Exploring the Use of a Test Automation Framework

Alex Cervantes
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109
818-354-0095
Alex.Z.Cervantes@jpl.nasa.gov

Abstract—It is known that software testers, more often than not, lack the time needed to fully test the delivered software product within the time period allotted to them. When problems in the implementation phase of a development project occur, it normally causes the software delivery date to slide. As a result, testers either need to work longer hours, or supplementary resources need to be added to the test team in order to meet aggressive test deadlines. One solution to this problem is to provide testers with a test automation framework to facilitate the development of automated test solutions.

The benefits of test automation are undoubtedly seen in big software projects. The development of automated test cases requires a lot of effort and time during the first test lifecycle, but there is a significant amount of time that can be saved with each repetition. Test automation gives a software tester the possibility of achieving unattended testing capability. With end-to-end test automation, a tester can schedule tests to run autonomously. While tests are running, testers can utilize the time saved by performing any requisite manual testing, or developing additional automated test cases to increase test coverage. Automated testing saves time as it provides the capability to perform concurrent/parallel testing. A tester can run multiple tests at the same time, whereas manual tests only allow for sequential test runs. If test automation is employed, testing does not have to end when the workday ends; automated testing can be exercised 24x7.

A test automation framework can provide numerous benefits to a software tester. A test automation framework provides the basic set of software tools and services that can aid testers as they develop automated test cases. With a test automation framework, software testers can focus on testing the software product instead of worrying about developing the infrastructure needed to support their test development environment. When choosing or developing a test automation framework, it is important to understand all the different components of an organization's software system. A good test automation framework should be general enough to apply to all the different components of the software system to be tested. The framework should also be easily extensible so the framework can evolve as the software system evolves. Whether creating one from scratch or using a commercial product, having a test automation framework available for test teams to use can help streamline the process of developing automated test solutions.[1][2]

TABLE OF CONTENTS

1. INTRODUCTION

In an effort to improve project system level testing practices within the ground data system(GDS) element at the Jet Propulsion Laboratory(JPL), a trade study of ways to improve test automation practices was conducted. One prospect that was identified as a candidate for improving current test automation practices is the use of a software test automation framework. The goal of a software test automation framework is to provide the infrastructure that a tester needs in order to facilitate the development of automated test solutions.

Looking for a candidate to evaluate the usefulness of a test automation framework was not easy. The framework evaluation team explored 3 options; developing one in-house, using a consulting company to recommend a commercial product, and performing a search on the web for a commercial product. After evaluating all available options, the evaluation team chose to use an open source test framework called Software Test Automation Framework (STAF). This product was identified as a general test framework that could fit within the system test environment. To evaluate the benefits that STAF advertised, a system level test involving two JPL software systems was chosen.; the Mission Data Processing and Control Subsystem(MPCS) and the Multi-mission Automated Task Invocation Subsystem(MATIS). These two software products work together as a system within JPL's GDS.

The STAF framework provides a collection of general test services that testers can use to develop their automated test

1

[1] "U.S. Government work not protected by U.S. copyright."

[2] IEEEAC paper #1477, Version 1, Updated October 31, 2008

solutions. These services provide software test functions that are part of the infrastructure testers use to facilitate automated test case development. STAF also provides the capability to extend beyond its core set of services by allowing testers to develop custom services specific to their testing environment.

The STAF Execution Engine (STAX) is an additional component of STAF which serves as a programming language designed for test automation. STAX allows a tester to develop their test execution workflow. STAX provides the means by which a tester can distribute test data, configure the test environment, execute the test, and analyze the test results. STAX also has an optional GUI which lets a tester conduct, monitor, and interact with test activities.

