Command and Control System Automated Testing

Tevin Mantock
John F. Kennedy Space Center
Major: Computer Science

Command and Control System
Spring Session
Date: 04-12-2019

Command and Control System Automated Testing

Tevin T. Mantock[1]
Oregon State University, Corvallis, Oregon, 97331

Nomenclature

CCS	=	Command and Control System
GUI	=	Graphical User Interface
IP	=	Internet Protocol
KSC	=	Kennedy Space Center
LCC	=	Launch Control Center
NASA	=	National Aeronautics and Space Administration
SLS	=	Space Launch System
TAS	=	Test Automation System

I. Introduction

The Kennedy Space Center (KSC) has developed its own Command and Control System for the launch of the Space Launch System (SLS) and Orion capsule. The Command and Control System (CCS) is used by console engineers for the launch and system checkout of aerospace vehicles. The CCS allows console engineers to read data from the flight hardware on the launch pad and from the ground control systems and allows console engineers to issue commands, like opening a valve, to the flight hardware and ground control systems. The CCS needs to interact with thousands of devices and hardware controllers for the spacecraft and ground systems, receive data from these devices, distribute the data to console engineers in real-time, and allow console engineers to issue commands to manipulate hardware on the launch pad. The system needs to be robust, fault tolerant, responsive, and fast.

In order to keep up with the pace of development of the CCS, a Test Automation System (TAS) is needed to validate the integrity of the system as a whole along with its individual components. Automated tests allow for faster development time, since tests can be ran through a Continuous Integration system and allow developers to check their code faster. Currently, the different modules, classes, and functions that make up the CCS are tested at the unit level, and the system level, with all the modules working together. My project was to implement a system for the data protocol layer of the Command Control System to be tested as a complete functional unit, with all of its classes and functions working together, but independent of the other modules of the CCS.

II. Objectives

The end goal for my implementation of a Test Automation System for the CCS is to automate testing for the data protocol layer between the end hardware devices for the spacecraft and ground control systems and the message bus for the CCS. The data protocol layer receives data from the flight hardware and translates it into a common format to be used in the CCS, and is also capable of translating the common format back to the format the hardware devices understand. In order to test the functionality, validity, and correctness of the data protocol layer, the following need to be accomplished first:

- Remotely start the data protocol layer
- Develop the simulated or mock vehicle and ground systems hardware
- Listen for and read output from the protocol data layer

III. Approach

A. Training and Setup

I spent the first month of my tenure at the National Aeronautics and Space Administration (NASA) waiting for access to source code, development tools, and a computer I could use for developing software. While I was waiting to receive access, I attended meetings and training for the CCS. After I received access to the software and a development unit, I took the time to explore the source code and learned how to start the data protocol layer. I found it difficult to use the provided online wiki for the CCS, since the setup section wasn't up-to-date and the instructions were left in a

[1] Software Development Intern, NE-XS, John F. Kennedy Space Center, Oregon State University

breadcrumb fashion, where one instruction would lead to another webpage or email and so on. I found that the best way to learn how to use the end product was to speak with the engineers and architects of the software.

B. Remotely Starting the Data Protocol Layer

In order to automate testing for the data protocol layer, the data protocol layer needs to be able to be consistently started using software. Typically, a software engineer for the data protocol layer would start the software from the terminal using a particular set of environment variables and commands. This is simple to implement in code, since many languages support executing commands from the command line. The most challenging portion for this task is making sure that the environment the software is executing on is consistent. The data protocol layer is intended to be used on a special implementation of the UNIX operating system, and it requires a large series of environment variables to be set. Fortunately, NASA provided development units with the special implementation of UNIX installed, which could be accessed through Secure Shell. The necessary environment variables needed for the data protocol layer to run could also be stored in a file that can be executed after successfully accessing the UNIX server. After I was able to demonstrate that I was able to do this manually, I implemented these steps in a higher level programming language that is human readable, so the process can be easily used, repeated, and abstracted for Test and Software Engineers to use when writing automated tests.

