Utilizing Testing Frameworks for Launch Control Systems Continuous Integration

Alexander McLaughlin
KENNEDY SPACE CENTER
Computer Science
NIFS Spring 2018

Date: 13/04/2018

Utilizing Testing Frameworks for Launch Control Systems Continuous Integration

Alexander McLaughlin[1]

University of Central Florida, Orlando, FL, 32816

Nomenclature

API = Application Programming Interface

SLS = Space Launch System

USRA = Universities Space Research Association

I. Introduction

Command and control software is an integral part of the launch procedure. The most important part of this type of software is its ability to communicate well with the user and relay information in a correctly formatted way such that the user can understand the data. There is a tool that aides the communication between the different parts of the system, and effectively, the user. This instrument is capable of taking several complex values and ensuring that they are correctly sorted into their distinctive message values and distributed properly among the different facets of the system.

This tool will easily translate and publish the data inside of messages in the system to something that is readable and understandable. The tool also allows for transmission of the recorded data to the user, effectively ensuring the communication between different components of the system. As well as keeping track of messages and ensuring that the information contained within each of them reaches the correct location, this tool has the ability to keep track of its own statistics and determine how many messages passed in were erroneous and how many were successfully transmitted. It is able to check and see what the total message failure count is when an invalid message is given, as well as the number of different messages and their respective types passed into the tool.

This tool is of great value to the new space launch system (SLS). As such, the tool must be thoroughly tested with test cases that, although improbable, are possible, where the tool may not function properly. Testing an interface this complex is necessary to ensure mission safety and create unlikely scenarios where the tool would work as intended, and stretch its limits to test that even under the most uncommon conditions it would still continue to function.

This software will be an important part of the control system for the newest spacecraft which will fly deeper into space than humans have ever travelled. It will fly beyond the moon, into deep space to mars and perhaps set the groundwork for a manned mission even further to create more opportunities for interplanetary and even interstellar travel by humans. This mission relies heavily on software and hardware to ensure the safety of the humans that will be on board and therefore must be checked, exhausting each and every different situation, such that there is not a doubt surrounding the well-being of the humans aboard the rocket. That is why testing is such an important part of the mission. It provides evidence that the systems aboard the rocket and on the launch pad are safe.

[1] Software Developer, Engineering, Kennedy Space Center, University of Central Florida

II. Unit Testing Framework

The framework that was used for the unit testing of the C++ files in our tool for recording and transmitting data is a testing tool aimed at verifying that the functions throughout the file are working functionally. The framework is used in a way such that the file being tested can easily be scrutinized for any issues in the code and corrected if needed. The unit test is aimed specifically at this goal, to take each individual and minute piece of code and examine it until all possible circumstances are exhausted, therefore, the framework used was a wonderful asset for testing the data tool.

The framework was used to test each class and function in the data tool that had been created to generate and record information then transmit and publish it. Many of the classes that were tested had to be isolated from outside classes to avoid any clashing and accidental failures due to an error from another class. This was done through the use of a very important tool in unit testing known as mocking. Mocking is similar to how it sounds, it mocks the functionality of the class while guaranteeing that it will not cause any errors, but rather that it will only assist the class that is being tested and nothing more. Mocking is essential in analyzing code and unit testing properly, and had to be used for most classes which used the Application Programming Interface (API) since this was a relatively difficult class to initialize

III. Importance of Unit Testing

Unit testing is important for mission critical code to be properly analyzed to the fullest extent and puts emphasis on how well the software recovers in all possible situations. This type of testing usually incorporates edge cases. Edge cases are the situations that occur when the code is stretched to, and perhaps beyond, its limits. It is imperative that the code recovers from something it may not recognize, such as data it may not have seen before or improper input from a user. In any system as complex as the new space launch system, the software and hardware on-board and on the ground are expected to be in peak condition, tested to the fullest extent, and compliant with all standards for launch.

The SLS relies heavily on ground control to make sure that all launch procedures go smoothly and that there are no anomalies. One tool that ground control uses, the data recorder tool, is connected to the many ground system applications that play an important role in making sure the rocket launches successfully. Thus, it is imperative that this tool be tested as rigorously as any other essential tool incorporated into the SLS.