This paper discusses the evaluation team's observations of utilizing STAF and STAX to support the implementation of an automated system level test between MPCS and MATIS. Described in this document is:

(1) An overview of the MPCS and MATIS software systems

(2) A description of STAF and STAX

(3) An explanation of the approach and design of the system test

(4) A description of the test implementation

(5) The overall impression perceived by the evaluation team

(6) What is needed to incorporate STAF into JPL's GDS system test environment

2. SYSTEM TEST DESCRIPTION

The system test that was chosen to evaluate the effectiveness of using STAF within the GDS system test domain consists of two software systems named MPCS and MATIS. The goal of this test was to validate the interface between these two software products. The communications interface between MPCS and MATIS is done over the network through the Java Message Service (JMS) bus. (See Figure 1.)

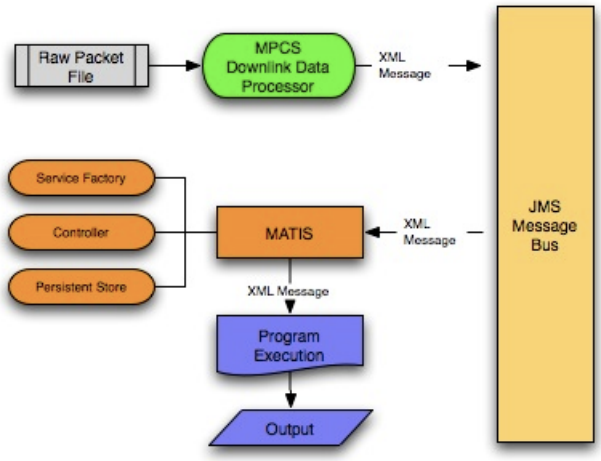

Fig. 1. System Test Overview

MPCS is a collection of programs that provide functions such as frame processing, data product creation, telemetry viewing, and additional GDS functions. The specific piece of software used in this test is called chill_down, which is the MPCS Downlink Data Processor. For this system test, chill_down was configured to take a raw packet file as its input, and process it to create raw data products. For each raw data product that MPCS generates, an XML message containing metadata about the data product is sent to the JMS bus. Any client listening to the topic that chill_down is publishing to will receive these messages. For this test, the client configured to listen to the chill_down topic is MATIS.

MATIS is a distributed data processing framework for the automated generation of instrument products. It is a workflow manager that executes programs in a specific order and under specific conditions. MATIS was installed in the testbed on a machine running the Solaris 9 operating system. Running MATIS requires that a MySQL database is configured and running, followed by three command line programs that must be executed in sequential order. These programs are the controller, persistent store, and the service factory. The version of MATIS that was tested used XML messages on the JMS message bus as the specific condition for triggering the execution of a specific program. In this case MATIS listened to the JMS topic that chill_down was publishing to. When the MATIS message listener receives the XML product message on the JMS bus, MATIS performs the following steps:

(1) Parses the XML message to determine what type of product was produced.

(2) Determines which program to execute based on the product type.

(3) Executes the program with the XML message as a program argument.

The end of the test is reached when all the programs invoked by MATIS have completed their execution.

3. EXPLORING STAF/STAX

The STAF test framework consists of 3 major components: STAF services, the STAF daemon, and the STAF API. STAF also has an optional component called STAX, which can be installed as an external service to STAF. STAX is a programming language that was designed for the purpose of testing. STAF and STAX provide general test capabilities to facilitate the development of end-to-end automated tests.

STAF Services - The core functionality of STAF is in its services. STAF services are a set of general functions that testers can use to facilitate the process of creating automated test cases. STAF services can be compared to built-in objects that are found in most modern programming languages such as C++ and JAVA. Built-in objects provide common methods that programmers can utilize to develop software applications. For example, the JAVA programming language contains built-in object libraries such as Swing for developing GUIs, a string object for parsing and manipulating strings, and a math object for performing basic mathematical computations. Object libraries such as these make up the infrastructure that software engineers depend on to develop their software applications. Without this infrastructure, development time for software projects would be prolonged.

