Automating Regression Testing for Command-Line Applications: RegressionTest 2.0

Sergei Sokolov
Manager, Professional Services
This white paper presents an overview of a new highly-scalable environment for regression testing command-line applications—an environment based on Parasoft Test Center. Typically, a regression test environment for an application is realized either via a complicated set of shell, Perl, or other scripts written by application developers or QA personnel, or using heavy-duty commercial test management systems, such as HP Quality Center. Both approaches have their advantages and disadvantages. Although a regression script system is “free” and does the job for a given application, it typically does not scale: it is difficult to maintain and extend for a different application. Commercial systems are expensive and generally are exposed only to QA teams, leaving development teams to their own devices for their regression testing.

Parasoft Test Center addresses both audiences and enables teams to set up a truly transparent and easy-to-maintain regression test system. It combines an intuitive easy-to-use graphical environment for test specification and execution with a plethora of built-in automated tools for processing test results in different formats, Java and script extension points, a deployment-ready reporting system, and automatic connections to project and quality management systems such as Quality Center and Parasoft Concerto. If you are interested, read on!

The Gist of Regression Testing

Regression testing of software is as old as software development itself. While software has evolved by orders of magnitude of capability and complexity over decades, the premise and goal of regression testing remain the same: verify that the software functions today just as it did yesterday. In other words, ensure that it did not regress.

There are many ways to assess the state of regression, but the most common is to create or specify some tests that can be run dynamically and execute them against the software system. Thus, typically, regression testing amounts to:

1. Executing a piece of software with a specified test or a use case (intended or erroneous use).
2. Capturing the result of the execution.
3. Validating the result as correct by manual inspection, computer-assisted checks, or other means. This first-time result is then saved and labeled as “expected” or “golden.”
4. Re-running the available tests over time—most critically, after the software or its platform has been modified—to ensure that the application still produces the expected results.
5. Rectifying any reported test differences:
   a. If the new result is incorrect, regression testing detected a bug. Addressing this involves fixing the software and rerunning the tests.
   b. If the new results were expected as a consequence of updated / changed functionality, this involves updating the test.

The simplest (though least effective) approach to regression test involves manually running the application under tests, reviewing the outputs, and addressing test differences. Admittedly, there are still some systems that may require such an effort-intensive approach to validation. However, for the most part, set up and execution of regression testing has been automated. Common approaches to automation range from custom-written shell and Perl scripts and/or make to commercial frameworks (e.g. HP Quality Center for enterprise software, Wind River Test Management for embedded systems, etc.). These tools typically handle all regression test execution steps (except for #3—first result validation), produce and distribute reports, and in some cases even automatically update the tests when the system under test changes.

However, automation options are more limited for teams developing applications that have only a command-line interface. In addition to writing scripts (e.g., Perl or shell scripts), it is also
necessary to construct scaffolding, such as external test drivers, results comparators, and report generators—which all need to be maintained and extended as the application evolves. Thus, it is not surprising that Parasoft has seen that a surprisingly large number of command-line applications are still developed without a robust and extensible regression framework.

This paper presents a new approach to regression testing command-line applications: an approach that uses the graphical environment of Parasoft Test Center to fully automate the core regression testing process.

Regression Testing with Parasoft Test Center

Parasoft Test Center provides advanced test capabilities that significantly impact testing command-line applications:

- Easy, intuitive interface for setting up tests—equally suits developers and QA personnel
- “Execute anywhere” design—transarently supports developer, integration, and QA test activities

The major productivity difference gained through using Parasoft Test Center for application regression testing is that the complete test setup is managed via GUI wizards, in a 4G style. Whether defining single-run tests or complicated scenarios involving multiple components or pre-conditions, the test structures are very similar. There is no need to write or maintain any external test drivers, results comparators, or report generators. Once the tests are set up, they can be executed interactively, from a batch script, or using an exposed web service interface.

In addition, Test Center provides test traceability and a host of additional capabilities that can assist with testing related applications or application components—for instance, command-line executables, web applications, web services, database components, CORBA applications, etc.

Introducing the Sample Test Scenario

We will now walk you through the steps of setting up sample tests for a program with a command-line interface. “cpptestcli” is a shell command-line program that performs analysis of C/C++ source code and generates reports. From the input/output point of view, the cpptestcli application is rather simple, as illustrated in the figure below. It takes source code, configuration files, and a test script as inputs, and produces HTML and XML reports, as well as puts out a log on stdout console. This is a fairly standard scenario for any command-line program, including embedded applications.
Performing regression testing on this application requires a means to:

- Provide or vary the desired inputs
- Start program execution
- Capture the exit status of the program
- Perform some meaningful analysis or comparisons of the outputs, such as the console log and the XML report

This is exactly the type of test structure we will establish with Test Center, which will provide us with control of inputs as parameters to the test, an execution trigger, and automatic regression checks on outputs.
Defining a Test that Invokes the Command-Line Program

First, we create an empty test project (Regression Test 1) in the Test Case Explorer Wizard:

![Create a SOA test project](image1)

Next, we add a new test to the project’s default test suite (note that a test project may contain multiple test suites). We use a built-in “External Tool”, which is designed to invoke executables.

![Add Test](image2)

The External Tool provides a means to invoke a third-party executable from within SOATest. Then, the output of that external utility can be returned to indicate success or to parse for values to use in later tests. The executable used must terminate on its own and must not require any user input, but it is possible to invoke the executable with command line arguments. Also see the Extension Tool for customizing tests.
Double-clicking on the created test opens up a configuration panel for the new command-line test. Here, we specify the name of the executable, command-line flags, and additional test options. Note that the flags for a given test, which are specified via an Add dialog, can be fixed values or parameterized via a data source.

