measurements and displayed in real-time.
How test will be performed: Test will be performed by developers. By checking the "Current Source" radio button and specifying other parameters then clicking "Current ON" button and "Current OFF" button. Checking the "Nanovoltmeter" radio button and clicking "Start Capture" button and "Stop Capture" button. Then verify the resistivity calculations by taking samples from the logs and doing a manual calculation. Simultaneously monitoring both the values displayed on the current source and the ones displayed in the application while observing for latency.

FR-T2. Control: Dynamic & Manual
Initial State: Entire system is set up and the application is running.
Input: Developers will set up a sample and perform thermal treatment on it.
Output: The application shall monitor and take note of critical changes in resistivity.
Test Case Derivation: While the application is constantly monitoring the state of resistivity of samples in real-time, any major change in the resistivity indicates a transition in the phase of the material.
How test will be performed: Test will be performed by developers. Performing thermal treatment and observing changes in resistivity as calculated by the application.

FR-T3. Control: Dynamic & Manual
Initial State: Entire system is set up and the application is running.
Input: Developers will control the sampling rate of the voltage measurement.
Output: The nanovoltmeter shall display measurements at a rate faster or slower than the default rate, and the application shall follow suit when displaying measurements and calculations.
Test Case Derivation: The application is expected to be able to sample the data at variable rates, by changing the 'integration rate' of the nanovoltmeter through Powerline Cycles per calculation (PLC) and

since the nanovoltmeter is connected via a high-speed communication bus (GPIB) to the control computer, calculations are expected to be synchronized to the measurements and displayed in real-time.
How test will be performed: Test will be performed by developers. By choosing an integration rate from the drop-down menu and starting capture. Simultaneously monitoring both the values displayed on the measurement devices and the ones displayed in the application while observing for latency in the display and calculation.

FR-T4. Control: Dynamic & Manual
Initial State: Entire system is set up and the application is running.
Input: Developers will set up a sample and perform thermal treatment on it.
Output: The application shall automatically calculate the slopes of resistivity-temperature in the phase change diagram, identify changes in these slopes and attributing slope changes to phase transitions. This information is highlighted with appropriate phase transition labels.
Test Case Derivation: While the application is constantly monitoring the resistivity of samples in real-time, major changes in the resistivity to temperature ratios correlate to phase changes of the material.
How test will be performed: Test will be performed by developers. Performing thermal treatment, capturing measurements, and observing notifications and labels on graphs or logs made by the application.

FR-T5. Control: Dynamic & Manual
Initial State: Entire system is set up and the application is running.
Input: Developers will prompt a wireless connection to control computer and stop an experiment.
Output: The application is able to be controlled using a proxy/web application.
Test Case Derivation: The control computer is expected to be connected to the network so that when the system is set up the application is operable remotely.
How test will be performed: Test will be performed by developers communicating remotely with the control computer through a web app. The main application is already capturing measurements and making calculations. Remotely, the testing developer will click "Stop Experiment" button on the web app and that will prompt the main app to

stop capturing data and turn the current supply off.

FR-T6. Control: Dynamic & Manual
Initial State: Entire system is set up, connected to a sample, and the application is running.
Input: Developers will start an experiment.
Output: The application displays a graph that shows real-time values and calculations.
Test Case Derivation: Since the application is constantly monitoring and displaying resistivity of samples and other measurements in real-time, a graph will be able to follow these values and display them in a visual representation.
How test will be performed: Test will be performed by developers. By initializing the current source and voltmeter, then turning current supply on and starting capture. Then observing the graph in the application window.

FR-T7. Control: Dynamic & Manual
Initial State: Entire system is set up, connected to a sample, and the application is running.
Input: Developers will start an experiment then stop it.
Output: The application shall save a file on the control computer containing the measurements and calculations of the experiment.
Test Case Derivation: Since the application is constantly monitoring and displaying resistivity of samples and other measurements in real-time, a file system will be able to follow these values and write them to a data file outside of the application.
How test will be performed: Test will be performed by developers. By inputting information about the experiment and the personnel involved, then initializing the current source and voltmeter, and turning current supply on and starting capture. When enough data has been captured, they shall stop the experiment and check the generated file.

4.2 Tests for Nonfunctional Requirements

4.2.1 Appearance Test

NF-AT1. Type: Static & Manual
Initial State: Application is opened and ready for use.

