A Test Driven Development Approach to Medical Device Development

Author(s):
Timothy D. Nolan, Senior Automation Systems Consultant, Data Science Automation, Inc.
Richard M. Brueggman, Founder & Chief Executive Officer, Data Science Automation, Inc.

NI Product(s) Used:
LabVIEW 2012 SP1
VI Package Manager
TestStand 2012
LabVIEW Object Oriented
NI RealTime
NI FPGA

Category:
Advanced Research
Physical Test and Monitoring

The Challenge
Creating a framework for automated regression testing of LabVIEW code in an Agile, regulated, Test-Driven Development Project, covering multiple targets with test traceability.

The Solution
By combining the flexibility and test reporting capability of TestStand with the modular design and adaptive function of our Object Oriented LabVIEW code, Data Science Automation performed both Unit Tests and System Integration Tests in an automated and traceable manner by applying Test Driven Development.

Introduction
Data Science Automation is a premier automation systems integrator, leveraging COTS products in the design & implementation of custom-engineered solutions spanning mechanical, electrical, controls & software engineering disciplines. Headquartered in of Pittsburgh, PA, we serve national and international needs for systems integration. This particular challenge was to develop a medical device used in cancer treatment. Our previous relationship with the client team as a National Instruments Alliance Member and Certified Training Center allowed them to easily select our expert experience for this project.

Background
In the medical device industry, the development of both hardware and software are highly regulated in order to deliver safe products to the end users. This means tracking of all automated testing of code and the ability to easily regression test code is an absolute requirement. The product we were developing needed to adhere to ISO 13485 and similar standards.
For code development DSA implemented the Agile development method, with two-week sprints by a team of five Certified LabVIEW Architects. At the end of each sprint we had a full code deployment. We knew that we would have to have to reliably and efficiently apply all of our Unit and Integration tests to the code prior to each release.

The requirements for the system testing were therefore:

- Easy interface with LabVIEW
- Ability to perform tests on the same code across a range of simulated parameters
- Reporting with traceability to the test step level
- One-click function for automated testing

Approach
Early sprints used manual testing. This served to impress upon the team the need for automation methods. We examined two methods for performing the Unit Testing:

- LabVIEW Unit Test Framework (UTF)
- NI TestStand.

The Unit Test Framework met most of our requirements, and had the advantage that it could run from within the LabVIEW project and show the results of the tests in the same environment. However, when we needed to run some of the more complicated Integration tests, we were unable to track the results to the level of precision (at the test-step level) that we required. Although the UTF had the advantages already discussed, we needed to move on to our second option, NI TestStand.

NI TestStand was a much more flexible framework (Figure 1) in which to build our tests and produce the reporting and traceability needed for the application. The built-in XML reporting capability was extremely useful. Having the experience of both having attended and taught the National Instruments TestStand courses enabled our team to customize the report output to match up with the Requirements Tracking software used in Medical Device development - Aligned Elements.

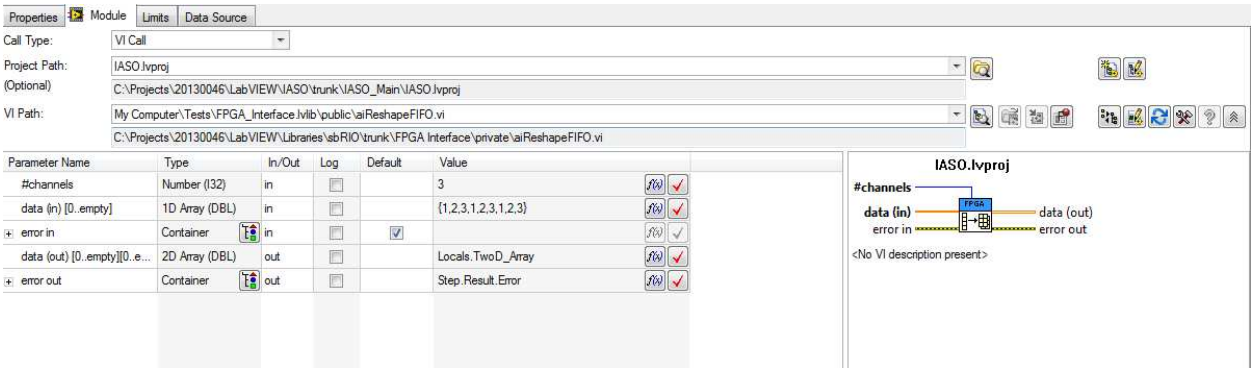

Figure 1. Unit Test Design in TestStand

For any specific test, we needed to provide a range of input values with different set ups or configurations. TestStand enabled us to vary those inputs across a range, or even to generate them with a custom equation or VI. With a simple selection of the LOG property per parameter, we could ensure that it was written to the test report for later traceability or auditing.

In addition to the parameters assigned to each test step, the test itself needed requirements traceability. This is an inherent part of the TestStand environment, where a Unique identifier entered into the Step> Requirements tab could be saved to the report as well.

The full interaction with LabVIEW allowed us to perform higher-level integration tests than might otherwise be performed with a Unit Testing system. For example, we could launch a module dynamically, using VI Server, and then run a series of steps that both simulated data or messages to the module, and were able to read responses from the launched module, either with queued messages or Functional Globals.

