Launch Control System Software Development Intern

Meriel Stein
Kennedy Space Center
Major: Computer Science
Spring Session
Date: 04.04.2017

Spaceport Command and Control System Automated Testing

Meriel Stein[1]
Rowan University, Glassboro, New Jersey, 08028

The Spaceport Command and Control System (SCCS) is the system interface to manipulate and monitor components of the National Aeronautics and Space Administration's (NASA) Ground Systems Development and Operations (GSDO) Program. To assure developers of its capabilities, SCCS requires comprehensive testing. For the past year, interns in the Electrical Engineering Division at Kennedy Space Center (KSC) have worked alongside full-time engineers to develop an automation framework and to innovate upon the current automation process to save the project time and money in implementing and executing these tests.

Nomenclature

CSV	= Comma Separated Values
TSV	= Tab Separated Values
GUI	= Graphical User Interface
KSC	= Kennedy Space Center
LCC	= Launch Control Center
NASA	= National Aeronautics and Space Administration
OCR	= Optical Character Recognition
Python	= High level, interpreted programming language
MD5	= Message Digest algorithm that produces a 128-bit hash value.
SCCS	= Spaceport Command and Control System
SSH	= Secure Shell

I. Introduction

This internship addresses the need to optimize the quality assurance portion of the software development process for future Human Spaceflight Program launch software. Introducing automated software testing would greatly reduce the resources demanded by manual testing. This will allow the software engineers, who previously spent days at a time executing dozens of test cases by hand, to dedicate their skills and time to other facets of the project. In addition, automated testing reduces the window of opportunity for human error that can appear during long hours of rigorous testing. Once these automated test cases are rigorously vetted and subsequently able to contribute to a successful launch, we can be confident that the automation can assure the success of future launches.

The core of our development software binaries consists of an automated testing framework and an open-source automation library. The automated testing framework has tabular syntax and utilizes low- to high-level keywords to operate automation features. This framework parses TSV data by first splitting all the content into rows and then each row into cells according to the arrangement of the tabular characters. The automation library is based on Python and automates anything appearing on a designated screen with the use of image recognition software to detect and manipulate GUI components such as are buttons, text fields, editable value fields, and dropdown menus.

II. Objectives

The main objective of this project was to automate testing of specific sections of the GSDO system graphical interface. These components requiring testing are ones that would use the streamed simulated data from the testing servers to produce measurements, plots, statuses, etc. to the GUI. These test steps are outlined in test case spreadsheets delimiting the step-by-step test procedures for each section that would normally be executed by engineers sitting in front of the monitor, manually operating the mouse and keyboard.

When we, the interns, could not work on our automated testing development in the LCC because of scheduling, we worked toward other minor but essential goals: to accommodate test cases that require communication between two or more workstations simultaneously, to extend the capability of the automation test framework to OCR, and manage the updating of SSH keys through a shell script.

[1] Launch Control System Software Development Intern, NE-ES, Kennedy Space Center, Rowan University

III. Approach

My approach to the intern-designated assignments was to first focus on gathering the necessary training and account access to work within the SCCS environment. Between training and account access grants, time was dedicated to familiarization with existing SCCS automation and quality tools. The regularly-scheduled meetings with technical mentors for this project and with the test automation working group kept all parties on schedule and prescient of one another's activities. After I properly familiarize myself with the automation and quality tools, including source control, automation software, OCR software, programming languages, and source file editors, I'd proceed to run tests, learning from trial and error until I have a thorough plan for the most optimal use of the tools and software I am using.

A. Training/Familiarization

a. Automation Testing (256)

Understanding the automated testing framework itself and libraries containing its native (i.e. low-level) keywords took about a week, whereas understanding the conventions and best practices specific to this automation team took decidedly longer. To learn more quickly, we were given an overview of the entire system by Jason Kapusta and of the firing room set configuration by Susan Pemble, a full-time engineer working on automation. Once our bird's-eye-view questions were answered, we partnered off into returning intern/new intern pairs to work on test cases together. Expanding my functional understanding of CLI environments such as vim and emacs improved my productivity as well.

b. Shell Scripting

The shell scripts were written in a CLI environment with a commonly used Linux terminal scripting language. Fellow intern Andrew Hwange wrote a script to set up SSH keys for the remote accessing of Linux boxes, and fellow intern Tom Plano wrote a semaphore script to manage control flow across several different boxes running sequenced steps from the same test case. These scripts were then reviewed by several interns, including myself, before being presented to management. Scripts were evaluated for effectiveness, consideration for boundary cases, and for being agnostic of context or test case.

