Project Title: System Verification and Validation Plan for Sun Catcher

Sharon (Yu-Shiuan) Wu

December 21, 2019

1 Revision History

Date	Version	Notes
2019/11/04 1	1.0	First version of System VnV
Date 2	1.1	-

Contents

List of Tables

List of Figures

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test
SC	Sun Catcher
Φ_P	the latitude of the solar panel
I_S	the intensity of the sun measured by the satellites
P_{A_h}	the height of the solar panel
P_{A_w}	the width of the solar panel
$\theta_{S_{\text{date}}}$	zenith angle of sun in the date of a sequence of day
$year_{\text{Start}}$	the year of the calcuation's starting date
$month_{\text{Start}}$	the month of the calcuation's starting date
day_{Start}	the day of the calcuation's starting date
$year_{\text{End}}$	the year of the calcuation's ending date
$month_{\text{End}}$	the month of the calcuation's ending date
day_{End}	the day of the calcuation's ending date

2.1 Type Definition

This section defines the general data type for Sun Catcher.

day = a tuple of $(year, day, month)$

$date$ = a sequence of day

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

[It would have been better to not include these comments, since it will become confusing as I add new comments. Using the diff feature on GitHub will help you to look at only the new comments. —SS]

[For the table above, I suggest removing the word "the" from the start of most of the descriptions. You should also consistently capitalize the first word of each description. —SS]

This document provides an outline of the system verification and validation for the Sun Catcher. The general introduction section provides readers a summary of the functions in Sun Catcher and the related documents as the resources for testing. The plan section provides readers the plan for verifying and validating SC's Software Requirements Specification (SRS) and introduces the method, tools, and external data to implement the testing. The system test description provides the readers with the test cases related to functional and nonfunctional requirements in Sun Catcher. The test builds for uncovering the errors and boosting the confidence of the software while ensuring an acceptable performance is provided.

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

3 General Information

This section descrips the purpose of this document. The section Summary describes the general purpose of the software: Sun Catcher. The detail of the goal of Sun Catchercan be found inside the SRS document. The section Objectives descrides the object of System VnV plan. It reflects to the requirements of Sun Catcher. The detail of the requirements can be found inside the SRS document.

[There should usually be text between two section headings. In this case a "roadmap" of this section would be helpful. —SS] [Get it. —Author]

3.1 Summary

The subsections below are the test cases of Sun Catcher. Sun Catcher is the software that calculates the optimum tilt angle of the days duration that decided by users. Then it takes the calculated angle to estimate the optimal solar energy output.

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

3.2 Objectives

The goal of the test is to build confidence in the software's correctness and strengthen the robustness of the software by testing its software require-

ments. The functional and nonfunctional requirements of Sun Catcher, and its related equations and its values' constraint are found in the SRS [5]. The Verification and Validation Plan follows the requirements described in the SRS [5].

The object of the Verification and Validation Plan is:

To build the robustness of the system input.

To build the confidence of the correctness of the system output.

[You don't really want to copy and paste the SRS requirements. This will cause maintenance problems. Referencing the SRS here should be enough. You can also use cross-references between documents to reference specific parts of your SRS. —SS][Deleted —Author]

[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

The document that related to the test objectives.

- Software Requirements Specification for Sun Catcher[Wu [5]]

[Reference relevant documentation. This will definitely include your SRS —SS]

4 Plan

4.1 Verification and Validation Team

The test team includes the following members:
Main reviewer: Sharon (Yu-Shiuan) Wu
Secondary reviewer: Deema Alomair, Bo Cao, Sasha Soraine, Zhi Zhang, and Doctor Smith
[Probably just you. :-) —SS]

4.2 SRS Verification Plan

- Get feedback from the reviewers: Sasha Soraine, Zhi Zhang, and Doctor Smith, after SRS is completed and put to the GitHub.
- Check the document by using SRS-Checklist and Writing-Checklist before publishing to GitHub.

[List any approaches you intend to use for SRS verification. This may just be ad hoc feedback from reviewers, like your classmates, or you may have something more rigorous/systematic in mind.. —SS]

4.3 Design Verification Plan

- The design should be verified by complete and success the test cases in the system VnV plan under section 5.
- The design should be verified by complete and success the test cases in the UnitVnVPlan.
- The design should satisfy all the functional and nonfunctional requirements stated in the SRS document [5].