IV. Unit Testing Outline

The data recorder tool was made up of a formatter, several types of messages differing in what information they sent but formatted similarly, and a class that kept statistics on each of the messages that could be instantiated and passed as a parameter to track a specific message. It also contained the actual data recorder class which incorporated all messages, the formatter, and statistic tracker into one and acted somewhat as a driver, or a program that calls separate programs to control data more effectively. My primary role was to test the recording part - the other part, which would be used for storing data was allocated to a fellow intern. Collaboration, question asking, and quite a bit of controlled testing got us to the point that we were able to complete the task of testing the data recorder and storage unit. The most difficult part of the entire task was understanding the structure of the preexisting API.

The API that needed to be implemented to use in testing was quite complex and working with a mock for the first time was a daunting task. Using a mock, or mocking a class in unit testing, is the act of creating an instance of an external object, outside of the one that you are testing, such that all of its function return values can be controlled to scrutinize as closely as possible, only the class that you are testing. If, for instance, you were to create an instance of a class you were unit testing and that class required a constructor input of another class object, you would want to verify that any failures are not caused by the other class object, but rather by the class being tested.

In unit testing you only want to test one class at a time, therefore any other objects that are used throughout a test should be properly initialized, defined, and utilized (proper function calls, correct parameter input, avoiding pointer dereferencing) to avoid them throwing any unexpected errors not caused by the main class. Mocking creates a situation where the programmer does not need to inspect each and every aspect of a class which they are not testing. Rather than wasting time inspecting a class that is expected to function properly in the first place, the programmer is able focus on the main class and verify that it works correctly.

Once mocks were implemented, the API was fairly easy to work with because most of the functionality that would have been difficult and very tedious to program was as simple as a single function call to the mock class, telling the

actual function call what to return. Simply put, unit testing is difficult and time consuming without the mocking functionality that is built into large classes such as the aforementioned API.

V. Test Outcomes

Unit testing is not done when the programmer believes he has tested enough, or when the code compiles and runs the correct way, rather, it must go through several peer-reviewing sessions, where the code will be looked at by people on your team to ensure that your code is correct and that you have not made any mistakes. Doing so helps to ensure that your code is properly testing the class and any functionality of the class that you may have missed in testing can be corrected and resubmitted. Once code review has been completed, your code will be run through a program to determine how well the class has been tested.

This coverage analyzer looks at the code in your unit tests to determine if it touches all different inputs, parameter passes, and other parts of the class you are testing. This will provide a percentage of how much of the code you are testing has been tested. After completion of any changes discussed in peer review and running the code through the coverage analyzer to ensure that it meets the specific percentage required, then the class has been successfully unit tested. Using the same methods for other testing frameworks, such as JUnit or CUnit, unit tests can be performed on Java files and C files in a similar way.

VI. Conclusion

Verifying the functionality of a system becomes increasingly important as we approach the launch date of the United States' new space launch system. It is important that every situation, no matter how small or seemingly unimportant, is tested. The testing done by this team is essential to guarantee the safety and performance of ground systems which are integrated with the launch of the SLS. The work done on the data recorder and data generator will assist in testing essential components of the larger system.

Coding the unit tests for the data recorder and data generator proves with absolute certainty that the message types, statistic tracking class, and formatting class contained within the data recorder work properly with the other classes and functions that they have the ability to call. The safety of the SLS and the employees of the Kennedy Space Center are of the utmost importance, so it is essential for each and every mission that the components of all rockets, internal and external systems pass checks and tests ensuring that they are as safe as can possibly be.

VII. Acknowledgments

It has been an honor participating in this internship and I would like to thank Jill Giles, Jamie Szafran, Caylyne Shelton, Oscar Brooks, Jason Kapusta, the Kennedy Space Center Education Department, and the Universities Space Research Association (USRA) for their continued support and for making this opportunity possible.

VIII. References

https://sciences.ucf.edu/biology/wp-content/uploads/sites/87/2011/09/UCF_logo.png

http://www.usra.edu/

Einstein, Craig, "Efficient Data Generation and Publication as a Testing Tool," NASA KSC – Internship Final Report, 2017.