Input/Condition: User will follow a simple set of instructions to explore application, followed by a survey
Output/Result: Survey result will indicate MIN_USER_ACCEPT_RATE of users agree or strongly agree application is uncomplicated.
How test will be performed: Test will be performed by the user. They will receive a survey with a question. If MIN_USER_ACCEPT_RATE of users indicate agree or strongly agree, then it will be considered successful. (Refer to Appendix for Sample Survey)

NF-AT2. Type: Static & Manual
Initial State: Application is opened and ready for use.
Input/Condition: Developers will navigate the application by exploring every screen and state.
Output/Result: If all sections of the applications is in English, then this test is considered a pass. Otherwise, it is considered a fail.
How test will be performed: Different sets of instructions are used to cover a subset of the application screens and states, each set of instruction is to be carried out by a developer to verify the result.

4.2.2 Usability Test

NF-UT1. Type: Dynamic & Manual
Initial State: The application is open to home screen and ready to use.
Input/Condition: The users with no prior experience with the product.
Output/Result: At least *MIN_USER_ACCEPT_RATE* of users with no prior experience can successfully complete each task without additional assistance
How test will be performed: The users will be observed while completing the task. The test will pass if users complete the task without additional assistance.

NF-UT2. Type: Dynamic & Manual
Initial State: The application is open to home screen and ready to use.
Input/Condition: User will be asked to modify the voltage and current parameters of the application to match a sample experi-

ment we provide.
Output/Result: The user takes no more than *INTERACT_TIME* to interact with the application and make the modifications to the voltage and current parameters matching the sample experiment provided.
How test will be performed: The users will be observed while completing the task. The test will pass if the user takes no more than *INTERACT_TIME*.

NF-UT3. Type: Dynamic & Manual
Initial State: The application is open to home screen and ready to use.
Input/Condition: User will use the application by following a predefined set of tasks.
Output/Result: The total number of mistakes made by the user when completing the tasks by interacting with application should be no more than *MAX_MISTAKE*.
How test will be performed: The observer will record the number of mistakes that they observe as the user completes the set of tasks.

NF-UT4. Type: Dynamic & Manual
Initial State: Application will be closed.
Input/Condition: Users will open the application and perform a set of tasks listed for a sample experiment. User then will be asked to return the next day to perform the same set of tasks.
Output/Result: The time that the user takes to complete the set of tasks for the second time will be less than the time taken for the first time.
How test will be performed: The users will be observed and timed while completing the set of tasks both times. The test will pass if the user completes the set of tasks in the same or less amount of time in the next day.

NF-UT5. Type: Static & Manual
Initial State: The application is open to home screen and ready to use.
Input/Condition: User will input sample parameters and application will perform calculations.
Output/Result: The application will display the result of the application and the user will not know how the calculations are per-

formed
How test will be performed: User will be observed while performing the task and asked to fill out an usuablity survey question. If the user is unaware of how the calculations were performed, the test will pass.

NF-UT6. Type: Static & Manual
Initial State: The application will be installed on the research device
Input/Condition: The size of the application will be checked manually
Output/Result: Application capacity on the device will be no more than *MAX_SIZE*.
How test will be performed: We will open the file properties of the application and verify that the size of the application is less than or equal to *MAX_CAPACITY*.

4.2.3 Performance Test

NF-PT1. Type: Dynamic & Automated
Initial State: Application is installed and running with measurement devices connected to a sample.
Input/Condition: Set *MIN_SAMPLE_RATE* and start thermal treatment of sample.
Output/Result: Calculate and output the application's sample rate and check to see if it is greater or equal to *MIN_SAMPLE_RATE*.
How test will be performed: Using the parameters of the devices, we will calculate the observed sample rate and compare it with the inputted sample rate by the user. The test will pass if the observed sample rate is *MIN_SAMPLE_RATE* or greater.

NF-PT2. Type: Dynamic & Manual
Initial State: Application is installed and running.
Input/Condition: User will be given parameters to change.
Output/Result: Application will reflect changes by within *TIME_ACCEPTED*.
How test will be performed: User will make modifications, we will observe how long the application takes to reflect changes.

NF-PT3. Type: Dynamic & Automated
Initial State: Application is installed and running with measure-

ment devices connected to a sample.
Input/Condition: Thermal treatment sample data will be read into the device.
Output/Result: The sample reading from the thermal treatment will be accurate to *ACCEPTED_SIGFIG* number of significant digits.
How test will be performed: An automated test will be used to compare the expected values to the measured values. The test will pass if the result is correct up to the number of ACCEPTED_SIGFIG significant digits.

