Unified Test Library and Fault-Tolerant Automatic Test Script Creation

Chandrashekhar M V; Manini Sharma

chandrashekhar.m.v@intel.com; manini.sharma@intel.com;

Abstract

The process of test content development, test automation and faultless execution of complex systems, involving unstable or under-development hardware and software, is a challenging task. The product design complexity, interoperability issues, component interdependencies, changing product requirements, crunched schedules for parallel product lines and market pressures lead to compromise on quality of test development, reuse, review, maintenance and automation scripting. How do we curb the churn in test content, bug escapes, coverage gaps, security vulnerabilities, automation reliability efforts, and test environment stability issues, and still save resources with maximum reuse across projects?

There are several commercial tools available for test management. These tools provide the option of traceability of requirements to test cases through relationship metrics. With product development methodology changing from traditional approach to agile, Behavior Driven Development (BDD) and Test Driven Development (TDD) are becoming vital for time-to-market (TTM) while maintaining high quality. While the new enhancements and feature additions are a continuous process in the agile world, regression testing is necessary to keep the product stable and reliable for every release. As regression testing is a costly process on continuous releases, the industry is moving towards more automated tests to minimize manual effort. Automating the test content in line with the manual test steps and having traceability at test-step level is a tedious process. There aren't many tools available for it. In complex multidisciplinary projects, the subsystem tests, Integration tests, and System Tests have commonality and can be re-used for use case testing while keeping focus on subsystem functionality testing. If the test system is carefully designed for re-use by decomposing the test steps with objective functions as meta data with the approach of BDD/TDD that is platform agnostic, the metadata can be used across subsystems, Integration tests, and System tests for functionality, regression, stress testing without much of effort both for manual and automated tests. Reliable automation can be achieved by coupling the BDD/TDD decomposed test functions with the Fault recovery mechanism during catastrophic failures of the test systems to auto recover during testing and continue the regression tests for lengthy (e.g. overnight) unattended tests.

The paper explains how the product development team built the Unified Tests and automation framework which makes the reliable test content for re-usability and automation with inbuilt Fault tolerance system for recovery of the system during catastrophic failure for test execution.

Biography

Chandrashekhar is an Engineering Manager at Intel India (Bangalore) with extensive experience in product development, with expertise in system architecture, operating systems and embedded software development related roles. He worked on implementing CMMi, ISO to the organizations with six sigma approach on the quality practices. He is currently managing teams which are responsible for Firmware (FW) System validation and automation at Intel products across all market segments (such as phones, tablets, laptops and desktops.)

Manini Sharma is a Platform Software Quality Engineer at Intel, based in Bangalore, India. She works on software for Windows devices based on Intel Architecture. She's worked in a variety of roles in the past decade including Product Marketing, Software Development and Application Engineering. She's been focused on Software Quality for the past five years. She has an M.Tech. in Information Technology from IIIT-Bangalore and an MBA from IIM Kozhikode.

1 Introduction

Validation is a key element of a product's lifecycle. Script-based validation has evolved in both software and hardware industries with multiple tool chains and test management systems. There are several systems and methodologies for test management and automation. However, there is scope for standardization of the process for moving from manual testing to seamless automation. This can be done using test steps as objects and methods, based on the specific industry or technology. The general tendency is to maintain test cases for manual execution and automation separately. During the execution of a project with multiple subsystems (being developed at the same time), the probability of test content getting changed during the lifecycle of the product is very high. This results in divergence between manual and automated test systems, if not monitored and controlled effectively.

In this paper, we apply the test content development methodology with standardized process using the technique of metadata creation. A unified test and unified user-defined metadata has been developed which helps in tight coupling of requirement changes to the test content and subsequent automation. The paper is structured as follows: in section 2 we elaborate on the problem of generating test content and automation, in section 3, we detail available methodologies on test case and automation management and the chosen approach. Section 4 is dedicated to the case study of implementation and the result. In section 5, we discuss the key benefits.

2 Problem: Test Content Management and Reliable Automation

The process of test content development, test automation and faultless execution of complex systems, involving unstable or under-development hardware and software, is a challenging task. The product design complexity, interoperability issues, component interdependencies, changing product requirements, results in the continuous maintenance of test content and automation scripts. Delays in development and frequent requirement changes invariably result in crunched schedule during the validation phase. The crunched schedule for validation of parallel product lines and market pressures, lead to compromise on quality of test content, reusability, review effectiveness, maintenance and automation scripting. How do we curb the churn in test content, bug escapes, coverage gaps, security vulnerabilities, automation reliability efforts, and test environment stability issues, and still save resources with maximum reuse across projects?

