openOBD

Automotive Telematics Unit

Validation - v3.1

Authors:

- Nicholas Mulvenna
- Kaibo Ma
- Ehsan Ahmadi
- Isaiah Thiessen

Table of Contents

History of Revisions

Version	Authors	Date	Changes
1	Isaiah Thiessen Kaibo Ma Nicholas Mulvenna Ehsan Ahmadi	2016-11-28	Initial Draft
2	Kaibo Ma	2017-02-13	Added OBD2 Validation for Simulation Testing and Actual Testing
3	Isaiah Thiessen Nicholas Mulvenna	2017-04-05	Removed plan to test heat output and power consumption
3.1	Nicholas Mulvenna	2017-04-17	Wording

List of Figures

List of Technical Terms

Term	**Definition**
Regression Testing	Regression testing is a type of software testing that verifies that software previously developed and tested still performs correctly even after it was changed or interfaced with other software.
Smoke Testing	Also known as “Build Verification Testing”, smoke testing is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work.
Unit Testing	Usually run by developers to test each function as a single entity. Tested against input and output.
Integration Testing	Run by developers to test newly created functionality once put into the rest of the system. Usually tested against specifications.
System Testing	Also known as Regression Testing, where developer or Quality Assurance Engineers test the whole system with newly added features. Usually tested against requirements.

User Acceptance Testing	Usually last step in testing cycle where the system as a whole is verified and validated to meet all client needs.

1 - Introduction

This document states the validation plan for the Automotive Telematics Unit. It will go over the techniques and methodologies in testing and ensuring quality control and safety for end users.

2 - Actions Taken Toward Product Validation

2.1 - Manual Tests

Approach:

- Manual testing is done using a terminal interface for FS03. We run basic user tests for functions like getting sensor readings from:
    - GPS
    - OBD2
    - Temperature
    - Accelerometer
    - Gyroscope

  We run Smoke Tests against the command line interface. Essentially we try to emulate what an end user may want to perform with our product. All features are tested against multiple trials to make sure there are no anomalies. It is important to cover all major feature of the application so that we can ensure the quality of the program.
- Manual measurements using lab tools for FS07
    - Oscilloscope to check CAN bus transmission is stable
    - Multimeter to ensure level converters are stepping up and down correctly
- **OBD2: There were two stages for OBD2 testing and debugging for FS01.**

    - **Stage One:** Was to test our software and hardware against an OBD2 CAN bus simulator that was custom written over an arduino. We connected our Pi and hardware to an arduino running software that simulated a CAN bus network of a vehicle. Our Pi would read the data from Arduino much like how we would read from the OBD2 port of an actual vehicle. Our Pi would either listen over the simulated CAN bus or send queries over the CAN bus to the arduino to get responses and data. For example, we would send requests to the virtual engine to request rpms. This method allowed us to be able to test our software and hardware without the need to be in a vehicle. It will allow us to transfer the data and knowledge of testing with the simulator to that of testing with an actual vehicle.
    - **Stage 2:** We test against the OBD2 Port of the car. We take the knowledge and confidence that we obtained with testing against a simulator to test with an actual OBD2 network. We connected our device to listen to the CAN bus network of the OBD2 port of a 2015 Toyota Camry. We were able to obtain data transmissions from the car as well as the ability to send the network queries for rpm, speed, and fuel levels. The car would respond instantaneously with what data we queried. From testing on Stage 2 with the car, we can now validate FS01.

2.2 - Automated Tests

Approach:

- We perform automated software tests using the pytest library. We begin at client needs and determine our requirements for the project. The development team then designs and implements the code as well as its unit tests. If they pass, the newly written code is integrated into the system to see if it also performs according to the design. Then once that passes, the whole system is tested against the requirements, also known as regression testing. Finally, our user acceptance tests (using client needs as a rubric) must pass.

