Project Title: System Verification and Validation Plan for Radio Signal Strength Calculator

Xingzhi Liu

October 31, 2020

1 Revision History

Date	Version	Notes
Oct 30, 2020	1.0	First Draft of VnV Plan

Contents

List of Tables

List of Figures

You can remove this section title if you don't have any figures.

2 Symbols, Abbreviations and Acronyms

symbol	**description**
FR	Functional Requirement
MG	Module Guide
MIS	Module Interface Specification
NFR	Non-Functional Requirement
RSSC	Sadio SignalStrength Calculator
SRS	Software Requirement Specification
T	Test
VV	Validation and Verification

This document records and presents the verification and validation plan for RSSC to help ensure the program meets the requirements. General information including a quick recall of RSSC's background is given in section 3. section 4 provides a plan for verification and section 5 describes the system tests, including tests for functional requirements and tests for non- functional requirements.

3 General Information

3.1 Summary

The software to test is Radio Signal Strength Calculator (RSSC). The purpose of RSSC is to simulate the radio signal propagation in the indoor environment defined by the user, and find the analytical signal strength for the user.

3.2 Objectives

The objective of this document is to build confidence in the software correctness. To reach this objective, all functional and non-functional requirements will be tested following the descriptions in this document.

3.3 Relevant Documentation

- SRS

You can list your other documents as well, since they will all be done when the project is done.

4 Plan

4.1 Verification and Validation Team

- Author: Xingzhi Liu
- Primary Reviewer: Siddharth (Sid) Shinde
- Reviewer: Leila Mousapour
- Reviewer: Shayan Mousavi Masouleh
- Dr. Spencer Smith

You can say specifically what docs these reviewers will review

4.2 SRS Verification Plan

SRS will be done by team members reviewing the document. Team members can put any comments, suggestions or questions in RSSC's Github repository as issues. The author will respond to the issues and make modifications when needed.

Can you give any further structure to the review exercise?

4.3 Design Verification Plan

Design verification will be done by team members, by reviewing whether the steps of calculation in the software follows the physical model in SRS or not.

4.4 Implementation Verification Plan

Implementation verification will be done by testing all the functional and non- functional requirements. Descriptions of the tests can be found in subsection 5.1 and subsection 5.2. In addition, we will undergo static verification by checking all the codes we build with Pylint. We will also conduct unit testing for modules within the testing scope. Details for unit testing can be found in section 6.

Any thoughts for code inspection?

4.5 Automated Testing and Verification Tools

- Python Unittest
- Pylint

[The details of this section will likely evolve as you get closer to the implementation. —SS]

4.6 Software Validation Plan

There are no plans for validation. We may be able to set up a simulation of radio frequency signal on existing physics simulation tools and take the simulation result as a reference to evaluate RSSC's output, but different tools makes different assumptions to the indoor signal propagation model and we cannot change their assumptions. It is highly unlikely that we could find a tool that makes the same assumptions as RSSC. Therefore we do not have any reference to evaluate RSSC's correctness.

Validation would mean comparing to experimental data.

5 System Test Description

5.1 Tests for Functional Requirements

Functional requirements for RSSC are given in SRS section 5.1. There are 5 functional requirements for RSSC, from R1 to R5. R1 and R2 are corresponding to inputs, while R3 to R5 are corresponding to outputs. subsubsection 5.1.1 describes the input tests for R1 and R2; and subsubsection 5.1.2 describes the output tests for R3, R4 and R5.

5.1.1 Area of Testing1 - Input

This test verifies the following requirements:
R1: RSSC takes input from the user;
R2: RSSC verifies user inputs.

Input Tests

1. Valid inputs

   Control: Manual

   Initial State: Pending Input

   Input: $Pos_{tsm} = [0, 0]$;
   $[Pos_{sp}] = [[10, 0], [-10, 0]]$;
   $[C] = [[1, 1], [2, 2], [1, 3]]$;
   $[D] = [[2, 2], [1, 3], [1, 1]]$;
   $[T] = [0.1, 0.1, 0.1]$;
   $[R] = [0.6, 0.6, 0.6]$;
   $P^{dBm}_{tsm} = 0$;
   $f = 2.48 \times 10^9$;

   Output: Return an input success message in command line.