3 Methodology of Test content Management & Automation

There are many methods the industry follows to define test content and automation. The usage varies from maintaining the test cases in Excel or SharePoint to test management systems like HPQC/JAMA/JIRA. With the BDD Approach, the tests are directly written for automation and maintained as source code, written using Cucumber-Gherkin or Google Autotests (Google Inc. n.d.). While these approaches are useful for the qualified automation engineer to write tests and execute them, many validation teams maintain manual procedures (Cognizant 2014) and automated procedures (Geometric White Paper 2011) separately for execution, in case automation fails. This paper deals with effective usage of available BDD methodologies, making the test methods readable as plain English for the users of manual execution and also for reading test results. It is blended with object oriented paradigm (Quali Systems n.d.) for exception handling during catastrophic failures, while enabling test class and methods, built in with hash mapping techniques for test decomposition and re-use. This is achieved by decomposing the test content creation and automation into 4 subsystems:

1) **Unified Tests and Meta methods creation** with BDD as focus. This is equivalent to a class template in code. Each functionality is decomposed and grouped into a test class and its parameterized test methods with defined test syntax.

2) **Instant Automation Development.** These are formed with hash mapping technique using regular expressions to map/pick the right template of test class/method and generates the test script. Test content

is written for various purposes like testing functionality, unit level tests, scenario testing, stress and stability using the test methods with instance of test objects. This module gives context-based syntax checks for writing the test content easily.
3) **Fault Tolerance System (FTS) for environmental recovery** - This system is for environmental recovery during systemic failures ensuring uninterrupted test execution. FTS monitors for any systemic failures, launches its environment recovery services in the background to enable seamless execution without manual intervention during overnight, or large size test executions. This includes re-setting of the system to factory mode/equivalent, re-installation of OS, firmware and initializes the test bed for re-execution of the failed tests.
4) **Standardization of test bed for continuous Delivery**. This is an Integrated Framework deployment setup of test bed for continuous delivery of the incremental changes in the product to get tested automatically by triggering the test through Build servers which monitors the code change (Check-in) and generates/selects the required test scripts get executed instantly for regression and functional tests.

4 Case study: Firmware Subsystem Automaton

4.1 Background

At Intel, multiple programs are executed in parallel and in incremental fashion. With traditional test development and test management, it was tedious to accurately identify test re-use. Also, the traceability of re-use became difficult when the test cases were cloned and modified to suit various program needs. System test designs are complex in nature for the hardware, firmware and software system as single product deliverable. Test automation traceability and maintainability also gets affected, impacting the productivity. With the traditional approach of test development and automation, the process is subject to low degree of re-use in the scenario of parallel and incremental projects. There was a need to have a unified standard, providing holistic improvement on test coverage, automation and seamless execution process with debugging and analysis, while not compromising on quality A solution was needed which allowed test content convergence, effortless, scalable automation, stable and reliable execution infrastructure, with increased velocity.

4.2 Standardization of Tests and Test Framework development

At Intel the Firmware team realized the need of standardization of test content which can be turned into instantly automated tests. The focus area was to develop a Framework which ties the following:

1) **Unified Tests and Meta methods creation** with BDD as focus
2) **Instant Automation Development**
3) **Fault Tolerance System** - A mechanism to recover the test bed from catastrophic failures to continue the test and
4) **Standardization of test bed for continuous Delivery**

Test content development is done with BDD as focus and **Unified Tests and Meta-methods** are created by scrubbing through all the available test cases on each market segment and subsystem. The focus is on identifying the common test steps/procedures to create standard methods in each category of system features. These methods are defined in a framework or library which consists of defining the Method Name, Syntax for the method with parameters and Regular Expression for syntax checking. This makes it easy for test content writer and automation engineer to use the methods for auto complete of content/scripts. Table 1 depicts the sample methods, syntax and the regular expression for syntax validation.