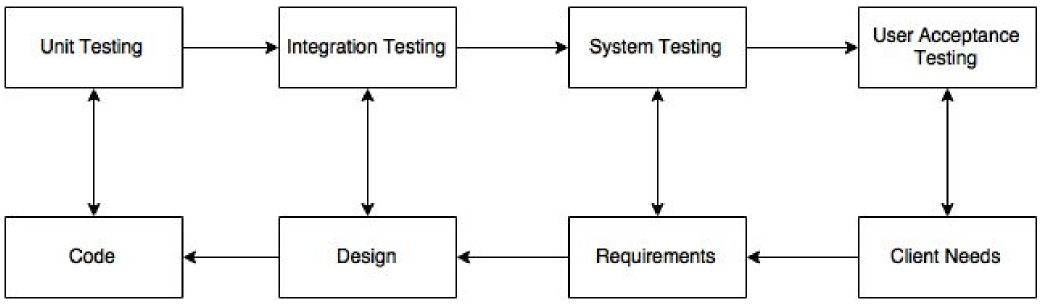

Figure 1.0: Testing Cycle

- During development, mocks are used in place of classes that directly interact with hardware in some tests to fulfill requirement NS04.

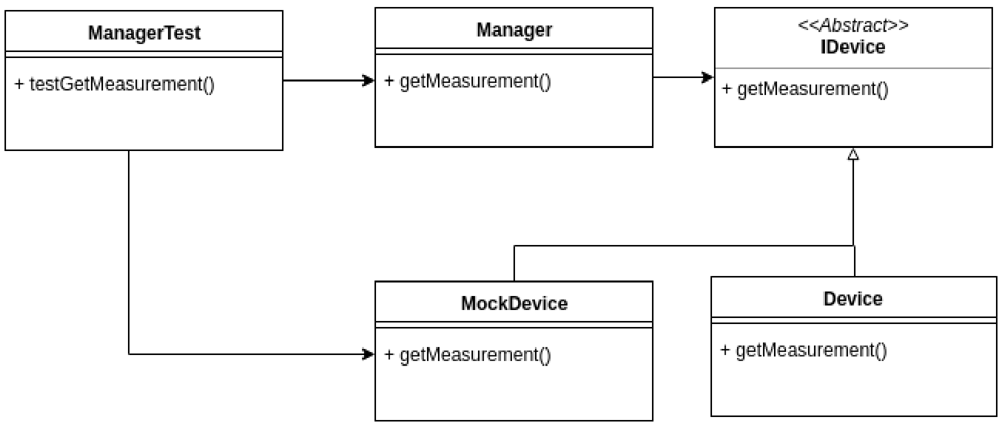

Figure 2.0 Testing Using Mocks

  - Figure 2.0 demonstrates the use of mocks when testing classes that do not directly depend on hardware devices (such as sensors or the GPS module). [1] [2] [3] In this example, suppose that a `Manager` class depends on an abstract `IDevice` class. When running the application, a concrete `Device` class is used. However, a concrete `MockDevice` class can imitate the physical hardware's behaviour so `ManagerTest` can run without it.
- Hardware tests were performed using short test scripts to test that sensors read correctly and consistently, in order to validate that FS02 is fulfilled. These scripts are written explicitly for each separate device. Tests performed to-date for all sensors have been successful.

- A CAN bus simulator was used to generate random data on a bus, which was read using the current prototype to verify that data was being read correctly, per FS01. This test was successful and will continue to be used as a test method for future iterations.
- Test points are included on the PCB in order to verify that the hardware is functioning correctly.

3 - References

[1]"Progress Before Hardware", Wingman Software, 2004. [Online]. Available: https://wingman-sw.com/papers/ProgressBeforeHardware.pdf. [Accessed: 27- Nov- 2016].

[2]A. Jordan Schaenzle, "The mock object approach to test-driven development", Embedded, 2016. [Online]. Available: http://www.embedded.com/design/prototyping-and-development/4398723/The-mock-object-approach-to-test-driven-development. [Accessed: 27- Nov- 2016].

[3]"For embedded TDD, don't worry so much about testing on the target", ElectronVector - Test-First Embedded Software, 2016. [Online]. Available: http://www.electronvector.com/blog/for-embedded-tdd-dont-worry-so-much-about-testing-on-the-target. [Accessed: 27- Nov- 2016].

[4] Kolawa, Adam; Huizinga, Dorota (2007). Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press. pp. 41–43. ISBN 0-470-04212-5.

[5] "Smoke Testing." *Software Testing Fundamentals*. N.p., n.d. Web. 28 Nov. 2016. <http://softwaretestingfundamentals.com/smoke-testing/>.

[6] @whatisdotcom. "What Is Regression Testing? - Definition from WhatIs.com." *SearchSoftwareQuality*. N.p., n.d. Web. 28 Nov. 2016. <http://searchsoftwarequality.techtarget.com/definition/regression-testing>.