B. Automated Testing for SCCS Dashboard

The first step to properly automating a test case is to break it town into test steps (see Table 1), and understand how to properly execute those steps by hand.

Step #	Operator Action	Expected Response
12	Launch display, "Some Display"	Verify "some Display" display appears
13	Execute the command script "Some Script", to Load data	Values on "Some Display" are changing

Table 1: Sample Test Steps from a Test Procedure Spreadsheet.

After understanding the purpose and sequence of micro-tasks appendant to each test step, the next step is to check the keyword files for high-level keywords specifically developed to the task. If a task is repetitive and involves the same micro-tasks, a new high-level keyword can be implemented.

a. Technical Specifics

This session saw sweeping changes to the higher-level keyword conventions, both with respect to implementation and organization. The common keywords file has been dropped in favor of a more object-oriented approach; keywords have been parameterized, given default parameters, and/or specialized to a certain repeatable common task and sequestered into the requisite page-specific file, termed a "page layout resource." These changes also included workarounds for the automation framework's lack of formalized conditional control flow.

In addition, verification of every step must be performed subsequent to and separately from every test step. This is to better serve the engineers vetting the test cases for correctness and effectiveness. Should a keyword fail, it is more apparent whether it was the functioning of the keyword itself or the effect of the keyword in that context.

b. Test File Formatting

The automated testing framework has a tabular-style syntax, meaning the functionality of a line of code depends upon the presence of the appropriate number of tab-separated phrases.

A given automated testing framework file is sectioned off into anywhere from one to four sections: settings, variables, test cases, and keywords. The settings section contains paths to resource files, suite setup either through a customized keyword or a library-determined one, or the names of necessary libraries. In the refactoring of resource files, the resource files included in settings were changed and tailored to the displays specifically involved by the test cases. The variables section contains 'global' variables strictly relevant to the automated testing framework file in which they live, and function globally for any test case within the file.

The test cases section sequentially runs the tests inside it, each identified with a title encasing one or more keyword calls. The keywords section holds locally-defined keywords that function like inner classes; the keywords themselves do not necessarily run in any order, but rather run as they are called by some test case from within the test case section or by a setting within the setting section. These locally defined keywords are only defined, and therefore only usable, within the file in which they live. Note that the variables and keywords sections can be left empty or completely absent if deemed unnecessary.

