Project Title: Unit Verification and Validation Plan for Sun Catcher

Sharon (Yu-Shiuan) Wu

December 18, 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 [**?**]. 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]

5.1.1 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. ...

5.1.2 Module 2

...

5.2 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]

5.2.1 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:

5.2.2 Module ?

...

5.3 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.

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]