[How are you going to verify the design? —SS]
[Plans for design verification —SS]

4.4 Implementation Verification Plan

The following tools will be used to facilitate testing:

- Mapping data with the external's data: Using the data provides in the external documents [Landau] [4] and [MarkandVijaysinh] [3] to verify the result values. It is used in the test section 5.1.2
- Rubber Duck Debugging: Performed by author, Sharon (Yu-Shiuan) Wu. The author should verbally explain the code line by line.
- Haskell Program Coverage: Dynamic Testing Tool, a tool-kit to record and display the code coverage of a Haskell Program. It aims to reinforce the correctness of the software and to eliminate the infeasibility problems.[Gill and Runciman] [2]

- QuickCheck: Automatic testing tool for Haskell programs, a library for random testing of program properties. It aims to boost the robustness of the software.[Claessen] [1]

- Tasty: a testing framework for Haskell, using it to build automatic test cases for 5.1.1

[You should at least point to the tests listed in this document and the unit testing plan. —SS] [In this section you would also give any details of any plans for static verification of the implementation. Potential techniques including code walkthroughs, code inspection, static analyzers, etc. —SS]

4.5 Software Validation Plan

Sun Catcher should be valid by satisfied all the functional requirement in SRS plan.

Based on the physical concept of Sun Catcher, the author, Yu-Shiauan Wu, should record the actual solar energy by using the output from Sun Catcher. Then verify whether the calculated tilt angle can increase the energy gaining.

[Validation means a comparison to actual experiments. I don't think this is what you intend to do. —SS]

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

5 System Test Description

5.1 Tests for Functional Requirements

The subsection below is designed to cover the functional requirements of Sun Catcher, which also describes in section 3.2.
The test is divided into four subsections, which are input reading, input bounds, output calculation, and output verification. Input reading testing is designed for testing the ability to receive information from the software interface. Input bounds testing and output calculation testing are designed for testing the robustness of the software. Output verification testing is designed for the correctness of the implemented equation.

[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. —SS]

5.1.1 Input Reading

This test covers the requirements of reading the inputs. Sun Catcherhas to identity users' inputs and then assign the values to designated equations or modules.

[Space are frequently missing between words. Please check for this during your editing step. —SS]

[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. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Identify users' input

1. InputReading-id1

   Control: Automatic

   Initial State: No input value

   Input: (P_{A_h}, P_{A_w})

   Output: The expected result will for the given inputs is based on the input value. The output should display the input value on the screen.

id	Input	Output
id1.1	(1455, 665)	(1455, 665)
id1.2	(1455.54, 665.13)	(1455.54, 665.13)

   [The expected result for the given inputs —SS]

   Test Case Derivation: The test case is testing the ability of capture the input value. Then output the designed variables for P_{A_h} and P_{A_w}. Calculate an **absolute error**, such that | expected output - actaul output |. [Justify the expected value given in the Output field —SS]

How the test will be performed:

- Input the designed test case input values by using testing framework, Tasty.
- Display the output of the assigned variables on the screen.
- Display the absolute error on the screen.
- If all the absolute error = 0 then the test case pass. Otherwise the test case fail.

[This test case should be automated, not manual. You do not need to enter the values from the keyboard. The values can be entered into code and the appropriate function(s) called. This test is really a test that your getters and setters are working. —SS] [Yes, plan to use testing framework Tasty to implement automatic testing. —Author]

2. InputReading-id2

Control: Automatic.

Initial State: No input value
Input: Φ_P [I've fixed some spelling mistakes and spacing mistakes. Please make an effort for more attention to detail. —SS] [Yes, I will do my proof read. —Author]

Output: The expected result will for the given inputs is based on the input value. The output should display the input value on the screen.

id	**Input**	**Output**
id2.1	90	90
id2.2	-90	-90
id2.3	3.2	3.2
id2.4	-3.2	-3.2
id2.5	0	0

[The expected result for the given inputs —SS]

The test case is testing the ability of capture the input value. Then output the designed variables for Φ_P. Calculate an **absolute error**, such that | expected output - actaul output |.

[The test case for invalid input should be a separate test case. In the invalid input case an exception should be raised. You can write a unit test that verifies that the correct exception is raised, when it is expected to be raised. —SS] [OK, move the input boundary to unit testing. —Author]

[Justify the expected value given in the Output field —SS]

How the test will be performed:

- Input the designed test case input values by using testing framework, Tasty.
- Display the output of the assigned variables on the screen.
- Display the absolute error on the screen.
- If all the absolute error = 0 then the test case pass. Otherwise the test case fail.