NF-PT4. Type: Dynamic & Manual
Initial State: Application is installed and running with measurement devices connected to a sample.
Input/Condition: Application will be opened and user will interact with it.
Output/Result: The application will be up and running for at least *MIN_UPTIME* and for the entire duration of the task given to the user to perform.
How test will be performed: User will complete a set of tasks using the application and the application will be left running for at least *MIN_UPTIME* afterwards.

4.2.4 Operational Test

NF-OT1. Type: Static & Manual
Initial State: Application is opened and ready with keyboard and mouse connected to the device.
Input/Condition: User will interact with the application with mouse and keyboard.
Output/Result: Application will accurately and correctly reflect the inputs from the mouse and keyboard provided by the user.
How test will be performed: User will be asked to perform a simple set of instructions using the mouse and keyboard such as clicking and typing.

NF-OT2. Type: Static & Manual
Initial State: A machine/computer that does not have the application installed.

Input/Condition: Users will be asked to install the application on the machine/computer.
Output/Result: Survey results should indicate that the installation was easy to follow for MIN_USER_ACCEPT_RATE of the users.
How test will be performed: Test will be performed by the user. User will be given a survey including a question where 5 indicates easy to use, straightforward, and 1 indicating confusing, and/or difficult. If MIN_USER_ACCEPT_RATE of users respond with 5, the test will be considered a pass.

4.2.5 Maintainability/Support Test

NF-MT1. Type: Static & Manual
Initial State: A machine/computer that does not have the application installed.
Input/Condition: Application will be installed on the device and connected to the required measurement devices.
Output/Result: The application is able to operate smoothly without error with the connected measuring devices.
How test will be performed: Test will be performed by a developer installing the application onto a lab computer running Windows 10.

4.2.6 Security Test

NF-ST1. Type: Static & Manual
Initial State: Application is opened and ready to use.
Input/Condition: User will attempt to enter or modify data/parameters that they currently do not have permission for.
Output/Result: Application will display a message informing the user that they do not have permission.
How test will be performed: Test will be performed by users with missing permissions, or a test account by a developer.

NF-ST2. Type: Static & Manual
Initial State: Application is opened and ready to use.
Input/Condition: User will be asked to change protected/hidden settings given the appropriate permissions.

Output/Result: The changes are accepted and saved by the application. Settings are also accurately reflected on the application's interface.
How test will be performed: Test will be performed by users who have been granted permission to access these settings, or test account by a developer.

4.2.7 Cultural Test

NF-CT1. Type: Static & Manual
Initial State: Application is opened and ready to use.
Input/Condition: Users will explore the application through a demo/test account.
Output/Result: Survey result will indicate MIN_USER_ACCEPT_RATE of users agree or strongly agree application is appropriate.
How test will be performed: Test will be performed by the user. They will receive a survey with a question. If MIN_USER_ACCEPT_RATE of users indicate agree or strongly agree that application is appropriate, then it will be considered successful. (Refer to Appendix for Sample Survey)

4.2.8 Health and Safety Test

NF-HT1. Type: Static & Manual
Initial State: Application is opened and ready to use.
Input/Condition: Users will explore the application through a demo/test account.
Output/Result: Survey result will indicate MIN_USER_ACCEPT_RATE of users agree or strongly agree application does not cause/pose any noticeable health concerns.
How test will be performed: Test will be performed by the user. They will receive a survey with a question. If MIN_USER_ACCEPT_RATE of users indicate agree or strongly agree that application is free from health and safety concerns, then it will be considered successful. (Refer to Appendix for Sample Survey).

4.3 Traceability Between Test Cases and Requirements

Table 3: Requirements Traceability

Requirements	**Tests**
FR1	FR-T1
FR2	FR-T2
FR3	FR-T3
FR4	FR-T4
FR5	FR-T5
FR6	FR-T6
FR7	FR-T7
NFR-L1	NF-AT1
NFR-L2	NF-AT2
NFR-U1	NF-UT1
NFR-U2	NF-UT2
NFR-U3	NF-UT3
NFR-U4	NF-UT4
NFR-U5	NF-UT5
NFR-U6	NF-UT6
NFR-P1	NF-PT1
NFR-P2	NF-PT2
NFR-P3	NF-PT3
NFR-P4	NF-PT4
NFR-O1	NF-OT1
NFR-O2	NF-OT2
NFR-O3	N/A
NFR-M1	N/A
NFR-M2	NF-MT1
NFR-M3	NF-MT1
NFR-S1	NF-ST1