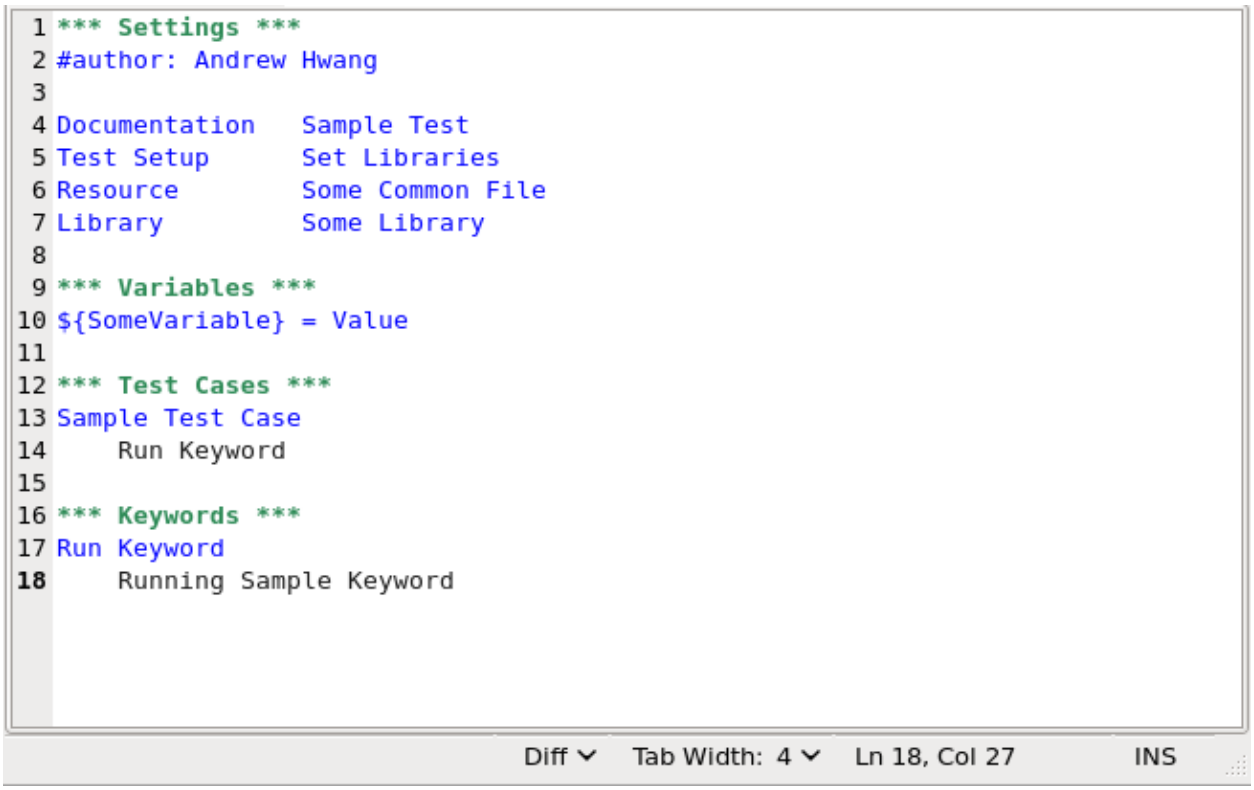

Figure 1: Format of Sample Automated Test File.
Picture credit: Andrew Hwang.

C. OCR Capability

As of now, the automated tests rely on image recognition capabilities with respect to the images that we have generated and screencaptured, but the need still exists for a means to "read" messages, statuses and warnings from the screen – a need OCR can address. This requires screen shots of the entire screen, which is far more automation-friendly than specifically capturing and creating images whenever something on the screen is updated. This will also make the control flow of the automated test cases more adaptable to the requirements of conditional programming by virtue of being able to account for changes in the graphical display.

The Project Lead of the Automated Testing Team, Jason Kapusta, assigned an appendant additional project of teaching the existing installation of OCR software to learn new fonts and layouts. The software's purpose is to read into a local variable the value or status of a given component and base future behavior of the test case around that value.

The OCR software binaries and compiler had already been set up last semester, so the previous setup was recovered. Box files were then generated and augmented as the software was run against different screencaptured display images.

D. Test Case Overlap Management Excel Macro

After previously completed test cases were mistakenly re-assigned several times over, I developed a VBA macro for our sprint management excel sheet that checks newly assigned test cases against those previously completed. The macro checks both the test category, number, and content to ascertain that work is not being assigned in duplicate.

E. Future Developments

The OCR capability still needs to be fine-tuned for complex images and subsequently integrated into the tests themselves. Further fine-tuning and refactoring of resource files will continue as the automation team progresses towards the Record & Retrieval test cases.

IV. Conclusion

As of the time of this paper, approximately 70% of all test cases have been automated. Still, the development team continues to test and refactor the GSDO system interface, so work remains to be done. The automation team made excellent progress in retooling their development framework to more effectively respond to these changes from the software development side. Among these changes to be completed are the fleshing-out of resource files and the integration of OCR software. While software development takes priority over automation when it comes to using the firing room set, all of these changes can receive the attention they need to benefit the development of a system that aims to send human beings to Mars.

Acknowledgements

I have a number of wonderful colleagues to thank for my progress over the course of this internship. I would like to first thank Caylyne Shelton for her graciousness and smiley disposition at each and every interaction, even when she had bronchitis and the flu at the same time, Oscar Brooks for the support and good humor he shows to all his interns, Jamie Szafran for her unflagging patience with the veritable fount of questions issuing forth from the dungeon each day, and Jason Kapusta for his extreme generosity with his wealth of technological information and relentless positivity. Lastly, I would like to thank all of the automated testing interns and full-time engineers: Michael Backus, Mark Rodriguez, Andrew Hwang, Tom Plano, Susan Pemble, Sam Bridges, Joanna Johnson, and Roger Zoerner, for being a paragon of cooperation, communication, and support.

References

N/A