"Test Automation in a Mixed Software/Firmware Environment"

Presented by:

Christopher Crapo
Boston Scientific

Brought to you by:

340 Corporate Way, Suite 300, Orange Park, FL 32073
888-268-8770 · 904-278-0524 · sqeinfo@sqe.com · www.sqe.com
Christopher Crapo
Boston Scientific

Christopher Crapo is the technical lead for the tools and automation team at Boston Scientific Neuromodulation and co-architect for the in-house-developed automation system. During the course of this effort, testing has progressed from fully manual to approximately 70% automated, with a testing cycle-time reduction of 60–95%. The automation system is now in use at every level of product development testing—from unit/regression testing to formal system-level verification. Bringing his background in electrical engineering and mathematics to software test and automation system design, Christopher has contributed on every level from PCB design and hardware interfacing to software development to overall design patterns and system architecture.
Test Automation in a Mixed Firmware/Software Environment

Christopher Crapo
Boston Scientific Neuromodulation
Chris.Crapo@bsci.com

Overview

- Challenges of mixed system automation
- Automation System Overview
  - What we actually built
  - Subsystem Breakdown
    - How we approached the individual components
- Organizational Approach and Challenges
  - Who really built it
- Questions and Comments
The Challenge

What makes automating a mixed environment difficult?

- Automation Tools are Domain Specific and do not usually share a deployment target.
  - UI test tools have only rudimentary support for real-time testing or signal driven testing.
  - Firmware test tools rarely focus on UIs and usually require a specialized operating system.
- Firmware poses problems that often require fully customized automation layers
How Can We Bridge The Gap?

We need a systematic approach to automation that is:

- Approachable
- Debug-able
- Flexible
- Adaptable
- Capable

For every domain that it supports

Remember

If the system is not understandable, people are not going to use it
Our Answer:

Build a Unified Script Environment
The Approach

- Leverage the flexibility of modern scripting languages to build a singular view of the test system.
- Each major component of the system is powered by a domain specific tool that is wrapped by a problem-centric interface.
- Each interface is crafted specifically to the needs of that sub-system.
- A scripting framework takes care of all test-centric details.

Overview

- Challenges of mixed system automation
- Automation System Overview
  - What we actually built
- Subsystem Breakdown
  - How we approached the individual components
- Organizational Approach and Challenges
  - Who really built it
- Questions and Comments
Tool Selection

- For all tools, emphasis is placed on customizability, maintainability, and capability.
- Ease-of-use is nice, but gets mostly hidden by the custom API.
- The end goal is to be able to make it simple for the testers to interact with the functionality from a script level.

Overview

- Challenges of mixed system automation
- Automation System Overview
  - What we actually built
- Subsystem Breakdown
  - How we approached the individual components
- Organizational Approach and Challenges
  - Who really built it
- Questions and Comments
An off-the-shelf test management program provides the testers an environment to plan, group, and start tests.

The test manager runs a script that starts our automation system from the command line.

The test management script does not include test functionality – only test management functionality.

Results are reported back through the return value of the automation process and XML files.
Scripting Framework

The scripting framework focuses specifically on the test by:

- Tracking test steps and overall test status
- Aggregating Logging/Reporting information
- Acting as a mediator between modules
- Providing functionality for manual steps
- Communicating with the test management layer

Test and Step Tracking

- The core manages tracking steps and test results.
- All pass/fail checks are routed through the test framework so that the overall result of the test can be managed.
- The test framework normally stops on verification failures and script/environment errors to prevent damage to the unit under test.
- A flag is provided, for debugging, that allows the system to continue after a verification failure, but not after a script or environment error.
Logging and Reporting

- The script framework aggregates the execution logs from all of the sub-systems – in chronological order.
- Aggregating the logs is critical for the testers to be able to correlate the systems actions and debug any failures.
- Consider having a second, easy to read, log that captures the calls the test makes to the API.
- This gives a human readable overview of the test case and can be useful for understanding and communicating test flow.