C. Mocking Vehicle Data

In order to test the functionality and validity of the data protocol layer, the TAS needs to be able to mock the input data the data protocol layer expects to receive from the flight hardware and ground control systems. Fortunately, NASA created a tool to simulate the SLS and other vehicles. Additionally, this tool does not require a Graphical User Interface (GUI) and can be used from the command line. The most difficult apart about mocking the vehicle data for the data protocol layer is mocking the production environment the data protocol layer expects to run in and receive vehicle data. The data protocol layer expects incoming data to come from certain Internet Protocol (IP) addresses using a specific networking protocol, which requires connected devices to be under a specific subnet. From speaking with the software architect of the CCS, I learned that NASA keeps its development servers on a particular set of subnets, which would allow me to run the vehicle mocking program from one of the servers and broadcast data to all of the other servers. After I learned about this configuration, I was able to manually start the vehicle mocking program from the command line and see that the data protocol layer is connected and receiving data from the mocking program. After I was able to validate my results, I implemented the software changes necessary for Test and Software Engineers to start up the vehicle mocking program and have it connect to the data protocol layer using the TAS.

D. Validating Output

The last requirement for testing the functionality, validity, and correctness of the data protocol layer is having a method to autonomously validate the translated output of the data protocol layer. Fortunately, the software architect for the CCS created a tool that can validate the output of the data protocol layer. The most difficult task for this portion of my project was getting the tool to start. Unfortunately, there were some environment variables that needed to be set that were not documented. Once I found the environment variables that needed to be set, I was able to get the tool to successfully start and keep running. However, I was unable to see the data the data protocol layer was sending or issue commands to the data protocol layer. With the help of the software architect, I learned that the server I was using to host the validation tool overwrites some environment variables, and that I would need to reset the values of these environment variables before I could start the validation tool. After I made sure that all the proper environment variables were set, I was able to start the validation tool, send commands to the data protocol layer to start sending data, and read the data the data protocol layer was sending through the validation tool. After I was able to make sure that my approach worked consistently, I implemented the software changes in the TAS to consistently open the validation tool and have it read data from the data protocol layer.

E. Bringing it All Together

After I was able to verify that I could remotely start the data protocol layer, send mock data to the data protocol layer, and validate the output of the data protocol layer, I needed to make sure that all of these separate tasks could be used autonomously and that the results of each task were consistent and repeatable. I needed to make sure that each portion was able to properly and successfully start up, and if any portion of setup failed, the test environment would be neatly cleaned up, so processes wouldn't be left orphaned and running on their respective servers. Once I implemented the software changes necessary to make sure that the setup and teardown for the TAS was consistent, resilient, and repeatable, I created a demonstration test. The test showed that once all the required setup components successfully started, a Test or Software Engineer can issue a command to the vehicle mocking program, which sent

data to the data protocol layer, and programmatically read the output of the validation tool to verify the functionality and correctness of the data protocol layer.

IV. Conclusion

Testing is an important part of the software development lifecycle. Even though it doesn't involve creating new features, it's not a trivial task and writing incorrect tests can have a large impact on the success of a project. Test Engineers should take great care in ensuring that their tests target the correct portion of code and that their test cases do not yield any false positives or false negatives. Currently, the data protocol layer for the CCS is tested at a unit and system level. That is to say that the individual classes and functions for the data protocol layer are tested at the unit level, and the data protocol layer is manually tested by users as part of the entire CCS at the system level. By the end of my project, Test and Software Engineers for the CCS will be able to effectively write tests against the data protocol layer and verify the validity and correctness of the data protocol layer as a complete functional unit independent of the rest of the CCS. Test and Software Engineers will now be able to write automated tests that can start the data protocol layer in an environment that's similar to production, receive vehicle data in a manner that's similar to the way it receives data in production, and validate the translated data output of the data protocol layer.

Acknowledgments

I would like to thank my mentors Jill Giles and Jamie Szafran for helping me get acquainted with NASA and the NE-XS department, helping me to get access to development tools and resources, and for helping me get the training I need for the position. I would also like to thank Jason Kapusta and Anthony Ciavarella for helping me overcome problems I encountered while trying to start the vehicle mocking tool and validation tool. I wouldn't have been able to overcome the issues I was experiencing with the environment setup, tool startup, and network connections without their help. I would also like to thank Brian Timmons and Michael Reed on the CCS team for helping me get set up with the code, build, and system startup for the data protocol layer. I would like to thank Mike Owens on the Test Automation System team for helping me with problems I experienced while using the Test Automation System. Lastly, I would like to thank the KSC Education Office for their help in making sure that my internship at NASA was a wonderful experience.

References

Not applicable for this paper.