Unit Testing for Command and Control Systems

Joshua Alexander
Kennedy Space Center
Computer Science / Math

NIFS Spring 2018
Date: 27 03 2018

Unit Testing for Command and Control Systems

Joshua A. Alexander
Towson University, Towson, MD, 21252

Unit tests were created to evaluate the functionality of a Data Generation and Publication tool for a command and control system. These unit tests are developed to constantly evaluate the tool and ensure it functions properly as the command and control system grows in size and scope. Unit tests are a crucial part of testing any software project and are especially instrumental in the development of a command and control system. They save resources, time and costs associated with testing, and catch issues before they become increasingly difficult and costly. The unit tests produced for the Data Generation and Publication tool to be used in a command and control system assure the users and stakeholders of its functionality and offer assurances which are vital in the launching of spacecraft safely.

Nomenclature

SDLC = software development lifecycle
OOP = object-oriented programming
CI = continuous integration

I. Introduction

Command and Control Systems play a vital role in launching spacecraft into orbit. These systems are often complex and rely on a large number of hardware and software components working in unison to issue commands, process metrics, and monitor hardware. Development for a new Command and Control System for a new launch vehicle has been underway for several years. The launch vehicle is being designed to be modular and allow for different configurations that vary based on the payload and requirements of the mission. These missions can take cargo and people into Earth orbit and beyond. On such an ambitious project, a process needs to be followed in order to produce the required software for the command and control system. The software development lifecycle is an efficient methodology for delivering quality software for specific needs.

One important part of the software development lifecycle (SDLC) is the Testing phase, where the software is tested to see if it performs as expected and meets the requirements. Testing is an essential part of any complex and safety critical software project. Testing is used to account for unforeseen issues and is used to verify functionality and safety of the system. It's imperative that both software and hardware components of such a system are extensively tested as the consequences of unforeseen issues can be costly and put lives at risk. Often testing begins even before the testing phase of the SDLC. During the development phase of the SDLC, unit tests are often created along with source code in order to verify specific parts of the software. This ensures specific components of software behave as expected. It is much easier to make changes or fixes to the source code before the system grows and other components depend on the unit which can break after changes. This leads to better software quality and captures small issues before other types of testing begin such as integration and acceptance testing. During these phases of testing, increasing amounts of software components work together resulting in issues being more difficult to narrow down and address.

The goal of a unit test is to test the software component and verify that the specific unit behaves as expected under all possible scenarios. Unit testing is vital for any software project, especially for a command and control system for a new launch vehicle.

II. Unit Testing

The first part of writing proper unit tests for software components is to understand the software to be tested. It is much easier for a software engineer to develop unit tests along with the source code they are developing as they best understand their software. Sometimes, development of unit tests may be tasked to a different developer for a number of reasons. This developer must understand the software component they need to write tests for in order to fully verify

the functionality and test all aspects of that software component. When the developer understands the functionality of the code, they can then create tests that verify it performs as intended. Once an understanding of the function of the specific code is attained a testing framework is selected.

Testing frameworks are a tool a developer uses to create unit tests. They provide tools and functions to verify aspects of a software unit. These can include methods to setup and teardown dependencies for a unit of software that are required to properly test. Testing frameworks also can mock objects that are needed in a unit test. In Object Oriented Programming (OOP), mocking is used in unit tests for required software objects that are not being tested but are needed for that unit test. It is key that the source code under test is not changed for the test, but other necessary components that are not being tested can be mocked.

There are a large number of testing frameworks for a variety of software languages but most create unit tests and make assertions on the behavior of the code. Assertions in unit testing verify that the actual result of some code matches what is expected. Assertions are the most basic part of any testing framework. For example, a developer can write a unit test to verify that a specific method returns a "True" Boolean when given certain parameters. The framework provides function would take an expression and its parameters and test if the result was true. Upon running the test, the method would be called and if the method actually returned true then the test would pass. Otherwise, the test would fail.

Many unit tests and assertions may be required to test all aspects of a particular unit of source code. A developer must be sure to account for all inputs and scenarios and test each one in the appropriate unit test to fully verify functionality. All the unit tests for a specific piece of software are called a test suite. A test suite contains methods written with the testing framework to make assertions on the source code under different test cases and verify they are true.

III. Design

During my internship, I had a variety of assignments for the command and control system currently being developed. One of my main tasks involved creating unit tests for a Data Generation and Publication tool to be used in testing the command and control system. This tool was developed by Jason Kapusta[1] and Craig Einstein[2]. This tool is used to test different functions of the command and control system[3]. The tool allows a user to create a list of messages specifying the delay, repeat option, and who is publishing and to what. This information is stored in a configuration file which is parsed to setup the tool for a specific routine to publish. Once a list is created, the messages can be published in order with the given delay to simulate the real system. Unit tests needed to be developed for this tool in order to assess its function and accuracy.