Requirements	Tests
NFR-S2	NF-ST2
NFR-C1	NF-CT1
NFR-H1	NF-HT1
NFR-H2	NF-HT1
NFR-I1	NF-MT1

5 Unit Test Description

[Reference your MIS and explain your overall philosophy for test case selection. —SS] [This section should not be filled in until after the MIS has been completed. —SS]

5.1 Unit Testing Scope

No individual module correspond to any non functional requirements so section for non functional unit tests are removed. [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

5.2.1 Input Communication Module

This module is a class used to initiate an object that contains properties and methods that will be used throughout the application. The following test will be used to verify if the module stores and returns the properties correctly using methods exposed.

UT-IC1. test-id1

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

Initial State:

Input: input.getHardwareInput() will be called.

Output: NULL

Test Case Derivation: When input.getHardwareInput() is called, NULL exception will be returned as properties has not been given.

How test will be performed: Object will be instantiated with input user properties and verified using Automated Unit Testing

UT-IC2. Type: Static, Automatic

Initial State: Object has been instantiated with hardware inputs and saved to input.

Input: input.getHardwareInput() will be called.

Output: HardwareInput ADT

Test Case Derivation: When input.getHardwareInput() is called, getHardwareInput ADT will be returned as properties has been given.

How test will be performed: Object will be instantiated with hardware input properties and verified using Automated Unit Testing

UT-IC3. Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input: input.getUserInput() will be called.

Output: NULL

Test Case Derivation: When input.getUserInput() is called, NULL exception will be returned as properties has not been given.

How test will be performed: Object will be instantiated with input user properties and verified using Automated Unit Testing

UT-IC4. Type: Static, Automatic

Initial State: Object has been instantiated with hardware inputs and saved to input.

Input: input.getHardwareInput() will be called.

Output: HardwareInput ADT

Test Case Derivation: When input.getHardwareInput() is called, getHardwareInput ADT will be returned as properties has been given.

How test will be performed: Object will be instantiated with hardware input properties and verified using Automated Unit Testing

UT-IC5. Type: Static, Automatic

Initial State: Object has been instantiated with hardware inputs and saved to input.

Input: input.getUserInput() will be called.

Output: NULL

Test Case Derivation: When input.getUserInput() is called, NULL exception will be returned as properties has not been given.

How test will be performed: Object will be instantiated with hardware input properties and verified using Automated Unit Testing

5.2.2 Remote Access Module

This module is used to allow remote access to the application so the user view the process of the experiment. The following methods will verify if the function of the remote access will work and if the user can connect and disconnect successfully.

UT-RA1. Type: Static

Initial State: Object has been instantiated and saved to remote.

Input: remote.connect(userName, password) will be called with valid userName and password.

Output: Will observe if it has been connected.

Test Case Derivation: When connect is called with valid userName and password, there will be a visual indicator showing connect is working.

How test will be performed: Object will be instantiated and result will be observed.

UT-RA2. Type: Static, Automatic

Initial State: Object has been instantiated and saved to remote.

Input: remote.connect(userName, password) will be called with valid userName and invalid password.

Output: INVALID

Test Case Derivation: When connect is called with valid userName and invalid password, an exception will be thrown because password is incorrect.

How test will be performed: Object will be instantiated and result will be verified by Automatic Unit Testing.

UT-RA3. Type: Static, Automatic

Initial State: Object has been instantiated and saved to remote.

Input: remote.connect(userName, password) will be called with invalid userName and valid password.

Output: INVALID

Test Case Derivation: When connect is called with invalid userName and valid password, an exception will be thrown because userName is incorrect.

How test will be performed: Object will be instantiated and result will be verified by Automatic Unit Testing.

UT-RA4. Type: Static, Automatic

Initial State: Object has been instantiated and saved to remote.

Input: remote.connect(userName, password) will be called with invalid userName and password.

Output: INVALID

Test Case Derivation: When connect is called with invalid userName and password, an exception will be thrown because both are incorrect.

How test will be performed: Object will be instantiated and result will be verified bu Automatic Unit Testing.

UT-RA5. Type: Static

Initial State: Object has been instantiated and saved to remote.