Test Method	Syntax	Regular Expression
Boot to *OS*	**Boot to** <environment> {**from** <source>}	`( *)Boot( +)to( +)(\bos\b|\bedk shell\b|\bsetup\b|\bBIOS setup\b|\bTBOOT\b|\bubuntu\b|\bmebx\b|\bRAID\b|\bFW RECOVERY\b)(( +)from( +)(\bHDD\bSATA-SSD\b|\bEMMC\b))?( *)`
Put *system* **to** *S3* **using** *pwr_btn* **for** *30* **seconds**	**Put** <System> **to** <Sx_state> **using** <Source> {**for** <time> **seconds**}	`( *)Put( +)(\bSystem\b|\bClient-System\b|\bHost-System\b)( +)to( +)(\bS[3-5]\b|\bDeepS[3-5]\b|\bCS\b|\bdMoS\b|\bcMoS\b)( +)using( +)(\bidle wait time\b|\bKVM-USBR\b|\brtc\b|\bpwr_btn\b|\blid_action\b|\bOS Power Policy\b)(( *)$|(( +)for( +)\d+( +)second(s*)( *)))`

Table 1 Sample methods, syntax and regular expression

Instant Automation Development framework is developed with a parser and code generation module which reads the test content and test steps, passes through syntax checker and hash mapping technique to pick the right template of test class/method and generates the test script on the fly for the expected behavior for the method and parameters used in the test scenario. Figure 1 depicts the Instant automation framework for unified test class which generates the test scripts automatically based on the test template and the input parameters/metadata. Automation code will be generated from the Instant Test framework by reading each step and using the library, test scripts will be ready for use as a script package.

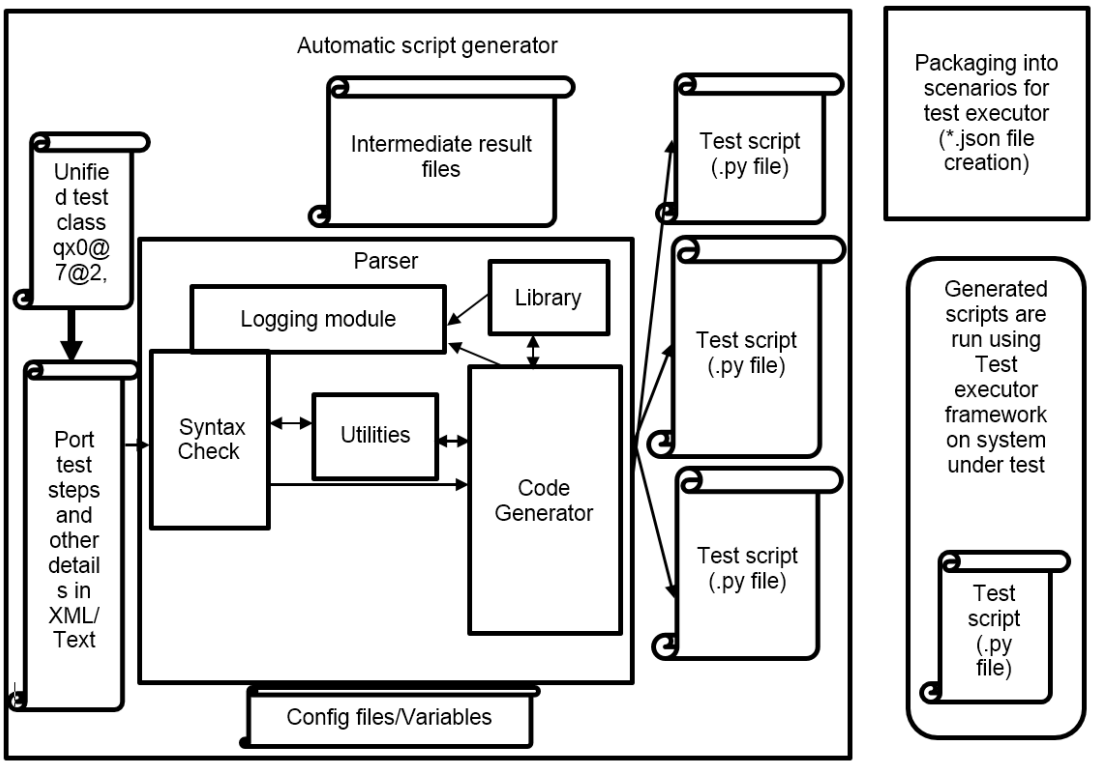

Figure 1: Unified Test Framework for Auto Test script Generation

Figure 2 depicts the flow of dynamic code generation when the change happens in the test content. There are two scenarios (marked in Unified test class – Use case) of test content change and the related code change is highlighted in the sample code.