STAF services follow the same concept as built-in software objects, but are focused specifically for testers developing automated test cases. Examples from the available set of STAF services which testers may find useful are the LOG service, VAR service, and the QUEUE service. The LOG service provides set of standard log operations that allow a tester to write data to a log file, query from a log file, and delete entries from a log file. The VAR service gives the tester the capability to manage variable pools within the test environment. This service gives testers the capability to set, get, and delete variables that can be seen by machines connected to the test environment. The QUEUE service can be used to simplify inter-process communication. For example, a tester can use the STAF QUEUE service to design a process workflow where process B only starts when a certain condition in process A transpires. This can be accomplished by running process A and sending its output to the STAF queue. Process B can be setup to listen to the specified queue for a specific output from process A. Detection of the output would then trigger process B to execute. These three services are examples from the STAF service set that can aid testers when developing automated test cases.

STAF allows testers to extend its built-in services by providing a template for developing custom services. STAF allows testers and developers to create custom services that can be "plugged" into the STAF framework as an external service for use by the rest of the test team. This is important because the default set of STAF services are meant to support general testing. If the testing environment requires specific functionality, testers can develop their own STAF service in C, JAVA, or Perl. STAF provides a custom service template for each language that testers can use to get started. For example, if the majority of software that is being tested uses the JMS technology, a custom service that listens to a JMS topic, publishes to a JMS topic, or executes a process when a specific message is seen on the JMS bus can be developed using the JAVA STAF template. Custom services installed into the framework are external services that reside outside the core of STAF. This architecture allows for the installation of newer versions of STAF without worrying about merging your previous installation of STAF with the new one. It is as simple as copying your external services into the new installation of STAF.

STAF Daemon – The STAF daemon allows for the distribution of STAF services to STAF enabled machines on a network. The daemon is a process that runs on each machine waiting for a STAF service request from a local or remote host. When the daemon receives a request, it parses the STAF string to and performs the request on the local host it is running on. This capability allows testers to manage test processes running on heterogeneous machines in the test environment. (See Figure 2.)

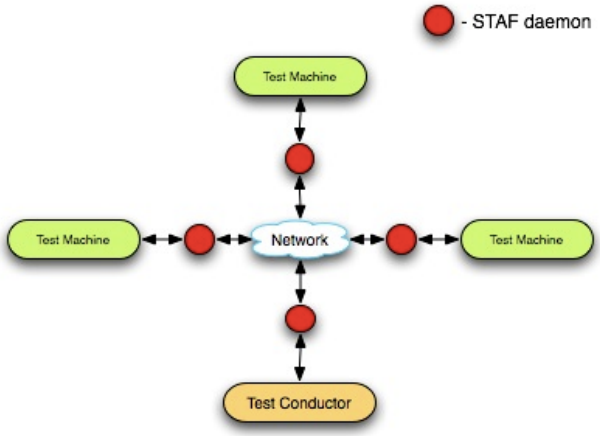

Fig. 2. Test environment communication through STAF daemon

STAF API – The STAF API is the development interface to the STAF services. It supports the development of test cases in popular programming languages such as shell, perl, python, C++, Java, and others. The variety of language support allows testers to write their tests in the language they are comfortable with. You can have one tester programming in perl, another tester programming in python, and both can have full access to all the STAF services.

STAF also has an optional component called STAX, which is a programming language designed specifically for testing. The STAX programming language is comprised of three technologies; XML, Python, and STAF. Testers can

program "STAX jobs" that perform functions such as distributing test data, configuring the test, and executing test cases. STAX also provides a GUI tool which allows a testers to visually monitor and interact with their STAX jobs.

STAX

STAX provides the major features found in modern programming languages that are useful for writing test programs. These features include the use of variables, functions, conditions statements, loops, and others. Python is integrated with STAX giving powerful programming control to the tester. The ability to use Python gives testers great flexibility because of the numerous built-in libraries that come with Python, and the open source libraries that are available on the Internet.