Input: remote.disconnect() will be called.

Output: Will observe if it has been disconnected.

Test Case Derivation: When disconnect is called, there will be a visual indicator showing disconnect finished.

How test will be performed: Object will be instantiated and result will be observed.

5.2.3 Current State Module

This module is a class used to display current state within the application for the user to understand. The following test will be used to verify if the module exposes the correct state for the user to see.

UT-CS1. Type: Static

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayUserInfo() will be called.

Output: Will observe if it has been displyed.

Test Case Derivation: When currentState.DisplayUserInfo() is called, there will be a change on the display that can verify if the state is correct.

How test will be performed: Object will be instantiated and result will be observed.

UT-CS2. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayUserInfo() will be called with invalid SamplingRate type.

Output: INVALID

Test Case Derivation: When currentState.DisplayUserInfo() is called, since SamplingRate is an incorrect type then a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS3. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayUserInfo() will be called with negative SamplingRate.

Output: INVALID

Test Case Derivation: When currentState.DisplayUserInfo() is called with a negative SamplingRate a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS4. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayUserInfo() will be called with invalid SampleLength type.

Output: INVALID

Test Case Derivation: When currentState.DisplayUserInfo() is called with invalid SampleLength type a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS5. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayUserInfo() will be called with negative SampleLength.

Output: INVALID

Test Case Derivation: When currentState.DisplayUserInfo() is called with negative SampleLength a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS6. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayUserInfo() will be called with invalid SampleWidth type.

Output: INVALID

Test Case Derivation: When currentState.DisplayUserInfo() is called with invalid SampleWidth type a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS7. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayUserInfo() will be called with negative SampleWidth.

Output: INVALID

Test Case Derivation: When currentState.DisplayUserInfo() is called with negative SampleWidth a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS8. Type: Static

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayHardwareState() will be called.

Output: Will observe if it has been displyed.

Test Case Derivation: When currentState.DisplayHardwareState() is called, there will be a change on the display that can verify if the state is correct.

How test will be performed: Object will be instantiated and result will be observed.

UT-CS9. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayHardwareState() will be called with invalid voltage type.

Output: INVALID

Test Case Derivation: When currentState.DisplayHardwareState() is called with invalid voltage type a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS10. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayHardwareState() will be called with negative voltage.

Output: INVALID

Test Case Derivation: When currentState.DisplayHardwareState() is called with negative voltage a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS11. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayHardwareState() will be called with invalid current type.

Output: INVALID

Test Case Derivation: When currentState.DisplayHardwareState() is called with invalid current type a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS12. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayHardwareState() will be called with negative current.

Output: INVALID

Test Case Derivation: When currentState.DisplayHardwareState() is called with negative current a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS13. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayHardwareState() will be called with invalid time type.

Output: INVALID

Test Case Derivation: When currentState.DisplayHardwareState() is called with invalid time type a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS14. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayHardwareState() will be called with negative time.

Output: INVALID

Test Case Derivation: When currentState.DisplayHardwareState() is called with negative time a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

UT-CS15. Type: Static, Automatic

Initial State: Object has been instantiated and saved to currentState, currentState.StateInit() will be called.

Input: currentState.DisplayHardwareState() will be called with invalid temperature type.

Output: INVALID

Test Case Derivation: When currentState.DisplayHardwareState() is called with invalid temperature type a invalid exception should be thrown.

How test will be performed: Object will be instantiated and verified using Automated Unit Testing.

5.2.4 Calculation Module

UT-C1. Type: Automatic, Static

Initial State: State varaible Resistance has a value of zero

Input: The method getResistance() is called and assigned to the variable Resistance

Output: Resistance has a value greater than zero and matches the calculated value.

Test Case Derivation: If getResistance() calculates and retrieves the resistance value correctly, then the value of resistance should be non zero and match the calculated value.

How test will be performed: The Calculation Module will be imported and the method getResistance() will be called. The value of resistance will be tested through automated unit testing.

UT-C2. Type: Automatic, Static

Initial State: State varaible Resistance has a value of zero

Input: The method getResistance() is called and results in a value less than zero.

Output: Method will throw an INVALID error

Test Case Derivation: If getResistance() calculates and retrieves the resistance value less than zero, then an error should be thrown.

How test will be performed: The Calculation Module will be imported and the method getResistance() will be called. The value of resistance will be tested through automated unit testing.