3. InputReading-id3
   Control: Automatic.

   Initial State: No any given value.
   Input: ($year_{\text{Start}}$, day_{Start}, $month_{\text{Start}}$) ($year_{\text{End}}$, day_{End}, $month_{\text{End}}$)

   Output: The expected result will for the given inputs is based on the input value. The output should display the input value on the screen.

id	**Input**	**Output**
id3.1	(2020, 28, 02) - (2021, 28, 02)	(2020, 28, 02) - (2021, 28, 02)
id3.2	(1996, 01, 03) - (2000, 14, 01)	(1996, 01, 03) - (2000, 14, 01)

   [The expected result for the given inputs —SS]

   Test Case Derivation: The test case is testing the ability of capture the input value. Then output the designed variables for ($year_{\text{Start}}$,

$month_{\text{Start}}$, day_{Start}) and ($year_{\text{End}}$, $month_{\text{End}}$, day_{End}). Calculate an **absolute error**, such that | expected output - actaul output |.

[Justify the expected value given in the Output field —SS]

How the test will be performed:

- Input the designed test case input values by using testing framework, Tasty.
- Display the output of the assigned variables on the screen.
- Display the absolute error on the screen.
- If all the absolute error = 0 then the test case pass. Otherwise the test case fail.

5.1.2 Output Verification

[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. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

This test covers the requirements of the verification of the outputs. This test uses external documents, which mentioned in section 4.4, to verify the output. The following test cases map the actual output to the expected output from the external documents. Then calculate a relative error the verified the correctness of the input values.

The Correctness of the Calculation

1. VerifyOutput-id4

   This test case used the external data from [Jacobsonand Jadhav] [3] as the expected output and the expected input latitude.

   Control: Automatic. The test cases contain cases that Φ_P = expected input latitude and $\Phi_P \neq$ expected input latitude.

   Initial State: Based on the assumption in SRS[5], I_S:1.35, and based on the assumption in [Jacobson and Jadhav] [3],
   ($year_{\text{Start}}$, day_{Start}, $month_{\text{Start}}$) ($year_{\text{End}}$, day_{End}, $month_{\text{End}}$)

(2018, 01, 01)-(2018, 31, 12)

Input: Φ_P

Output: The expected result will for the given inputs is the optimal tiltangle

id	Input	Output
id4.1	64.13	43
id4.2	63.13	43

[The expected result for the given inputs —SS]

Test Case Derivation: Based on the equation described in SRS[5], we get the **actual result**. Then we calculate the relative error using the data in the [JacobsonandJadhav][3] as our **expected result**. Calculate a **relative error** such that $|1 - \frac{\text{actual result}}{\text{expected result}}|$

[Justify the expected value given in the Output field —SS]

How the test will be performed:

- Build a linear graph using the expected input latitude as the x-axis and expected output as the y-axis.
- Input the input values from a file, verifyOutputId6.txt.
- Calculate the **actual result** by the equation descibes in in SRS[5]
- Place the point($P_{\text{actual input}}$)(x-axis: input latitude, y-axis: actual result) in the linear graph
- Find the point($P_{\text{upper bound}}$),the lowest upper bound of $P_{\text{actual input}}$ and point($P_{\text{lower bound}}$), the greatest upper bound of $P_{\text{actual input}}$
- Calculate the area between $P_{\text{upper bound}}$ and $P_{\text{actual input}}$; and $P_{\text{lower bound}}$ and $P_{\text{actual input}}$ using the equation,
  $Area = \frac{|(x_{\text{input latitude}} - x_{\text{expected latitude}})| \times |(y_{\text{actual result}} - y_{\text{expected result}})|}{2}$
- If $Area_{\text{actual input - }\textbf{upper bound}} < Area_{\text{actual input - }\textbf{lower bound}}$, then **expected result** = the y-axis of $P_{\text{upper bound}}$, otherwise **expected result** = the y-axis of $P_{\text{lower bound}}$
- Verified the output by the test case derivation instruction.

- Display the relative error in the file, resultVerifyOutputId6.txt
- If all the **relative error** $\leq$ errorTolerance, then the test success, otherwise the test fails.

