Project Title: Unit Verification and Validation Plan for Sun Catcher

Sharon (Yu-Shiuan) Wu

December 19, 2019

1 Revision History

Date	Version	Notes
2019/12/18	1.0	First Version
Date 2	1.1	Notes

Contents

List of Tables

[Do not include if not relevant —SS]

List of Figures

[Do not include if not relevant —SS]

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

[symbols, abbreviations or acronyms – you can reference the SRS, MG or MIS tables if needed —SS]

This document ... [provide an introductory blurb and roadmap of the unit V&V plan —SS]

3 General Information

3.1 Purpose

The purpose of this document is to verify the software, Sun Catcher. The test cases in this document follows the module defined in Designed documentation, MG and MIS. This document is not an repitation of the ducument, SystemVnVPlan. [Identify software that is being unit tested (verified). —SS]

3.2 Scope

In this document, the module, Day ADT Module, will not be tested. Day ADT Module is developed by using external framework "time: A time library" which is mentain by external developer, Ashley Yakeley. The framework's document can be found in [3]. Although Day ADT Module is an extension of the external library, it use the functions from the sources directly without overwriting the original sources. The module, Table-layout Module, has a low priority for verification. The purpose of this module is to write the output value to the designed file. Because this module is not effect to the output value, this module is considered having lower priority than others.

[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]

4 Plan

4.1 Verification and Validation Team

Main reviewer: Sharon (Yu-Shiuan) Wu
Secondary reviewer: Doctor Smith
[Probably just you. :-) —SS]

4.2 Automated Testing and Verification Tools

- 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] [1]
- Tasty: a testing framework for Haskell.

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. —SS]

4.3 Non-Testing Based Verification

- code walkthrough: Code walkthrough will be hold between Sharon (Yu-Shiuan) Wu and Doctor. Kal. The code notes with clear comment. The code should be writen by using external framework "HaTeX". The code should not be repeated and redundant.
- hlint: a framework for giving suggestions on the source code. The code should be not has any warning from hlint.

[List any approaches like code inspection, code walkthrough, symbolic execution etc. Enter not applicable if you do not plan on any non-testing based verification. —SS]

5 Unit Test Description

The test cases follow the module descibed in designed document, MIS. MIS can be found in the link `https://github.com/sharyuwu/optimum-tilt-of-solar-panels/blob/master/docs/VnVPlan/UnitVnVPlan/UnitVnVPlan.pdf`.

Input verification Module has the highest priority for verification. The purpose of this module is to test whether or not the input value is valid. Because the software will fail if implement an invalid input, this module has the highest priority.

Calculation Modul has the second priority for verification. The purpose of this module is to do necessary calculate for the output value for the requirements of the software. The accuracy of the output if this module will affect the final output value.

Control Modul has a lower priority for verification than others. The purpose of this module is to connect and associate other modules. Because this module is not effect on the value of the output, this module has a lower priority.

[Reference your MIS and explain your overall philosophy for test case selection. —SS]

5.1 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]

5.2 Modules

5.2.1 Input Verification Module

[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. InputBounds-id1
   This test case tests the capability of capture the invalid input value of Φ_P.

   Control: Automatic.

   Initial State: No input value
   Input: Φ_P

   Output: The expected result will for the given inputs is based on the input value. If Φ_P is between the value of [-90..90], output “valid”. Otherwise ouput “invalid”.

id	Input	Output
id1.1	90	valid
id1.2	-90	valid
id1.3	89.9	valid
id1.4	-89.9	valid
id1.5	0	valid
id1.6	91	invalid
id1.7	-91	invalid
id1.8	90.1	invalid
id1.9	-90.1	invalid

Test Case Derivation: The test case is testing the ability of capture the input value, Φ_P. The constraint of Φ_P is driven by the geographic coordinate that the range from North pole to South pole is from 90 to -90. This constraint can be found in SRS [2].

How the test will be performed:

- Input the designed test case input values by using testing framework, Tasty.
- Write the output in the file, resultInputBounds-id1.txt.
- Verified the output showing in the file by match the result with the expected output above.
- If all the result match with the expected output, then the test case success.
- Otherwise the test case fails.

2. InputBounds-id2

   This test case test the capability of capture the invalid value of dayT.

   Control: Automatic.

   Initial State: No any given value.

Input: Based on calendar, the given inputs is: ($year_{\text{Start}}$, $month_{\text{Start}}$, day_{Start})

Output: The expected result will for the given inputs is showing if the input date exist in the standard calendar.
The existence of the day is based on the fact, if the day exist in the standard calendar.

id	Input	Output
id2.1	(0, 0, 0)	not exist
id2.2	(-1, -1, -1)	not exist
id2.3	(2020, -1, 29)	not exist
id2.4	(2020, 02, -1)	not exist
id2.5	(-1, 02, 29)	not exist
id2.6	(2020, 13, -1)	not exist
id2.7	(2020, 02, 29)	not exist
id2.8	(2020, 02, 28)	exist

[The expected result for the given inputs —SS]

Test Case Derivation: The expected result for the given input is based on the standard calendar. If the given input is exist in the calendar, display "exist", otherwise display "not exist".