UT-C3. Type: Automatic, Static

Initial State: State varaible Resistivity has a value of zero

Input: The method getResistivity() is called and assigned to the variable Resistivity

Output: Resistivity has a non-zero value and matches the calculated value.

Test Case Derivation: If getResistivity() calculates and retrieves the resistivity value correctly, then the value of resistivity should be non zero and match the calculated value.

How test will be performed: The Calculation Module will be imported and the method getResistivity() will be called. The value of resistivity will be tested through automated unit testing.

UT-C4. Type: Automatic, Static

Initial State: State varaible Resistivity has a value of zero

Input: The method getResistivity() is called and produces a non-real number/value.

Output: Method will throw an INVALID error

Test Case Derivation: If getResistivity() calculates and retrieves the resistivity value is not a number, then an error should be thrown.

How test will be performed: The Calculation Module will be imported and the method getResistivity() will be called. The value of resistivity will be tested through automated unit testing.

5.2.5 User Input Validation Module

UT-UI1. Type: Automatic, Static

Initial State: The module UserInputValidation is imported

Input: The method getUserInput() is called.

Output: The abstract data type UserInput is returned

Test Case Derivation: If getUserInput() is called, UserInput should be returned.

How test will be performed: The User Input Validation Module will be imported and the method getUserInput() will be called through automated unit testing.

UT-UI2. Type: Automatic, Static

Initial State: The module UserInputValidation is imported

Input: The method validateFileData() is called with correct values for FileName, Date, Name.

Output: TRUE

Test Case Derivation: If FileName, Date, Name are valid, then it should return TRUE.

How test will be performed: The User Input Validation Module will be imported and the method validateFileData() will be called through automated unit testing.

UT-UI3. Type: Automatic, Static

Initial State: The module UserInputValidation is imported

Input: The method validateFileData() is called with incorrect values for FileName, Date, or Name.

Output: FALSE

Test Case Derivation: If FileName, Date, or Name are not valid, then it should return FALSE.

How test will be performed: The User Input Validation Module will be imported and the method validateFileData() will be called through automated unit testing.

UT-UI4. Type: Automatic, Static

Initial State: The module UserInputValidation is imported

Input: The method validateSampleData() is called with correct values for SamplingRate, SampleLength, SampleWidth.

Output: TRUE

Test Case Derivation: If SamplingRate, SampleLength, SampleWidth are valid, then it should return TRUE.

How test will be performed: The User Input Validation Module will be imported and the method validateSampleData() will be called through automated unit testing.

UT-UI5. Type: Automatic, Static

Initial State: The module UserInputValidation is imported

Input: The method validateSampleData() is called with incorrect values for SamplingRate, SampleLength, or SampleWidth.

Output: FALSE

Test Case Derivation: If SamplingRate, SampleLength, or SampleWidth are not valid, then it should return FALSE.

How test will be performed: The User Input Validation Module will be imported and the method validateSampleData() will be called through automated unit testing.

5.2.6 Hardware Input Validation Module

UT-HI1. Type: Automatic, Static

Initial State: The module HardwareInputValidation is imported

Input: The method getHardwareInput() is called.

Output: The abstract data type HardwareInput is returned

Test Case Derivation: If the method getHardwareInput() is called, then it should return the abstract data type HardwareInput.

How test will be performed: The Hardware Input Validation Module will be imported and the method getHardwareInput() will be called through automated unit testing.

UT-HI2. Type: Automatic, Static

Initial State: The module HardwareInputValidation is imported

Input: The method validateParameters() is called with correct values for Voltage, Time and Current.

Output: TRUE

Test Case Derivation: If the values for Voltage, Time and Current are correct, then it should return TRUE.

How test will be performed: The Hardware Input Validation Module will be imported and the method getHardwareInput() will be called through automated unit testing.

UT-HI3. Type: Automatic, Static

Initial State: The module HardwareInputValidation is imported

Input: The method validateParameters() is called with incorrect values for Voltage, Time or Current.

Output: FALSE

Test Case Derivation: If the values for Voltage, Time and Current are correct, then it should return FALSE.

How test will be performed: The Hardware Input Validation Module will be imported and the method getHardwareInput() will be called through automated unit testing.

5.3 Traceability Between Test Cases and Modules

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

References

Author Author. System requirements specification. https://github.com/..., 2019.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

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

6.2 Survey Questions

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1.

2.