Project Title: System Verification and Validation Plan for Mechatronics Engineering

Team #1, Back End Developers
Jessica Bae
Oliver Foote
Jonathan Hai
Anish Rangarajan
Nish Shah
Labeeb Zaker

November 2, 2022

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

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

[Remove this section if it isn't needed —SS]

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

3 General Information

3.1 Summary

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

3.2 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]

3.3 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. —SS]

Author (2019)

4 Plan

Verification and Validation plan will be done on both the hardware and software side individually as well as the system integration. The Researcher/Stakeholders and Back End developers will have different roles and techniques to test the device's functionality and usability as described below.

4.1 Verification and Validation Team

The following table defines the roles for the stakeholder and Back End Developers team for Verification and Validation.

Name	Role	Description
Dr Luciano Macedo	End user	Stakeholder who will be the end user, verifying all the user requirements and functionality in every sprints.
Johnathan Hai	Subsystem tester	There will be a set of test cases pre-defined for every functional requirement. Jonathan will be testing the specific functions against the test cases using unit testing, integration testing and boundary condition tests as well as SRS verification.
Jessica Bae	Black Box tester	Testing the specific functions without knowing how the internal workflow is structured. This will include stress testing for I/O for all subsytems and boundary condition testing. Valgrind will also be used for code profiling and debugging memory leaks.
Labeeb Zaker	Remote codebase manager	Ensuring the codebase on GitHub has no flaws, maintaining CI/CD, reviewing every commit added to ensure it meets the requirements without any possible case of errors.
Nish Shah	Automated tester	Maintaining the automated testing scripts to run certain functions against test cases that are repetitive to keep checking code is functional. Primarily using Python code coverage tools and unit testing frameworks for code development (unittest, Coverage.py).
Anish Rangarajan	Hardware & Automated tester	Verification and testing of embedded development/hardware in C using Cunit and bullseye coverage for unit testing framework and code coverage.
Oliver Foote	White Box & Database testing	Testing the internal workflow such that for a given set of inputs, an expected output is returned. Database validation using Orion for stress test of I/O and database functionality.

Table 1: Team roles for Verification and Validation.

4.2 SRS Verification Plan

The SRS verification plan will follow a checklist for reviewers based on an ad hoc approach. It will also follow a rigorous task based inspection listed in the table below. This is to make sure all the requirements and goals receive several iterations of verification and validation.

Section of SRS	**Description**	**Approach of Feedback**
General feedback and discussion	The verification will be done based on an ad hoc approach through the checklist below: • Verify System constrains for hardware and software. • Monitored and controlled variables are consistent. • Comparison to existing solutions is unambiguous. • Project Goals are relevant to description of device. • Requirements (functional & non-functional) are prioritized and verified. • Event handling and FSM is verified and correct.	Will be done by Stakeholder Dr Luciana Macedo, supervisor, classmates and Back-end Developers.
General System Description 4.1-4.3	The verification for this section would be checking if System constrains and user characteristics are provided in detail without any ambiguity.	Will be done by team member Jonathan Hai.
Section 5 Definitions and variables	Verifying if terminology and the use of monitored and controlled variables is consistent and correct. Assumptions are clearly mentioned and provided along with goals of the project.	Will be done by team member Nish Shah.
Required behavior and Requirements	Validation of Functional and Non-functional requirements and to verify if the requirements are classified correctly. Checking required behavior as well.	Will be done by team member Anish Rangarajan.
Section 8-12 Traceability matrices and normal operation	Verification of matrices and graphs according to Requirements and also verifying if normal operation covers the required goals of the device.	Will be done by team member Labeeb Zaker.
Section 14-16	Verification of FSM, Legal factors and Phase in Plan.	Will be done by team member Jessica bae.

Table 2: SRS verification checklist and task based inspection

4.3 Design Verification Plan

[Plans for design verification —SS]
[The review will include reviews by your classmates —SS]
[Create a checklists? —SS]

4.4 Verification and Validation Plan Verification Plan

[The verification and validation plan is an artifact that should also be verified. —SS]
[The review will include reviews by your classmates —SS]
[Create a checklists? —SS]

4.5 Implementation Verification Plan

The Implementation Verification Plan will involve static and dynamic code analysis as the debugging methods to find bugs early in development. The checklist below can be used in development to follow the Implementation verification plan.
The following methods will be used for static code analysis:

- Bi-weekly code review with sub-teams (hardware, software, frontend/backend) for code walk through.
- Weekly code review with entire team for system-level software design.
- Use a SAST tool to scan source code, binary and byte code to reveal vulnerabilities.
- Use static analysis tools such as Pylint, lint described in 4.6 to find bugs in code.

Dynamic code testing will help identify exploitable vulnerabilities. The following methods will be used for dynamic code analysis:

- Use a DAST tool as a black-box tester which inputs malicious SQL queries, Long input strings and invalid data to exploit assumptions made by developers.
- Running modular code separately against variety of inputs to find bugs in code.
- Use open source tools based on the programming language (CrossHair for Python, CHAP for C) to provide a comprehensive view of performance and security of the device.

4.6 Automated Testing and Verification Tools

Automated testing and verification tools will be based on the different programming languages and tools used in the development of the device. As covered in the development plan, (reference dev plan¿¿¿) the following unit testing frameworks, profiling tools and linters will be used for verification of code:

- Python:

  - unittest will be used as the Unit testing framework for Python to cover test automation, aggregation of tests (integration testing) and independence of tests from the reporting framework.
  - Coverage.py will be used to measure code coverage and to gauge the effectiveness of tests performed.
  - Pylint will be used as the linter/static code analyser which will check for errors and enforce a coding standard.

- C:
  - CUnit will be used as a unit testing framework for embedded system development.
  - Bullseye coverage will be used for C code coverage analysis.
  - lint will be used as the linter/static code analyser for development in C.

- Valgrind will be used for Python (pytest-valgrind) and C as a debugging tool for code profiling.

- SQL/Database testing: Orion will be used to stress test an I/O coverage and to make sure the database functions as expected.

4.7 Software Validation Plan

There will be no plan for Software validation since there is no external data used and all the software-related validation is covered in section 4.2.

5 System Test Description

5.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]
[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

5.1.1 Area of Testing1

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

5.1.2 Area of Testing2

...

5.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

5.2.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

Input/Condition:

Output/Result:

How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.2.2 Area of Testing2

...

5.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

6 Unit Test Description

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

6.1 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]

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. ...

6.2.2 Module 2

...

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 ?

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:

6.3.2 Module ?

...

6.4 Traceability Between Test Cases and Modules

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

References

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

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

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

7.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

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:

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. 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 etc. You should look to identify at least one item for each team member.

2. 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?