WHITEPAPER –

GREENLIFF'S AUTOMATED TESTING FRAMEWORK SOLUTION

TECHNOPARKSTRASSE 1

8005 ZÜRICH

SWITZERLAND

PHONE: +41.43.20 40 800

FAX: +41.43.20 40 808

info@greenliff.com

www.greenliff.com

WHAT IS TESTING?

A software company's survival in the rapidly changing and competitive field of application development depends on the soundness of its product. Consumer confidence can drastically be increased because of new test methods and approaches. However, if test methods and framework implementation are not considered carefully, it can easily detract from a company's profits. Test design strategies need to have precise and accurate outcomes to prove any value for the money being spent. A test framework used uniformly will successfully streamline test activities across multiple and significantly different environments. This paper will introduce a test framework specifically used for automated software test environments.

AUTOMATED TESTING

The most basic description of automated testing states: "Automated testing is where technology is used to replace testing workflows once done by humans." There have been many books and articles written, tools developed and web sites dedicated to the topic of "Automation of Software Testing". Some sources suggest that automation of Unit testing (prior to building) realizes multiple benefits of early stage bug detection and resolution. Other sources say automated integration-level smoke tests concurrently decreases developer time spent and margin for error. Unanimously, resource after resource, there is always one type of test that is fully agreed as the most advantageous to execute using automation. This is the "Regression Test".

REGRESSION TESTING

Regression tests can be automated quite successfully and are easy to maintain. Checking functionalities that have already been tested is what regression testing is. Due to ongoing development and source code changes, the newly compiled application must be validated for all its main functionalities on each release. Also certain bug fixes justify regression tests using an automation framework.

Most often, manipulated code in one area of the application may inadvertently affect other non-changed areas. This can result in negative reactions to the overall software functionality. In some cases, a

functionality that was already developed, coded and tested successfully shows up as a failure in a regression test. The overall purpose is to catch problems that may not have been discovered after even the slightest source code changes. This also means having to rerun previously passed tests along with the known failed tests.

Regression testing can also occur in situations where there have been zero changes to the source code, but its operating Environment has been changed. This too, would entail regression testing to achieve the overall purpose and speed.

Lastly, a hidden benefit of regression testing is also the simple fact that if a developer knows there is an existing regression test plan, he feels more confident in making his modifications to the source code. He is more comfortable knowing that possible errors will show up in regression tests, thus achieving more cutting-edge functional software.

DATA DRIVEN

Automated software regression tests that are executed repeatedly with different input data often use a "data-driven" test approach. "Data-driven" testing utilizes input data that resides "outside" of test scripts in a separate database. This approach is tremendously efficient when comparing maintenance efforts. Any script modifications will automatically carry over to every test case in the spreadsheet. On the flip-side, scripts that include the input data "inside" each test case entail repeated modifications for every test case. This can be extremely time consuming if there is a large amount of input data.

Consider a spreadsheet of possibilities, and combinations of possibilities, for input in a software application field. This can be used to validate the software's expected behavior when using every row of input data.

	Possible Input 1	Possible Input 2	Possible Input 3
Test Case 1	x	y	z
Test Case 2	y	z	x
Test case 3	z	x	y

A data-driven approach to testing would be a perfect way to repeat the tests with a different row of data each time.

GREENLIFF AUTOMATED FRAMEWORK SOLUTION

Geenliff is the company with the Framework for Automated Software Testing ("FAST") that proves advantageous with streamlining automated testing. The most successful approach to automated software testing should be of data-driven and framework-based design which is the foundation of Greenliff. FAST takes the idea of execution and management of automated test cases that run on many different sub-systems and brings them together into one centrally managed infrastructure.

FAST has an organized method of displaying and storing result data from existing legacy or new test systems. The implementation utilizes various open-source, third party automation tools and scripting/programming languages. Nightly build(s), test authoring, bug tracking and metric reporting systems are integrated to have one clear "test results" oriented user interface. Most test environments execute scripts on different sub-systems scattered along the network. The reason they have different sub-systems is due to the numerous phases of development, unrelated test objectives, varying test system requirements and diverse data types and output files. FAST is aware of this and can manage your overall test environment.