2. VerifyOutput-id5

This test case used the external data from [Landau][4] [This is redundant to manually do the citation and to use BibTeX. Just use BibTeX. If you want to use the author year style, use the Natbib option for BibTeX. —SS] as the expected output, expected input latitude

Control: Automatic. The test cases contain cases that Φ_P = expected input latitude, $\Phi_P \neq$ expected input latitude.

Initial State: Basedon the assumption in SRS[5], I_S: 1.35, and based on the assumption in [Landau][4], the days duration of the winter in northern hemisphere is from
($year_{\text{Start}}$, day_{Start}, $month_{\text{Start}}$) ($year_{\text{End}}$, day_{End}, $month_{\text{End}}$)
(2018, 05, 10)-(2019, 05, 03)

Input: Φ_P

Output: The expected result will for the given inputs is average the solar intensity during winter.

id	Input	Output
id5.1	30	5.6
id5.2	31	5.6

[Use table to summarize your test cases. —SS][Yes —Author]

[The expected result for the given inputs —SS]

Test Case Derivation: Based on the equation described in SRS[5], we get the expected result. Calculate a **relative error** such that $|1 - \frac{\text{actual output}}{\text{expected output}}|$ [Your error will not be 0. It will be a small number. Your test case should require reporting this number. —SS] [Yes, I was trying to express the relative error will be close to 0 by ≈ 0. —Author]

[Justify the expected value given in the Output field —SS]

How the test will be performed:

- Input the input values from a file, verifyOutputId7.txt.
- Calculate the daily solar intensity during winter
- Calculate the average solar intensity during winter, using the equation,
  the average solar intensity $= \frac{\text{the sum of the daily solar intensity}}{\text{days duration of winter}}$
- Display the relative error in the file, resultVerifyOutputId7.txt.
- Output the average solar intensity as the actual output
- Verified the output by the test case derivation instruction.
- If all the **relative error** $\leq$ errorTolarance, then the test success, otherwise the test fails.

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. —SS] [Tests related to usability could include conducting a usability test and survey. —SS]

5.2.1 Reliability

The reliability of the System

1. reliability-id1
   Type: Manual

   Input/Condition: Software requirements

   Output: Test report.

   How the test will be performed: Follow the instruction of the software requirements. Provide the test report.
   The test case for the Sun Catcherrequirements is from section 5.1.1 to section 5.1.2

2. reliability-id2

   Input/Condition: Software requirements changed.

   Output: The corresponding trace table 1.

   How the test will be performed: If the software requirements change, then the trace table has to modify according to the changing. The trace table has to match the relationship between software requirements and the test cases.

5.2.2 Correctness

The correctness of the System

1. correctness-id3

   Type: Dynamic analysis - Code Coverage [good idea to report this number. —SS] [Yes, it does. My output will be the percentage of the coverage. —Author]

   Use the code coverage tool, Haskell Program Coverage(HPC), to test the code coverage. The description of HPC can be found in section 4.3.

   Input/Condition: The main code of Sun Catcher

   Output: The number of how percentage that the software has been coverage by the implementation.

   How the test will be performed:

   - Use the compiler, ghc-6.8.1 or later version, to active Haskell Program Coverage.
   - Enable hpc with the command line, -fhpc.
   - Implement the code of the Sun Catcher
   - Display the coverage in the file "correctnessid1.txt".
   - The test case success, if the output gets the 100% of the coverage, otherwise the test case fails.

5.2.3 Portability

The portability of the system

1. portability-id4

   Type: Manual

   Initial State: -

   Input/Condition: Implement Sun Catcherin diverse environments.

   Output/Result: The result from all the test case.

   How the test will be performed:

   - Implement Sun Catcheron the virtual machines with Windows operating system like Windows 10, Windows 8, Windows 7.
   - Implement Sun Catcheron the virtual machines with the Mac operating system like macOS Catalina , macOS Mojave
   - Implement Sun Catcheron the virtual machines with the Linux operating system [What is the OS and what is the virtual OS? Why not Linux too? Virtual box is easy to install and run. —SS][Yes, I have make the statement more explicite —Author]
   - Run the test cases in system VnV plan and unit VnV plan in all of the identified environment .[What do you mean to run every function? Why not simply say that the test suite will be run in all of the identified environments? —SS][Yes, that is what I mean —Author]
   - If function works, then success, otherwise fail.