**Capturing and Validating the Program’s Output**

The simplest arrangement of a command-line regression test is to capture the stdout output of the program run and create a regression control with it. This way, the next time the test is run, the program’s actual output will be compared with the previously captured “expected” output.

To facilitate this, Test Center offers multiple options to process such output, starting with a simple Diff tool (which will perform a Unix-like diff), to various other tools that can post-process the stdout output to strip out unnecessary text and then compare.
As you can see, the output of cpptestcli was captured as a regression control. If we needed to compare just a few specific lines, we’d need to pass this output through some filter tool, and then attach a regression control to it.

We can also create a regression test based on an output file of the program under test—either directly comparing it to a “golden” version, or post-processing it. cpptestcli generates a primary output data file in XML format. We will expand the test to include a regression control on an external XML file and use a built-in XML diff utility. At this point, our test is fully set up.

**Using Advanced Test Options**

Test Center’s GUI controls also enable more sophisticated test suite setup. For example:

- Specific set up / tear down steps for each test suite or a test case. These set up / tear down tests are defined in much the same manner as the “traditional” test case defined above.
- Conditional execution of tests based on their results (e.g. continue only if a certain previous test passed).
- Utilizing various data sources, including SQL databases, to drive the test cases.

**Test Execution, Reporting, and Traceability**

The created tests can be executed interactively from GUI, via a command-line interface to Test Center, or using a web services interface. Testing from the GUI is very simple: select any number of tests from the test tree, and either right-click to select a test configuration to run, or just click the blue triangle button on the tool bar:

![Test Center GUI](image)

Finally, Test Center’s built-in reporting effectively captures the status of all the existing tests, pass or fail, and includes a detailed caption of each failure, as shown in an example frame from a generated report:

![Test Center Report](image)

The red failure lines (e.g. *Changed 1 line*) are clickable, and open a separate window with the transcript of the difference for review.

Test traceability is realized by specifying a requirement or defect ID, along with a URL for the relevant information, when setting up a test in one of the Test Center dialogs. The status of test
execution can then be automatically correlated with requirements or defect status—either within reports created by Test Center, or, more comprehensively, within Parasoft Concerto.

Parasoft Concerto integrates into the various components of the SDLC and correlates all key artifacts, from tests, to requirements, to code, to project tasks. For instance, each test scenario is automatically linked to a related requirement or task. At any point, you can instantly assess the current level of verification for each requirement or task by back tracing to all associated tests. Moreover, as Concerto monitors the status of code and test artifacts for each requirement, it automatically issues a "Retest Required" alert when it determines that code was modified without rerunning tests.

This traceability not only establishes a flawless audit trail but also provides the base correlations for change impact analysis. This, in turn, helps the team optimize testing efforts by identifying and executing only the test cases directly related to the most recent source code modifications. Not having to test the entire system after each modification yields tremendous productivity improvements.

Extending Beyond Command-Line Regression Testing

The example we have considered here covers a shell command-line program. Test Center provides highly-automated facilities to set up regression tests involving one or more applications, including command-line executables, web applications, web services, database components, CORBA applications, etc. For instance:

- Web service functional testing ensures that the server delivers appropriate responses for the given requests. A test suite containing any combination of user-defined and automatically-generated tests can instantly be converted to a regression test, which helps you determine whether functionality changes over time. The same tests can also be leveraged for load testing and security testing.

- For web applications, it isolates and tests individual application components for correct functionality across multiple browsers—without requiring scripts. You can automatically create regression tests that ignore expected content changes for entire pages, or you can
create specific regression tests on individual page elements, thus eliminating the noise of annoying false positives.

- Database validation can also be performed. For example, assume you are testing a web service invocation that is supposed to add a record to the database. You can not only ensure that the service returned the expected response, but also query the database to verify whether the data was added to the database correctly.

- Parasoft's native support for CORBA assists you to construct, continuously execute, and maintain test suites that directly exercise components via the CORBA protocol. This provides visibility into how functional test transactions pass through the components—helping you to truly understand what functionality you have covered and diagnose the cause of any test failures that occur.

With a single test suite, you can continuously validate all critical aspects of complex transactions which may extend through web interfaces, backend services, ESBs, databases, and everything in between. During test execution, you can visualize and trace the intra-process events triggered by tests, facilitating rapid diagnosis of problems directly from the test environment. You can also continuously validate whether critical events continue to satisfy functional expectations as the system evolves.

To learn more about end-to-end testing with Parasoft Test Center, see the Parasoft “End to End Process Testing & Validation” white paper.
About Parasoft

For 21 years, Parasoft has investigated how and why software defects are introduced into applications. Our solutions leverage this research to dramatically improve SDLC productivity and application quality. Through an optimal combination of quality tools, configurable workflow, and automated infrastructure, Parasoft seamlessly integrates into your development environment to drive SDLC tasks to a predictable outcome. Whether you are delivering code, evolving and integrating business systems, or improving business processes—draw on our expertise and award-winning products to ensure that quality software can be delivered consistently and efficiently. For more information, visit http://www.parasoft.com.

Contacting Parasoft

USA
101 E. Huntington Drive, 2nd Floor
Monrovia, CA 91016
Toll Free: (888) 305-0041
Tel: (626) 305-0041
Fax: (626) 305-3036
Email: info@parasoft.com
URL: www.parasoft.com

Europe
France: Tel: +33 (1) 64 89 26 00
UK: Tel: + 44 (0)208 263 6005
Germany: Tel: +49 731 880309-0
Email: info-europe@parasoft.com

Asia
Tel: +886 2 6636-8090
Email: info-psa@parasoft.com

Other Locations
See http://www.parasoft.com/contacts