A testing framework was chosen to evaluate this tool using assertions and provided functions to verify specific expectations of the tool. To better understand the testing framework for the data generation and publication tool, my first steps included thoroughly reading the description of the tool as well as consulting with Jason Kapusta on any parts that were unclear. With an understanding of its function the actual designing and writing of unit tests can begin. My work then involved designing and writing tests to make assertions on the tool and determine how it would respond to various inputs and actions as well as test sequences for the publishing of messages.

The data generation and publication tool required several dependencies in order to function. Since these dependencies weren't being tested, mocks were created. These mocks imitated the behavior of the software component enough to be used to test the data generator and publication tool utilizing these mocked components. Mocking the required components was done after seeking assistance from Jamie Szafran[4]. The mock allowed setting return messages that were called from the data generation and publication tool to test both error and non-error conditions.

Tests involved publishing messages to the command and control system and then verifying the actions were actually published, logged, and the correct metrics were kept for each publication. This was done by creating one of several specific messages with its associated data and asserting using the testing framework's assert functions that the published action was the one given in the command. Also, the parsing of several different configuration files was checked to ensure the tool read the settings correctly from a configuration file and set the appropriate parameters for the tool. Using the testing framework, assertions were also made about the expected actions that should be published given a command string in the configuration file. A dynamic cast was required to check if the returned action was of the correct type or else the test would fail.

Some issues encountered designing unit tests were that some of the source code being tested contained protected methods or data members. This made it difficult to verify method functionality and the values of the data members. To overcome this issue, subclasses were created that simply re-designated protected classes as public for visibility so the testing framework could access otherwise unavailable functions and data members necessary to make assertions

on them. This allowed testing of the source code and protected members without changing the source code which is a necessity when unit testing.

Upon completing a unit test, the tests are compiled and run to verify that they pass. In the testing framework, if a test fails the output will show the reason for failure such as an assertion that did not assert correctly or an exception that was thrown. Throughout the process of designing a unit test, it's important to remember to not write unit tests that will pass, but instead write tests that will verify expectations of the source code.

Written unit tests are also code reviewed to ensure that they sufficiently test the code and do so efficiently. Unit tests are frequently used in a Continuous Integration (CI) development environment. In a CI environment, code is built automatically according to a schedule or trigger from source code changes. Unit tests can also be run as part of this continuous integration to reevaluate source code automatically when there are changes. It is important that these tests are optimized for runtime and resource utilization since tests may be run many times a day or any time changes are made. Optimization of unit tests can frequently save system resources and electricity as a result of decreased build and test durations. Optimization is also addressed in code review. During code reviews peers review written source code to look for mistakes or opportunities for improvement. Code reviews along with unit tests greatly contribute to ensuring the quality and safety of the command and control system.

IV. Analysis

The unit tests created for the Data Generation and Publishing tool provide proof of the functionality and accuracy of the tool and validate its use for the command and control system. These tests cover all the scenarios and conditions to fully validate the use of this tool and guarantee its accuracy. Users of this tool can be assured of its use and that it will function as expected. Guarantees for individual software components like a command and control system help the system in the long run to expand and reduces the risk of issues during later testing such as integration and level 5 testing. As the systems grows, rerunning tests will always reevaluate the software component for any issues that may arise.

V. Conclusion

Unit testing is an instrumental part of any professional software developer's skills. These tests catch problems, bugs, or inaccurate results/behavior of source code before the later stages in development where changes to code can be costly in time and effort. This method of testing is the earliest testing and often prevents the most issues as it is the first to check for them. For a command and control system, verifying individual constituents during development will save the project resources and time down the line from potentially undetected issues that may have slipped through without the use of unit testing. It can also be said that the unit tests ensure the function of the specific system and can be continuously run to ensure as the system grows that these components still function as anticipated.

Testing at all levels is an integral component to any software project, especially command and control systems for launch systems. These command and control systems have risks and are safety critical. For this reason, when launching spacecraft carrying people or multimillion dollar payloads, there can never be too much testing. This testing starts with small unit tests while development is underway.

The unit tests created for the data generation and publication tool provide assurances for all those involved with actively developing the command and control system. Assurances are a rarity in spaceflight, but with the help of unit testing and other forms of testing, these assurances can lead to safer and increasingly reliable command and control systems.

Acknowledgments

I would like to personally thank Jamie Szafran and Jason Kapusta for their guidance and support. I would also like to thank the Education department and NE-XS at Kennedy Space Center for this great opportunity to learn and gain real world experience as a Command and Control Systems Development Intern.

References

[1] Systems Architect, Test and Operations Support Contract, Kennedy Space Center
[2] Command and Control Systems Intern Fall 2017, Kennedy Space Center, Boston University

[3] Einstein, C. J., "Efficient Data Generation and Publication as a Testing Tool," Intern Fall 2017 Final Report, Kennedy Space Center, 2017.

[4] Software Engineer – Aerospace Technologist, NE-XS, Kennedy Space Center

Towson University Logo, https://www.towson.edu/universitymarketing/brand/downloads.html