5.2.4 Usability

The usability of the system

1. usability-id5

   Input/Condition: The output result of the Sun Catcher

   Output/Result: The energy absorption of the shows in the result should be increased by the time of adjustments the solar panel.

How the test will be performed:
Sun Catchershould provide the optimum angle of adjust one time and adjust multiple times during the inputs' period.

- Output the result to the file "FinalResult.txt".
- Observe the result, if the energy absorption increases by the time of adjustments, then the test case success. Otherwise the test case fail.

2. usability-id6

   Input/Condition: Usability Survey Questions, describing in the Table 4

   Output/Result:
   # Based on the number of inputs. The steps take to get the optimum tilt angle should be $\leq$ 4 steps

   The steps take to get the expected solar energy gaining $\leq$ 4 steps

   How many error alerts you take at the fist time using the software? $\leq$ 2 alerts

   How many error alerts you take at the second time using the software? $\leq$ 2 alerts

   # Based on the scale of the software. How long it need to get the optimum tilt angle $\leq$ 1 mins

   How long it need to get the expected solar energy gaining $\leq$ 1 mins

   The power consume after using the software for an hour $\leq$ 1

   The memory consume after install the software $\leq$ 1

   How the test will be performed: The survey should be filled by the people with or without technology background.
   The result of the survey are expected as above.

3. usability-id7

   Input/Condition: Usability Survey Questions, describing in the Table 3

Output/Result:
The Yes/No questions in the survey should be all filled with Yes.

How many time you open this software in a week? ≥ 0

$10 \leq$ How many stars would you like to give to this software?(1 - 10 starts) $10 \geq$ answer≥ 7

How the test will be performed: The survey should be filled by the people who have been using the software for a period of time.
The result of the survey are expected as above.

[Your project is fairly straightforward Sharon. If you want to aim for a maximum mark in this course, you'll need to do more than just implement your project. One place where you could add some extra challenge to the project is to consider assessing usability. I'm open to discussion if there is something else you would like to add instead. I see that you have a usability survey in the appendix, but you do not reference it from the main document. —SS]

5.3 Traceability Between Test Cases and Requirements

	R1	R2	R3	R4	R5	R6	R7	R8	R9
id1	X								
id2	X								
id3	X								
id4								X	
id5								X	

Table 1: Traceability Between Functional Requirements Test Cases and Requirements

	NFR1	NFR2	NFR3	NFR4	NFR5	NFR6
id1	X	X		X	X	
id2	X	X		X	X	
id3		X				
id4						X
id5			X			
id6			X			

Table 2: Traceability Between NonFunctional Requirements Test Cases and Requirements

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

References

[1] Koen Claessen. Quickcheck: Automatic testing of haskell programs, 2000.

[2] Andy Gill and Colin Runciman. Haskell program coverage, 2000.

[3] Mark Z. Jacobson and Vijaysinh Jadhav. World estimates of pv optimal tilt angles and ratios of sunlight incident upon tilted and tracked pv panels relative to horizontal panels. Technical Report CAS-17-01-SS, Department of Civil and Environmental Engineering, Stanford University, Stanford,USA, 2018.

[4] Charles R. Landau. Optimum tilt of solar panels, 2001.

[5] Yu-Shiuan Wu. Software requirements speci
cation for sun catcher, 2019.

6 Appendix

6.1 Symbolic Parameters

Symbol	Description	Value
I_S	Solar insensity	1.35
errorTolerance	Error Tolerance	1

6.2 Usability Survey Questions

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

Symbol	Answer
Do you think this software helps?	Yes/ No
Do you think this software is easy to use?	Yes/ No
Do you think this software is easy to use for elders?	Yes/ No
Do you think this software is easy to use for children(under 12)?	Yes/ No
Do you think this software works flawlessly?	Yes/ No
How many time you open this software in a week?	_____times
How many stars would you like to give to this software?(1 - 10 starts)	_____starts
Do you like to recommend this software to others?	Yes/ No

Table 3: Survey for home users

Symbol	**Answer**
The steps take to get the optimum tilt angle	_____steps
How long it need to get the optimum tilt angle	_____times
The steps take to get the expected solar energy gaining	_____steps
How long it need to get the expected solar energy gaining	_____times
The power consume after using the software for an hour	_____%
The memory consume after install the software	_____%
How many error alerts you take at the fist time using the software?	_____times
How many error alerts you take at the second time using the software?	_____times

Table 4: Survey for researching group