An example of this was our State Machine code. We could launch the main state machine, and then send it the state transition messages it was expecting and measure the correct transitions actually happening within the engine. This application matched very well to the TestStand environment, as each loop of the state machine could be applied as a sub-sequence of the main Transition Test Sequence.

Organization

In addition to the technical abilities of TestStand, the TestStand workspace and project enabled us to bundle tests by group or module (Figure 2). This allowed efficient testing of just portions of code during the development process. For example, if we had changed any of the database tables, we could easily run a regression test of just the Database group to ensure that existing functionality was still in place.

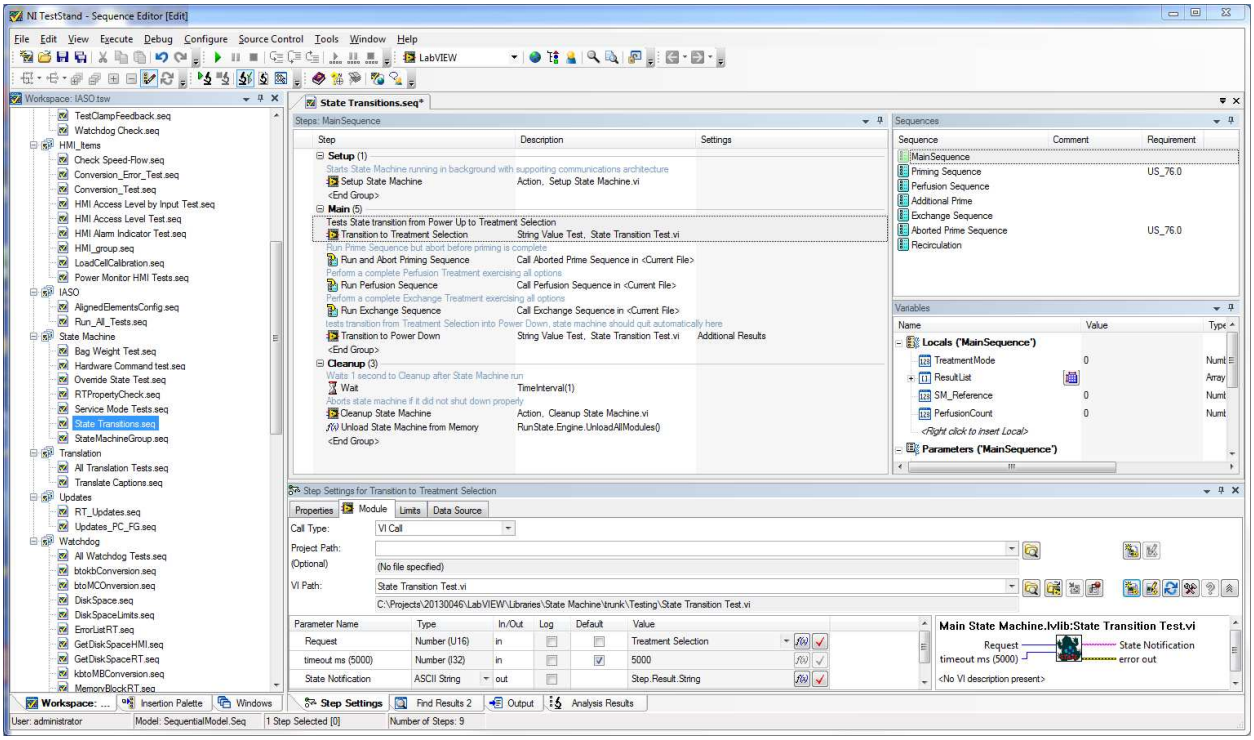

Figure 2. Grouping of Tests by Module

Implementation

Once the set of tests were in place, they could be traced and submitted with each code release as a part of Agile development. Each release would then include the code, tests, test specifications, and test report. We used Source Code Control (SCC), in this case Tortoise and SVN, to track the production and submission of the code and tests, as well as to allow rolling back to earlier versions for testing purposes.

During development, new tests would be added as the functionality of the code was improved. Also, as the code was changed, the need to check existing functionality was always a concern. To solve both these issues we established Continuous Integration. While some continuous integration systems consist of building the EXE and making sure that is successful, our implementation was to run the automated tests across the code as it was submitted each evening.

To this end, we had a dedicated computer that was able to run a dedicated LabVIEW program each evening. It would use command line arguments to download the latest code from the SCC repository, then use the TestStand command line to run the top-level sequence "Run_All_Tests.seq". In this way we would have feedback on whether newly developed code would successfully pass old tests, as well as how close the new code was to passing the newly established tests.

Results

The implementation of Continuous Integration and the Automated testing made our code development much more efficient, as catching issues with the code early saved a lot of development time later. Modules such as Network Communications modules could be continuously and automatically checked

for data integrity and message coverage, which in a manual mode would be time consuming and difficult to perform in a repeatable manner.

Using TestStand as a regression and automated test environment for code was extremely successful, and will be implemented as a framework for future projects, especially those with product development as an end goal, particularly in regulated industries that require test traceability.

Contact Information

Timothy D. Nolan, tdn@DSAutomation.com
Data Science Automation Inc., 375 Valley Brook Road, Suite 106, McMurray PA 15317
(724)942-6330
www.DSAutomation.com