STAX contains a number of key functionalities that are specific to testing. One key feature is the ability to concurrently distribute a STAF service onto multiple test machines. For example, lets say a tester wants to run a STAF service to execute a command which records system performance data on 100 test machines. You can use perl and the STAF API to do this by looping through a list of machines and executing the performance capture software on each machine in the list. However, there will be a latency time difference between each execution because the iteration through the list is done in sequence. This can cause the start time of the recorded data on the last item the list to be significantly later than the start time of the first item in the list. In this type of situation, it may be more beneficial to use STAX's parallel iteration functionality to loop through a list of machines and execute the performance capture software on the test machines at approximately the same time.

Another STAX function that is useful in managing processes distributed across multiple machines is the "block" wrapper. Blocks give testers the capability to manage and control groups of test processes. Working off the previous example, lets say a tester wraps the process calls running on the 100 machines calls into a block. When it is time to kill those processes, you can write a single line of code to terminate the block, and all the processes running on the 100 machines will die. Other useful functions are the hold and release capabilities. These can be effective for managing process flow in a STAX job, or when you want to manage process execution through the STAX GUI. Two other valuable STAX wrappers are the test case wrapper, which helps a tester visualize testcase pass/fail status on the STAX GUI, and a timer wrapper, which is used for timing process execution times.

The STAX GUI tool gives the test conductor useful functions that facilitate the management, interaction, and visualization of STAX jobs. Management functions include the ability to load, validate, and run STAX jobs through the GUI. A tester can interact with STAX jobs by passing parameter values to the job before it is submitted for execution. This is similar to passing an argument to a command line program. Testers can also interact with tasks within the STAX job, such as terminating, holding, or releasing blocks. Visualization capabilities allow the test conductor to see all STAX jobs running, and the state of each STAX job. For more detail on a STAX job, the tester may open a STAX job to see the status of test case results, logs, messages, etc. STAX jobs can be executed as well on the command line, but the GUI is a good way to see a snapshot of the state of all tests.

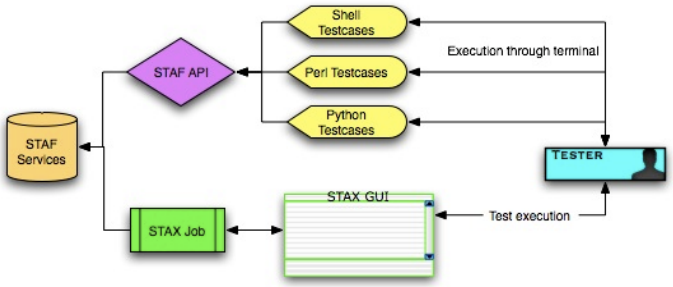

Fig. 3. STAF and STAX used in the test environment

4. Implementation Approach and Design

The purpose of the system test between MPCS and MATIS was to evaluate the use of STAF as a test automation framework for JPL GDS system level test teams. For this reason, the approach for this evaluation was not to force the use of the STAF framework wherever possible, but instead to use STAF only where it made sense. The list below describes the steps taken to automate this system test.

(1) Create procedures for the system test

(2) Test the procedure by running the test by hand

(3) Translate the procedures into pseudocode

(4) Obtain a general overview of all available STAF services

(5) Analyze the pseudocode and use STAF services where it makes sense

Implementation of the pseudocode was done using STAX, Perl with the STAF API, and Python without the STAF API. This approach gave the evaluation team a good understanding of the framework so they could later discuss the benefits and difficulties they encountered. Between the three methods the primary focus was given to the STAX implementation because of its test-centric capabilities. Implementation for all three approaches was performed on a Mac.

The goal of the evaluation team was to automate most of the

pseudocode with the goal of having the test fully automated. After looking at the STAF services available, they found that it was possible to automate the entire test and not need the interaction of a tester to do manual testing. The automated test design consisted of the following:

(1) Delivering configuration files from the test conductor machine to the MPCS and MATIS machines.

(2) Creating a temporary directory on the test machines for test results collection.

(3) Starting up the MySQL database (Needed by MATIS).

(4) Starting up MATIS processes.

(5) Starting chill_down with the input file to initiate the test.

(6) Timing the test.

(7) Shutting down MATIS processes and MYSQL upon completion of the test.

(8) Analyzing the test data collected on the MPCS and MATIS test machines.