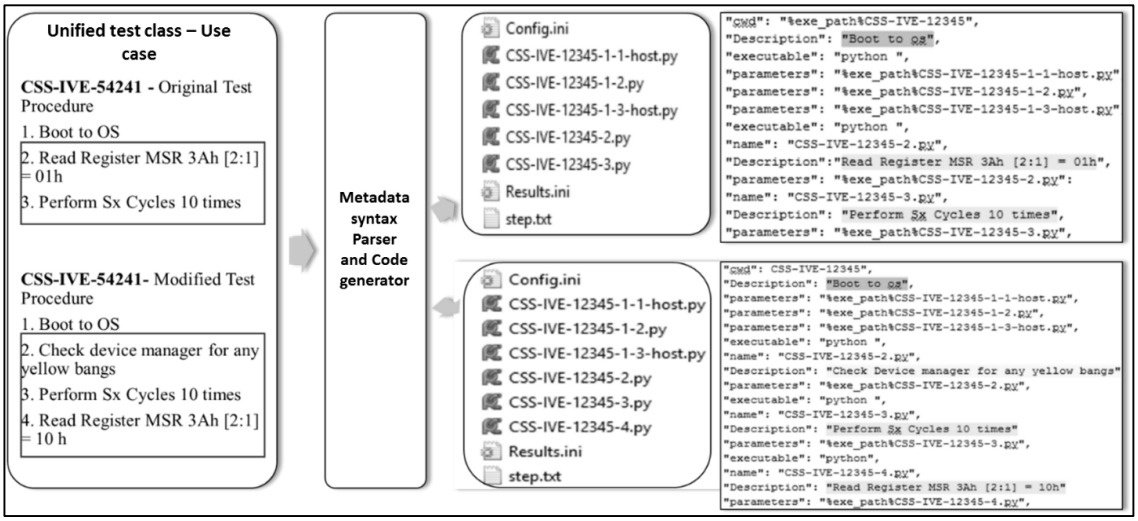

Figure 2: Code generation Flow

Fault tolerance system (FTS) for environmental recovery framework developed which interacts with test executor and identifies the systemic failure of the system (e.g. blue-screen-of-death (BSOD) error or system startup failures). During systemic failures, FTS launches its environment recovery services in the background to enable seamless execution without manual intervention during nightly, or large size test executions. FTS is a continuous monitoring service and has the capability of recovering the system by re-flashing the firmware and software through add-on devices like FW Programmer to the system under test (SUT). Along with the recovery of the system, FTS also collects the logs and information about the system failure, for easier debugging and creating the environment of faultiness to reproduce the issue as well as determine the root cause, helping ensure faster turnaround on product defect resolution. Figure 3 shows the FTS sequence.