[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.
- Write the output in the file, resultInputBounds-id2.txt.
- Verified the output showing in the file by match the result with the expected output above.
- If all the result match with the expected output, then the test case success.
- Otherwise the test case fails.

3. InputBounds-id3

This test case test the capability of capture the invalid input days, which means the end day is smaller than start day.

Control: Automatic.

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

Output: The expected result will for the given inputs is based on the standard calendar. If the end day < start day then output "invalid", otherwise output "valid".

id	**Input**	**Output**
id3.1	(2020, 02, 28) - (2021, 02, 28)	valid
id3.2	(2020, 02, 28) - (2019, 02, 28)	invalid
id3.3	(2020, 02, 28) - (2020, 01, 28)	invalid
id3.4	(2020, 02, 28) - (2020, 02, 27)	invalid
id3.5	(2020, 02, 28) - (2020, 02, 28)	valid

Test Case Derivation: This test is driven by the standard calendar system.
By the fact days can be sorted, if the end day is the day after the start day in the calendar, output "valid". Otherwise output "invalid".

How the test will be performed:

- Input the designed test case input values by using testing framework, Tasty.
- Write the output in the file, resultInputBounds-id3.txt.
- Verified the output showing in the file by match the result with the expected output above.
- If all the result match with the expected output, then the test case success.
- Otherwise the test case fails.

5.2.2 Calculation Module

1. calculation-id4
   This test case tests the capability of calculate the zenith angle.

   Control: Automatic.

   Initial State: Giving a fixed sun declination angle. $\delta_{date} = 20$
   Input: Φ_P

   Output: The expected result will for the given inputs is based on the input value. The output is calculated by the method descibed in MIS.

id	Input	Output
id4.1	43.250943	23.250943
id4.2	-43.250943	-23.250943
id4.3	-89.250943	-69.250943
id4.4	89.250943	69.250943

   Test Case Derivation: The output of this test case should follow the constraint of the latitude. The values should between [-90..90]. The equation of calculating the output is:
   (δ_{date} * $\Phi_P < 0 \Rightarrow \delta_{date} + \Phi_P$ | True $\Rightarrow \delta_{date}$ - Φ_P). Calculate an **absolute error**, such that | expected output - actaul output |.

   How the test will be performed:

   - Input the designed test case input values by using testing framework, Tasty.
   - Verified the output showing in the file by calculating the **absolute error**.
   - Write the absolute error in the file, resultCalculation-id4.txt.
   - If all the absolute error is equal to 0, then the test case success.
   - Otherwise the test case fails.

2. calculation-id5
   This test case tests the capability of calculate the sun Intensity.

   Control: Automatic.

Initial State: Giving a fixed solar intensity $I_S = 1.35$
Input: $\theta_{S_{\text{date}}}$

Output: The expected result will for the given inputs is based on the input value. The output is calculated by the method descibed in MIS.

id	Input	Output
id5.1	23.250943	1.0883942
id5.2	-23.250943	1.0883942
id5.3	-69.250943	2.8226608
id5.4	69.250943	2.8226608

Test Case Derivation: The equation of calculating the output is: $I_S \cdot (\frac{1.00}{\text{energy}})^{sec(\theta_{S_{\text{date}}})}$. Calculate an **relative error** such that $|1 - \frac{\text{actual result}}{\text{expected result}}|$.

How the test will be performed:

- Input the designed test case input values by using testing framework, Tasty.
- Verified the output showing in the file by calculating the **relative error**.
- Write the relative error in the file, resultCalculation-id5.txt.
- If all the relative error $<$ error Tolerance, then the test case success.
- Otherwise the test case fails.

5.2.3 Control Module

This test case test the coverage of the Control Module.

3. control-id6

Type: Dynamic analysis - Code Coverage

Input/Condition: Call the main function in code of Control Module

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

Test Case Derivation: Use the code coverage tool, Haskell Program Coverage(HPC), to test the code coverage. The Control Module is controlling the process of geting the final output. Therefore, by testing the coverage of it, the output number can represent the coverage of the entire program.

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 output
- Write the result in the file, resultControl-id6.txt.
- The test case success, if the output gets the 100% of the coverage,
- Otherwise the test case fails.

5.3 Tests for Nonfunctional Requirements

This software do not have any nonfunctional requirements for unit testing.

[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]

5.4 Traceability Between Test Cases and Modules

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

References

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

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

[3] Ashley Yakeley. time: A time library, 2019.

	R1	R2	R3	R4	R5	R6	R7	R8	R9
id1		X							
id2		X							
id3		X							
id4				X					
id5				X	X	X	X		
id6			X	X				X	X

Table 1: Traceability Between Functional Requirements Test Cases and Requirements

6 Appendix

[This is where you can place additional information, as appropriate —SS]

6.1 Symbolic Parameters

[The definition of the test cases may call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance. —SS]

Symbol	Description	Value
I_S	Solar insensity	1.35
error Tolerance	Error Tolerance	1