The framework consists of variations on the following core components:

- **Test Engines; automated testing sub-system.**
- **Job Manager; job distribution to different test engines.**
- **Test Job; job submission XML interface for initiating test execution.**
- **Execution Monitor; monitoring the test process**
- **Collector; test results database**
- **Results Application; results user interface**
- **Reporting Tool; unified result reporting**

Using the component flow diagram in Figure 1 you can identify all the FAST components in blue.

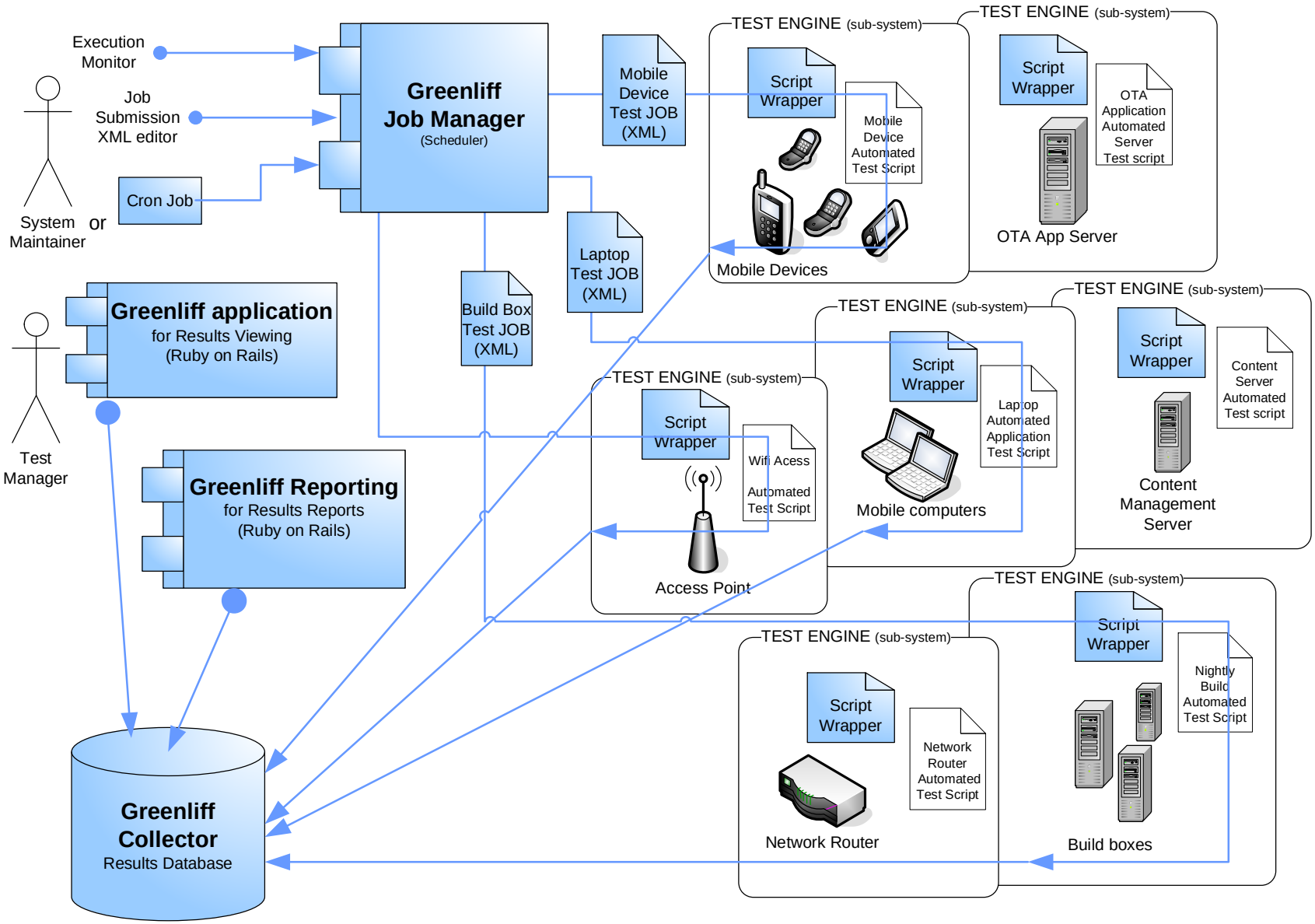

Figure 1 FAST Component Diagram

TEST ENGINES

Automated sub-systems (3rd party automation applications or script files) that actually perform the automated test execution are the FAST "Test Engines". The engines usually require unique environments in order to run test scripts.