(9) Publishing the test results on the tests results database website.

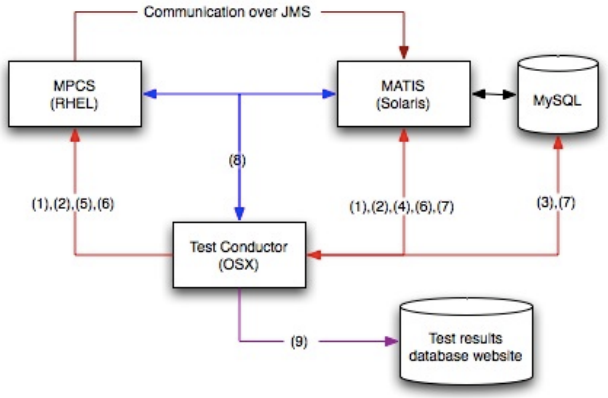

Fig. 4. Test Automation Workflow

5. IMPLEMENTATION OBSERVATIONS

The implementation of the automated system test was quick with the exception of the time it took get over the STAF and STAX learning curve. Just like any other piece of software, time is needed to learn how to install, configure, and use STAF and STAX.

Using STAF

Configuring and installing STAF on the test machines was clear-cut. STAF was up and running on all the test machines within a few hours, and sample STAF service calls were tested shortly after on the unix terminal as a shell command. The syntax for calling STAF services is well documented in the STAF user's guide and easy to understand. In addition to this, each STAF service provides a built in "help" option that describes how to use the service. This is similar to a man page for unix commands.

Using STAX

Configuring and installing STAX as an external service to STAF was also explicit. Like STAF, STAX also has a comprehensive user's manual that explains in detail STAX's usage. Within one hour, STAX was installed on OSX running on a Mac laptop. The STAX GUI worked on the first try, and a sample STAX job XML file was loaded and executed through the GUI.

Learning and getting used to the STAX programming language took some getting used to. It took one week before the evaluation team became comfortable with writing STAX code. The idea of writing a program in XML was awkward at first for the evalutation team. Some of the frustrations that were perceived included:

(1) A lot of code is needed to do simple things. This is mostly due to the nature of XML.

(2) Editing XML code can be frustrating because of the constant necessity to adjust the indentation of XML code so that it is easy to read.

(3) Quoting in STAX be confusing and convoluted at times, especially when wrapping complex python quoted strings inside XML specific quoting syntax.

(4) The utilization of Python with the XML syntax takes some getting used to. For example, arrays are declared in Python syntax, but iterating through the array requires STAX specific syntax. Getting used to this takes some time, but becomes natural over time.

(5) Since syntax is in XML, you cannot nest comments which can be a hassle when debugging programs.

Some of these frustrations could have been relieved with the use of a text editor specific for XML, but the evaluation team did not have one and used regular text editors. Although it took some time for the evaluation team to adapt to using STAX to write the test case, the end result was positive.

Using STAF and STAX to implement the automated testcase showcased the benefits of using a test automation framework. The biggest benefit was that there was no need to develop test infrastructure to support the implementation of the automated test. There seemed to be a STAF service or STAX capability to facilitate the development of every part of the test. The major features that simplified the test development include:

(1) STAF daemon – provided the capability to distribute STAF services between test machines. This made it possible to run everything from the test conductor machine (Mac).

(2) File System (FS) STAF service – This service was used to distribute configuration files and collect test results to and from the MPCS and MATIS test machines.

(3) Process STAF service – This service allowed processes such as MySQL, the MATIS components, and MPCS to be initiated from the test conductor machine, but run like it was started from the resident host machine.

(4) STAF logging service – Took care of managing test log operations.

(5) STAF global variables – Allowed all the test machines to define and share a common variable set for easy communication between the machines.

(6) STAF HTTP service – This was used to autonomously publish a test report to the test results database.

(7) STAX GUI – This feature was very helpful to visualize the test processes on a single page on the GUI.