   Test Case Derivation: RSSC correctly takes the inputs from the user

   How test will be performed: Tester manually feeds the inputs into RSSC and executes RSSC.

2. Inconsistent input array sizes

   Control: Manual

   Initial State: Pending Input

   Input: $Pos_{tsm} = [0, 0]$;
   $[Pos_{sp}] = [[10, 0], [-10, 0]]$;
   $[C] = [[1, 1], [2, 2], [1, 3]]$;
   $[D] = [[2, 2], [1, 3]]$;
   $[T] = [0.1, 0.1, 0.1]$;
   $[R] = [0.6, 0.6, 0.6]$;
   $P_{tsm}^{dBm} = 0$;
   $f = 2.48 \times 10^9$;

   Output: Return error of inconsistent array size in command line.

   Test Case Derivation: Correct error message displays.

   How test will be performed: Tester manually feeds the inputs into RSSC and executes RSSC.

You can automate this test — unittest will expect an exception

Same comment applies elsewhere.

3. Out-of-range position coordinates

   Control: Manual

   Initial State: Pending Input

   Input: $Pos_{tsm} = [21, 0]$;
   $[Pos_{sp}] = [[10, 0], [-10, 0]]$;
   $[C] = [[1, 1], [2, 2], [1, 3]]$;
   $[D] = [[2, 2], [1, 3], [1, 1]]$;
   $[T] = [0.1, 0.1, 0.1]$;
   $[R] = [0.6, 0.6, 0.6]$;
   $P_{tsm}^{dBm} = 0$;
   $f = 2.48 \times 10^9$;

   Output: Return error of out-of-range position coordinates.

   Test Case Derivation: Correct error message displays.

   How test will be performed: Tester manually feeds the inputs into RSSC and executes RSSC.

4. Out-of-range transmitter power level

   Control: Manual

   Initial State: Pending Input

   Input: $Pos_{tsm} = [0, 0]$;
   $[Pos_{sp}] = [[10, 0], [-10, 0]]$;
   $[C] = [[1, 1], [2, 2], [1, 3]]$;
   $[D] = [[2, 2], [1, 3], [1, 1]]$;
   $[T] = [0.1, 0.1, 0.1]$;
   $[R] = [0.6, 0.6, 0.6]$;
   $P_{tsm}^{dBm} = 20$;
   $f = 2.48 \times 10^9$;

   Output: Return error of out-of-range transmitter power level.

   Test Case Derivation: Correct error message displays.

   How test will be performed: Tester manually feeds the inputs into RSSC and executes RSSC.

5. Out-of-range signal frequency

   Control: Manual

   Initial State: Pending Input

   Input: $Pos_{tsm} = [0, 0]$;
   $[Pos_{sp}] = [[10, 0], [-10, 0]]$;
   $[C] = [[1, 1], [2, 2], [1, 3]]$;
   $[D] = [[2, 2], [1, 3], [1, 1]]$;
   $[T] = [0.1, 0.1, 0.1]$;
   $[R] = [0.6, 0.6, 0.6]$;
   $P_{tsm}^{dBm} = 20$;
   $f = 2.48 \times 10^{12}$;

   Output: Return error of out-of-range signal frequency.

   Test Case Derivation: Correct error message displays.

How test will be performed: Tester manually feeds the inputs into RSSC and executes RSSC.

[TODO - learn how to write automatic testing scripts to generate random valid input sets and feed into RSSC. —XZ]

5.1.2 Area of Testing2 - Output

This test verifies the following requirements:
R3: RSSC shall find P_{sp}^{dBm};
R4: RSSC shall verify P_{sp}^{dBm};
R5: RSSC shall generate a file to store P_{sp}^{dBm};

Output Test

1. Simple single valid output

   Control: Manual

   Initial State: Pending Input

   Input: $Pos_{tsm} = [0, 0]$;
   $[Pos_{sp}] = [[1, 0]]$;
   $[C] = []$;
   $[D] = []$;
   $[T] = []$;
   $[R] = []$;
   $P_{tsm}^{dBm} = 0$;
   $f = 2.48 \times 10^9$;

   Output: A .txt or .csv file with 1 line, showing the sampling point position (1,0) and P_{sp}^{dBm} that satisfies $P_{sp}^{dBm} \leq 0$.

   What is the correct output?

   Test Case Derivation: RSSC returns the correct output.