Mediating Between Modules

- This is a primarily data driven activity for the script framework.
- System specification-level data structures are passed between modules.
- These data modules are likely to become the test cases for a number of tests.
- Avoid emulating the behavior of the software or firmware as much as possible.
Plan for manual steps!
It is important to be able to establish the framework of a test before all of the pieces are in place to automate it.
Sometimes, you just have to have a person unplug a cord.

Have a plan to move past manual steps.
It will likely be necessary for testers to add manual steps to tests—especially early in the development of the automation system. Be careful, because this can lead to a lot of rework.

Manual Test Steps

- Wireless / Serial Communication
- Tester Callable Interface
- Test Management
- Test Execution Engine (Test Script)
- Signal Analysis
- Firmware Test API
- Software Test API
- UI Automation
- Database Automation
- Firmware
- Software
- Database
The main interface between firmware and software is our system’s Command Protocol, which provides a rich set of interactions.

Much of the testing of the firmware is accomplished by sending and receiving these commands.

Each command can be sent using an auto-generated class. This allows the IDE to prompt the tester with command options.

The therapy delivered by our device is a set of electrical signals output on a group of electrodes meant for the spinal column.

This “Therapy Program” is represented in the test framework by an object that can read/write itself between XML or to the device using the command objects.

In general, an event driven firmware architecture is easier to automate, because the automation system can operate by injecting events.
Signal Analysis

Scripting languages are not fast or time-deterministic enough to perform on-the-fly signal analysis. Temporarily turning control of the test execution to a real-time tool is necessary.

The ability to measure real outputs of the device is enabled by the same hardware that is used in manufacturing to test individual devices.

This hardware gives the ability to control or read:
- The Battery Voltage
- The Output Electrical Resistance
- A Noise Generator
- An Electromagnet
- Power Consumption
- Internal Signals of the Device

The “Therapy Program” object has proven to be a powerful tool for signal analysis, thanks to an underlying tool that accepts the XML and checks that it is actually produced on the device output.

This enables a testing flow of:
- Set the device to a particular configuration
- Write a “Therapy Program” to the device
- Monitor the electrical outputs to verify they are correct
UI Automation

- The First focus is on enabling basic functionality.
  - Identifying elements
  - Button Presses
  - Reading and Writing Text
  - Checking UI Attributes
  - Reading lists and grids
- The code for this level exists primarily at the tool level.
- The most important role the tool plays is making the GUI appear stable to the script.
UI Automation

- Advanced functionality is built on top of the basic functionality.
- This functionality is built in the script level and not at the tool level and focuses on the workflows.
- Because it is in the scripting environment, most of this functionality can be built by the testers themselves.

Database Automation

- Our database is currently localized to individual units and not subjected to intense interactions.
- Database automation mostly requires being able to execute queries from the script.
- An executable is placed on the unit-under-test to bypass remote permissions issues.
- Incorporating load testing would require the addition of another underlying tool.
Overview

- Challenges of mixed system automation
- Automation System Overview
  - What we actually built
- Subsystem Breakdown
  - How we approached the individual components
- Organizational Approach and Challenges
  - Who really built it
- Questions and Comments

Conway’s Law:

Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations.
The Real Geniuses

- Each testing group has at least one member that is the primary contributor to automation for the portion of the system that their group focuses on.
- These individuals are most familiar with the test strategy, difficulties, and opportunities their group faces.
- The stand-out value of the system is due to expertise that they bring to the table.

Remember the Developers

- Design tweaks, or even non-intrusive test hooks, can be a huge boon to automation.
- Developing a relationship with the people that make the decisions and implement those types of changes can make life a lot easier for everyone.
- Our most successful part of the automation system is driven by a firmware developer – he ensures that the firmware and automation play well together.
Value

- Automation set us up to go from product completion to release in weeks instead of months.
- Automating the more technical aspects of testing allowed us to bring on testers at the last minute to push through.
- Ability to continuously re-run tests enabled real regression testing and took much of the uncertainty out of our final testing.
- These add real business value!

Conclusion

- Automating a mixed Firmware/Software system is possible!
- Bringing off-the-shelf tools together in a custom scripting environment harnesses the power of the tools, while keeping the focus on testing the product.
- The value in automation is present for custom systems as long as the custom pieces focus on filling the needs of the organization.
Questions?