JOB MANAGER

The FAST "Job Manager" component knows the location on the network of all the separate "Test Engines" and can siphon off jobs to the appropriate engine. It understands that scripts which require MS Windows with a Perl interpreter go to Engine A, scripts that require a Unix environment go to Engine B and Test Director macro tests go to Engine C. The Test Job Manager also takes care of test progress monitoring and communication with the test execution scripts.

TEST JOB

The FAST solution commences with a "Test Job" submission using either an HTTP XML Editor or through custom cron jobs. The Test Job is an XML file with test case dependencies (prerequisites that need to be fulfilled prior to running the current job), test engine name, and executable command (usually a call to launch the actual automated test script file).

EXECUTION MONITOR

Test processes can be monitored on the sub-system of the test engines once the Test job is submitted. It shows what tests have been completed and what tests are still in progress. When the test process finishes (or doesn't finish) it receives a result code of either PASS or FAIL.

TEST SCRIPT

The script file is using a software wrapper to talk the common language "whatever that may be" of the Test Engine to start running the automated test.

COLLECTOR

The results are collected in a relational database. This is achieved by parsing the output and mapping it into a standardized schema.

RESULTS APPLICATION

Testing results, as stored by the 'Collector', are displayed through the 'Results Application'. It can be intricately detailed to display specific error details, have hyperlinks to bug tracking systems, be summarized in test data comparison charts and decision tree displays.

REPORTING TOOL

The reporting tool provides unified reporting including charts and diagrams of the testing results.

BENEFITS OF GREENLIFF'S AUTOMATED FRAMEWORK SOLUTION

Companies can get a "jump-start" at establishing a proven framework for automation testing early on in the software development process. Greenliff and FAST gets you the most productivity out of (rarely used) automation tools that your company has already purchased. Since the approach is interoperable and scalable, it makes it adaptable to almost every software development environment. Test Managers are getting "Full automation" and immediately reap the rewards of "results capturing and comparing" with FAST. The historical data is stored in the database so test reports can be quickly retrieved at any point in time without a whole lot of effort. It also provides data for Software Managers to determine the current status of their software development efforts.

The framework has some specific feature requirements in order to get successfully implemented. Your test environment usually has some types of existing or partially existing automated test systems in some form or another. This could mean a half written Perl script to automate a checkout process or a complicated end-to-end test of a GUI application using a 3rd party tool. Let Greenliff consultants introduce you to test automation and let us drive your automated test pilot plan. Our development services can help you create test jobs in script languages such as Perl, Python, Ruby, batch files and UNIX/Linux shell scripts. The Greenliff FAST approach utilizes open source to create customizable scripts, a platform-independent repository for storing data, a Test Manager and an application for results viewing.

CONCLUSION

Greenliff can provide you consultancy to commence your testing automation and follow up with the implementation of the FAST system to bring your testing environment up to speed.

BIBLIOGRAPHY

Beck, Kent. "JUnit: general Java Testing." Tomcat Book: Chapter on Testing. Chapter 1. Testing Frameworks. Approximity. <http://approximity.com/testing/book1.html>.

Fornander, Anders and Nicolas Wettstein. "eQATor: Testing Java phones all day and all night." Java User Group Switzerland (JUGS). 27 Jan. 2005. Geenliff AG.

Kaner, Cem Ph.D. "Improving the Maintainability of Automated Test Suites [1]." Los Altos Workshop on Software Testing (LAWST). Feb. 2006. Quality Week '97. <http://www.kaner.com/lawst1.htm>

Mosley, Daniel J. and Bruce A. Posey. Just Enough Software Test Automation. New Jersey: Prentice Hall PTR, 2002.

Pettichord, Bret. "Five Ways to Think about Black Box Testing." StickyMinds.com Weekly Column. 5 Nov. 2001. http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=2968

Pilz, Markus. "Automated Functional Testing." Tekzone Forum 061. 27 Feb. 2006. Geenliff AG. http://tekzone.netcetera.ch/downloads/index.html