   How test will be performed: Tester manually feeds the inputs into RSSC and executes RSSC.

   automate!

2. Simple multiple valid output

same comments elsewhere

Control: Manual

Initial State: Pending Input

Input: $Pos_{tsm} = [0, 0]$;
$[Pos_{sp}] = [[1, 0], [1, 1]]$;
$[C] = []$;
$[D] = []$;
$[T] = []$;
$[R] = []$;
$P_{tsm}^{dBm} = 0$;
$f = 2.48 \times 10^9$;

Output: A .txt or .csv file with 2 lines. Line 1 shows the sampling point position (1,0) and P_{sp}^{dBm} that satisfies $P_{sp}^{dBm} \leq 0$. Line 2 shows the sampling point position (1,1) and P_{sp}^{dBm} that satisfies $P_{sp}^{dBm} \leq 0$.

Test Case Derivation: RSSC returns the correct output.

How test will be performed: Tester manually feeds the inputs into RSSC and executes RSSC. [TODO - learn how to write automatic testing scripts to generate random valid input sets and feed into RSSC. —XZ]

5.2 Tests for Non-Functional Requirements

Non-functional requirements are given in SRS section 5.2. There are 5 non-functional requirements: Portable, Maintainable, and Understandable. The rest of this section provides detailed descriptions on how to test them.

5.2.1 Portable

Portability Test RSSC shall be able to run on different OS. We will test execute RSSC on both Windows and Linux Ubuntu.

1. Portability on Windows 10

   Type: Manual

   Initial State: Pending Input

Input: $Pos_{tsm} = [0, 0]$;
$[Pos_{sp}] = [[1, 0]]$;
$[C] = []$;
$[D] = []$;
$[T] = []$;
$[R] = []$;
$P_{tsm}^{dBm} = 0$;
$f = 2.48 \times 10^9$;

Output: A .txt or .csv file with 1 line, showing the sampling point position (1,0) and P_{sp}^{dBm} that satisfies $P_{sp}^{dBm} \leq 0$.

How test will be performed: Tester deploys RSSC on a machine with Windows 10 OS and execute with the input set above. On success, RSSC should provide the output as described.

why not run your unit tests on the different operating systems?

2. Portability on Linux Ubuntu

   Type: Manual

   Initial State: Pending Input

   Input: $Pos_{tsm} = [0, 0]$;
   $[Pos_{sp}] = [[1, 0]]$;
   $[C] = []$;
   $[D] = []$;
   $[T] = []$;
   $[R] = []$;
   $P_{tsm}^{dBm} = 0$;
   $f = 2.48 \times 10^9$;

   Output: A .txt or .csv file with 1 line, showing the sampling point position (1,0) and P_{sp}^{dBm} that satisfies $P_{sp}^{dBm} \leq 0$.

   ✓ How test will be performed: Tester deploys RSSC on a virtual machine with Ubuntu 20.04 and execute with the input set above. On success, RSSC should provide the output as described.

5.2.2 Maintainable

Maintainability Test Proper documents should be included in this project.

1. Maintainability Test

   Type: Manual

   Initial State: none

   Input: none

   Output: none

   (This is good but completeness alone doesn't determine maintainability. You could look at the traceability. Does every req map to a module? (for instance)

   How test will be performed: Tester manually checks the contents in the Github repo. On success, documents shall be uploaded following the schedule of CAS741 and no issue shall be closed without a proper response.

5.2.3 Understandable

Understandability Test Programs of RSSC should be organized, well commented, and easy to understand.

1. Understandability Test

   Type: Manual

   Initial State: none

   Input: none

   Output: none

   (It would be great if the Tester is someone other than you.

   How test will be performed: Tester review the code and complete the survey shown in Table 1.

Item	Score
Variable names are rational and follow consistent conventions	{0 - 5}
Functions are well commented on what they do	{0 - 5}
Functions tasks are broken down / No super complicated functions	{0 - 5}
No repeated chunks in the code	{0 - 5}
Code is well organized and follows the order of instance models	{0 - 5}

Table 1: Understandability Survey

5.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	R4	R5	Portable	Maintainable	Understandable
5.1.1	X	X						
5.1.2			X	X	X			
5.2.1						X		
5.2.2							X	
5.2.3								X

Table 2: Traceability Between Test Cases and Requirements

6 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]

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]

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]