(8) Testcase STAX wrapper – This wrapper allows the tester to define individual test cases with a STAX job. The visualization of the pass/fail status is shown on the GUI for convenience.

(9) Embedded Python support – This was used analyze the test results and create the test report.

Using Perl and the STAF API

Using Perl with the STAF API to access STAF services certainly simplified the development of the test case. The STAX visualization features and wrappers were missed, but not necessary to automate the test case.

Without a test automation framework, a lot of infrastructure code would need to be developed before test case development could start. For example, In place of the daemon, methods to carry out the remote execution of tasks would need to be created using technologies such as Secure Shell(SSH) and remote procedure calls(RPC). Development time for this would most likely prolong the overall development time of the test case because the infrastructure software, like all software, would have to be tested on its own. Given this, testers would be testing software that is used to test their software, causing the inefficient use of a testers time. This exact inefficiency was experienced when implementing the test case in Python without the use of a test automation framework.

Using Python as a test automation framework

To better study the use of a test automation framework as a means of improving testing practices, a case study involving the implementation of the same task without STAF had to be exercised. This was accomplished by using only the Python programming language to replicate what was previously done with STAF. This task proved that infrastructure development can occupy a large amount of a tester's time. The inefficiency of developing everything from scratch was observed early on, so the full completion of this task was never reached. The amount of time and work it took to replicate STAF's abilities of distributing tasks and threading processes resulted in the early realization of the benefits of using a test automation framework.

6. OVERALL IMPRESSION

The overall impression of test automations frameworks, specifically STAF/STAX, was positive. The evaluation team felt that having a test automation framework could benefit GDS system-level testers. The GDS is a complex system comprised of a heterogonous mix of operating systems and programming languages thus making system-level test automation difficult. It is also common that members within a test team have different skill sets and methods when it comes to testing. There are test automation power users who try to automate everything, manual testers, and a testers who dabble in both. Within the group that does practice test automation, it is usually the case that all the testers do not use the same programming language to do their test automation development. Because of these factors, STAF's flexibility to support numerous operating systems and programming languages was the driving factor for recommending the STAF test automation framework to GDS system-level testers.

STAF's modularity and expandability is another important facet of the framework. Software systems, over its lifetime, expand and evolve. When this happens it is important that test functionalities provided by the framework can grow just as software capabilities grow. The modular structure of STAF and STAX both provide templates that testers can use to extend the functionality of the framework.

STAF's and STAX's general capabilities were mostly useful and easy to use. Overall, the general consensus was that STAF and STAX made complex testing tasks simple. The evaluation team tested most of the STAF/STAX capabilities that were not used in the MPCS/MATIS system-level test, and most of them seemed like it could simplify the different areas of system-level testing.

The fact that STAF is open source is both good and bad. The biggest benefit is that the testers don't have to worry about maintenance of the framework, which can be costly if it were done in-house. On the flip side, if bugs are found in the framework, you are at the mercy of the framework

developers to have them fixed. In the worst case scenario, the test team could be heavily invested in the framework only to have the support for the project stop. When evaluating the possibility of this occurring, it is important to look at the size of the open source community behind the project, and the development history of the open source project. In this case, STAF has a striving community, and has been in active development since 2001 with routine updates to the software.

Although STAF/STAX could work well within the JPL system-level testing environment, it may not be suitable for all software testing scenarios. There is definite overhead associated with utilizing a test framework. Testers must invest a considerable amount of time to learn how to use and adapt to using a test automation framework. Also, since test automation is a development project, testers need a considerable amount of knowledge in the area of software engineering. For these reasons, a test automation framework may not be suitable for small software projects with small team of developers and testers. In this case, the use of simple scripting may be more efficient than utilizing a test framework. A test automation framework is best suited for large and long term projects where the cost of the overhead associated with framework is relatively small when compared to the cost of the software project.