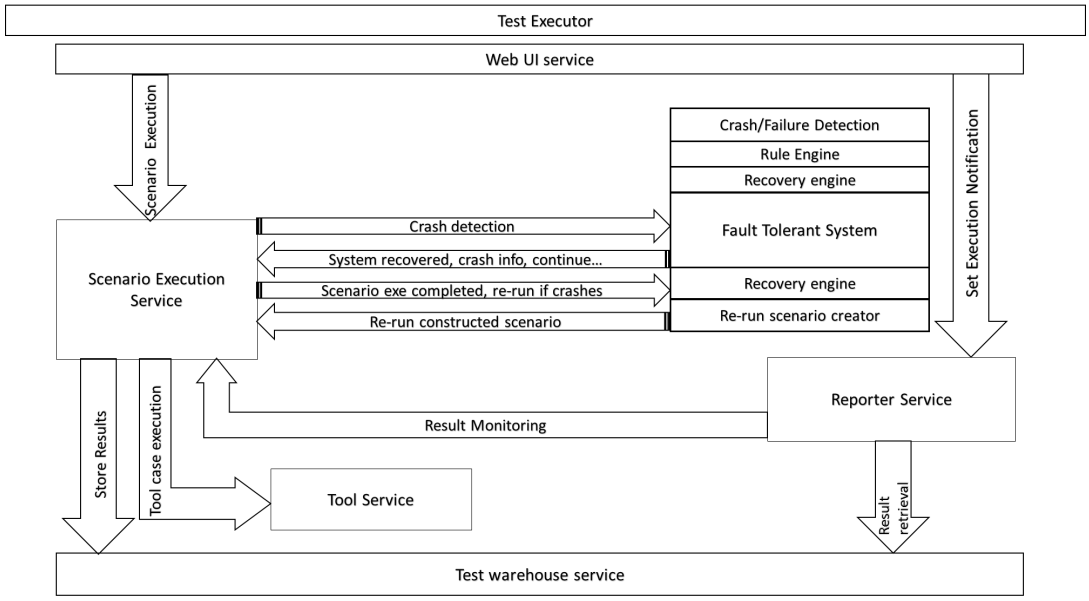

Figure 3: Fault Tolerance system Architecture

With the above framework developed, the last part is the **Standardization of test bed for continuous Delivery.** Below diagram shows the Integrated Framework deployment for continuous delivery of test content for execution. Test beds are paired with a Host monitoring system which acts as test agent, execution controller and Fault monitoring system. Test agents also connect with programmable HW device (relay, power switches) for any external electro mechanical action, like Power button control on/off, switching between power devices and touch sensor tests. The system is also connected with the subsystem product source code through the build server for continuous integration.

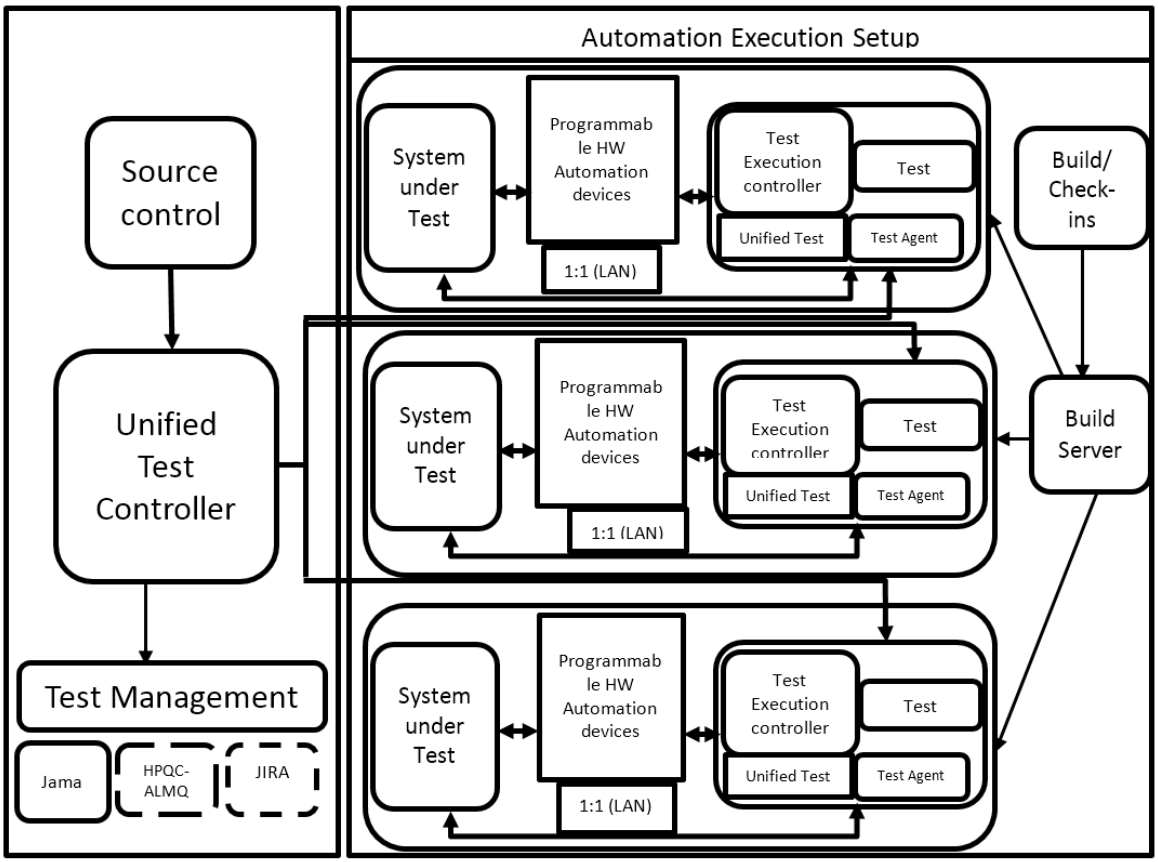

Figure 4: Unified Tests Automation Deployment Diagram

5 Key Takeaways and Benefits Derived

There were several key takeaways in the process of standardizing the specification for unified test framework:

1. Traditionally automation effort used to start after completion of test content creation. With this methodology, automation becomes instantaneous and can be used immediately.
2. Ramp up time for new resource reduced from 8 weeks to 2 weeks to understand the test methodologies. This happened because the focus was only on the smaller set of test objects rather than the entire suite of test cases.
3. Single library of test metadata helped cross-team synergy for subsystem and system validation with more than 60% reuse.
4. Fault tolerant system as a standalone mechanism for system recovery and telemetry recording was used by teams aiming at nullifying manual dependency during systematic failure such as Windows Blue screen error (BSOD).

5. Each of the unified meta-methods are associated with the dependent hardware and software tools as attributes. This helps in long term maintenance of the framework in case of any changes in the tools used.

6 Extending the application of Unified Tests

At Intel, this approach was first piloted with FW validation. Multiple improvements in the approach were implemented on the re-use and standardization of framework subsequently. The approach was then extended to System Integration and Validation teams to maximize re-use with minimal additions of test metadata. The benefit of re-use and instant automation reached up to 80% from the existing library. This methodology is planned to be extended further to multiple other subsystem teams to make an organization-wide test library, which can be used across projects and teams for instant automation readiness.

Application of this framework/methodology can be extended to any type of SW/HW/embedded program testing. Product development organizations can achieve significant productivity benefits across the organization. Having a common library of unified tests related to functionality, stress, regression, performance, integration and system tests can make the test organization to re-use and leverage across teams with common library. In the open source world, the same suite of test cases can be shared with the open source community to contribute similar source code to make standardization a solid test suite for the products.

7 Conclusion

The Intel product development team spent multiple weeks of effort to deal with high volume testing of client products across market segments. This inspired a search for better methodologies to optimize the test content with automation focus. A unified test content framework was piloted in a subsystem team and proliferated across other Intel teams to validate tests across. This was done while keeping re-use and standardization automation process in mind. This helped in saving of over a million USD in one subsystem team in a year.

Unified Test Class, User-defined Metadata and FTS make a powerful solution to have a tightly coupled automation focused test development and execution which can be used as a standardized software testing product.

8 Bibliography

Cognizant. 2014. "Transforming Test Automation: An Unconventional Approach to Shift-Left by Removing Scripting from the Equation." *https://www.cognizant.com/.* Accessed July 2016. https://www.cognizant.com/insightswhitepapers/transforming-test-automation-an-unconventional-approach-to-shift-left-by-removing-scripting-from-the-equation.pdf.

Geometric White Paper. 2011. "Scriptless Test Automation." *http://geometricglobal.com/.* February. Accessed July 2016. http://geometricglobal.com/wp-content/uploads/2013/03/Geometric_Whitepaper_scriptless_test_automation.pdf.

Google Inc. *Autotest for Chromium OS developers.* Accessed August 2016. https://www.chromium.org/chromium-os/testing/autotest-user-doc.

Quali Systems. "Object Oriented Test Automation." *http://www.webtorials.com.* Accessed July 2016. http://www.webtorials.com/main/resource/papers/QualiSystems/paper1/Object-Oriented_Test_Automation.pdf.

9 Acknowledgements

The authors would like to thank the reviewers for their valuable comments and suggestions to improve the quality of the paper. We also thank our organization, Intel Corporation, for the support and funding for this paper. We especially want to thank

- Keith Stobie – PNSQC Reviewer
- Patt Thomasson – PNSQC Reviewer
- Soumya Mukherjee – Intel Corporation
- Sneha Pingle – Intel Corporation
- Vivek Malhotra – Intel Corporation
- Christopher Hall – Intel Corporation
- Pramod Mali – Intel Corporation