Automating Software Module Testing for FAA Certification

Usha Santhanam
Boeing

Abstract

Automatic software testing is gradually becoming accepted practice in the software industry. The shrinking development cycle and higher expectation of software quality are forcing software teams to seek help from automation. Current state-of-the-practice in software test automation uses tool-specific programming and scripting languages to automate testing. Automation aims to achieve both reduced costs and a more thorough analysis, testing and verification and is vital to keeping pace with increasing software complexity.

Software complexity and accelerated development schedules make avoiding defects difficult. The user reported bugs occur in software because the user executed untested code, or the order in which the statements were executed in actual use differed from that during testing, or the user applied a combination of untested input values, or the user's operating environment was never tested [3]. About 80 percent of the defects in software come from 20 percent of the modules, and about half the modules are defect free. In addition 90 percent of the downtime comes from, at most, 10 percent of the defects [2]. One of the common experience during software testing is the difficulty of accurately and efficiently capturing and analyzing the sequence of events that occur when a program executes[4].

Structural testing, also called code-based or white box testing, is performed for the purpose of exercising the code thoroughly. The code coverage measurements are used for evaluating effectiveness of the test cases toward this goal. Manual code-based testing, is cumbersome and time-consuming. Automation provides help in selecting test data, applying those test cases to the software, deciding whether a program has been tested enough by relating testing effort to coverage metrics.

This paper emphasizes automated module and subsystem level testing for FAA certification of software written in Ada using a tool set collectively called Test Set Editor

(TSE). Additional tools, Microsoft Excel spreadsheet program and home grown test scripts written in Tcl/tk are part of our test automation process.

The automation techniques and tools adopted in our process are designed to achieve shorter turnaround time and cut cost during regression testing. Although commercial off-the-shelf test automation tools such as AdaTEST are available, either they are limited to a specific application domain or they require the tester to code the test cases in the form of a test driver function in some programming language. The testing process we describe is domain-independent and it requires only that the tester create test cases in a tab-delimited text file perhaps using a spreadsheet program such as Excel. The rest of the process of testing is automated, removing the language-dependent test driver generation chores from the hands of the tester. The tester with little or no knowledge of Ada can create test cases without having to worry about debugging test drivers.

TSE is a multi-purpose test tool designed for use with Ada. It is a tool for building test sets (or test cases). Test cases are not automatically derived from code, but a test engineer must construct them. They are normally derived from requirements, but some may be derived from code for coverage purposes. TSE assists the test engineer with many of the common chores associated with testing:

- identifying inputs and outputs of a module (or a collection of modules),
- constructing a test driver,
- constructing stubs for subprograms that are called from the module or subsystem under test,
- tracking module requirements to test cases, and
- analyzing coverage .

The benefits of TSE are significant:

- Tests can be developed by engineers who are not necessarily expert programmers in Ada. This is a huge advantage in an environment where Ada programmers are hard to come by, especially if they must be brought in from outside for independence of verification and validation.
- TSE includes a tool to analyze the source code and identify inputs and outputs of a given module. These inputs and outputs include parameters and globally referenced

variables. Additionally, all subprograms called by the module are also identified so that the tester may plan a stubbing strategy for them. This semantics-based analysis step ensures that inputs, outputs, and subprograms are not overlooked due to renaming or overloading.

- Coverage information is provided both in tabular form for certification documentation as well as in HTML form which can be browsed using standard browsers. This latter form is useful during development when the tester needs to know where coverage deficiencies exist.
- TSE tests are easy to maintain. The tool-generated driver and stubs have uniform structure that is simple to comprehend in the unlikely event that generated script must be examined. Tests written by one engineer can be modified or maintained by another years down the road.
- TSE tests are self-verifying. When the test is run, the output identifies the result as pass or fail. There is usually no need for a separate verification step.
- TSE tests are repeatable. Everything needed to run a test is captured in files; no manual interaction is normally needed. Testing for regression is easy, and can be totally automated.
- TSE tests can be administered on a host or the target platform. Using slightly different support package, the same tests can be run on either platform. TSE also provides the convenience of host platform for development, eliminates additional work to apply the tests on the target and even targets with no I/O capability can be handled.

Quality of software is becoming increasingly important and testing related issues are becoming crucial for ensuring high quality for safety critical software. Keeping this in mind, we have designed our test automation to provide the testers with an effective test environment. The process is designed to be repeatable so that the cost of retesting of new versions of the software is kept to a minimum.

This testing process has been used in the structural testing of FAA certified commercial avionics software as well as the qualification of tools used in the production of such software.

References

1. Tsuneo Yamaura "How To Design Practical Test Cases "IEEE Software, November/December 1998, pp. 30-36.

2. Barry Boehn, Victor R. Basili "Software Defect Reduction Top 10 List", Computer, January 2001, pp. 135-137.

3. James A. Whittaker, "What Is Software Testing? And Why Is It So Hard", IEEE Software, January/February 2000, pp.70-79.

4. Thomas Ball, James R.Larus, "Using Paths to Measure, Explain, and Enhance Program Behavior", Computer, July 2000, pp. 57-65.