Implementing STAF/STAX into the GDS system test team

Before committing to using the STAF test automation framework, a few important tasks need to be addressed. The first task is to have the entire test team try out the STAF framework. To facilitate this process, it is a good idea to have a "STAF maintainer" that can teach and assist the test team as they learn to use the framework. It is important for the maintainer to be attentive to the test team's suggestions and complaints because those will be the initial drivers for custom STAF service development. The coordination of custom service development is another job of the maintainer. The maintainer should work with developers to encourage them to write custom services to meet the needs of testers. As custom services are developed, it is the maintainer's job to install the custom service into the framework for immediate use by the testers. Having a dedicated STAF maintainer available can be very useful in the early stages of adopting STAF as the team's test automation framework.

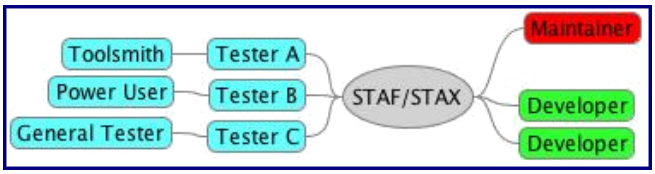

Fig. 5. Role of the STAF/STAX maintainer

7. CONCLUSION

As technology continues to grow and become more complex, software testers will be faced with tougher challenges to fully test the software product within the time given to them. To keep up with this trend, testers must consistently look for ways to improve their testing practices. A test automation framework is a tool that can help a tester efficiently develop end-to-end automated test solutions. Utilizing a test automation framework is be a big investment up front, but it can lead to a net cost savings in the future. Determining which test automation framework to use is not a trivial process. So before your organization decides to commit to using a test automation framework, it would be wise to first explore its use within your testing environment to ensure that it is suitable to your software testing needs.

ACKNOWLEDGEMENT

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

[1] Software Testing Automation Framework(STAF) Website http://staf.sourceforge.net

[2] "STAF V3 User's Guide," Software Testing Automation Framework (STAF) Website http://staf.sourceforge.net/current/STAFUG.htm, Feb 26, 2008.

[3] "STAX Service User's Guide," Software Testing Automation Framework(STAF) Website http://staf.sourceforge.net/current/STAX/staxug.html, Sep. 26, 2008

[4] "STAF Service Developer's Guide," Software Testing Automation Framework (STAF) Website http://staf.sourceforge.net/current/STAFUG.htm, Dec. 14, 2007

[5] "STAX Extensions Developer's Guide," Software Testing Automation Framework (STAF) Website http://staf.sourceforge.net/current/STAFUG.htm, Oct. 3, 2007

[6] Andrew W. Bingham, "Mars Science Laboratory MS Functional Design Document (FDD), JPL Docushare, Feb. 7, 2008

[7] Jake Engleman, "Exploring a Test Automation Framework: Using STAF/STAX to Perform a System-Level Test of MPCS and Matis", File transfer from author, Aug. 5, 2008

[8] Jake Engleman and Alex Cervantes, "Exploring a Test Automation Framework", Group Presentation at JPL, Sep. 4, 2008

[9] Cecilia Cheng, Rajesh Patel, Elias Sayfi, and Hyun Lee, "Multi-mission Automated Instrument Product Generation Implemented Capabilities", File transfer from author, Oct. 21, 2008

[10] "Java 2 Platform Standard Edition 5.0 API Specification", Sun Microsystems Website http://java.sun.com/j2se/1.5.0/docs/api

[11] IBM Software Group, "Hands-on Automation with STAF/STAX", Software Testing Automation Framework (STAF) Website http://staf.sourceforge.net/education

[12] Alex Cervantes, "Test Automation Proposal", Group presentation, Jun. 18, 2007

BIOGRAPHY

***Alex Cervantes**, Project Integration, Test, & Deployment Engineer, Jet Propulsion Laboratory, Pasadena, California.*

Alex is currently working on MSL project as an Integration, Test, and Deployment engineer as part of the GDS team. He is also working on integrating the STAF test automation framework into the Project Integration, Test, & Development group at JPL. He received his B.S. in Computer Science from UC Riverside in 2002.