Flight Software Development and Validation Workflow Management System

Dan Gultureanu
Space Systems/Loral (SSL), Software Systems Engineering
Manager, Flight Software
3825 Fabian Way, Palo Alto, California 94303, USA
Dan.Gultureanu@sslmda.com

Kevin Kerns
Space Systems/Loral (SSL), Software Systems Engineering
Senior Software Engineer

Tom Henthorn
Space Systems/Loral (SSL), Software Systems Engineering
Manager, System Validation

John Quach
Space Systems/Loral (SSL), Software Systems Engineering
Senior System Validation Engineer

Mitch Kleen
Space Systems/Loral (SSL), Software Systems Engineering
Senior Software Engineer

ABSTRACT

The aerospace industry is undergoing unprecedented transformation. New players, new ventures, and new technologies drive the established business towards a transition from traditional models and strategies to automated production and innovative methods. Software architecture for space systems needs to become more agile while preserving a high standard of quality. To that end, the rigorous verification and validation process that yields high quality software products should embrace automation and take full advantage of COTS hardware and software products. In line with the industry transformation, SSL's Flight Software Development and Validation Department uses a modular, layered architecture that has evolved over decades of orbital experience. This paper describes the SSL Flight Software Development and Validation Workflow Management System and the advantages of such a system/framework for development and verification of very complex spacecraft. This system/framework allows faster cyclical test-fix-test process and also provides a platform that can be extended for a multitude of applications.

KEYWORDS: *Software Development, Software validation, Testing architecture, Layered architecture, COTS hardware and software*

NOMENCLATURE

API - Application program interface
COTS – commercial-off-the-shelf
CVS - Concurrent Versions System
FSW – flight software
Git - git-scm.com
I/O – input and output
SBC – Single Board Computer
SDLC – software development lifecycle
SVN - Apache Subversion
T&C – telemetry and commands
UI – user interface
V&V – verification and validation

1 INTRODUCTION

1.1 Waterfall, V-Model and Agile

The aerospace industry traditionally uses a waterfall or V-model (Fig. 1) in software development lifecycles. The V-model [1] derives its name from the terms Verification & Validation. It is based on a sequence of phases where each phase is completed in its entirety before the next phase is initiated. Each development phase is associated with a well-defined test phase. It provides a top-down validation approach and supports requirements-driven design.

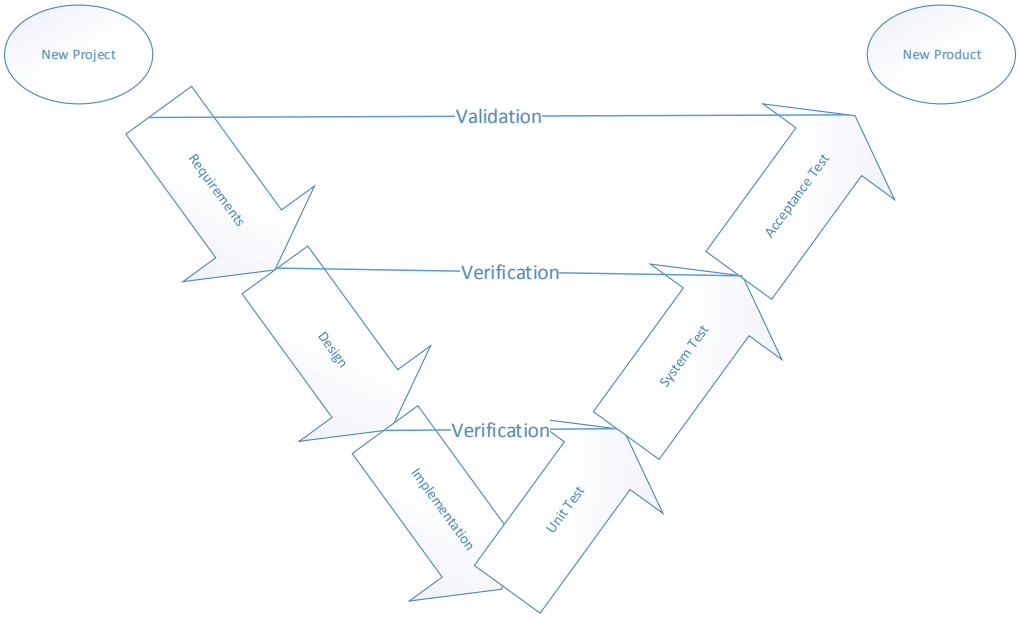

Figure 1: Simplified V-Model

The need for faster development lifecycle, introduction of new technologies, expanded business portfolio, and frequent changes of requirements during software development demand adoption of commercial methods and techniques already utilized in other industries. Agile [2] (Fig. 2) and Spiral are examples of such methods that are considered for integration in the software development lifecycle along with the traditional models.

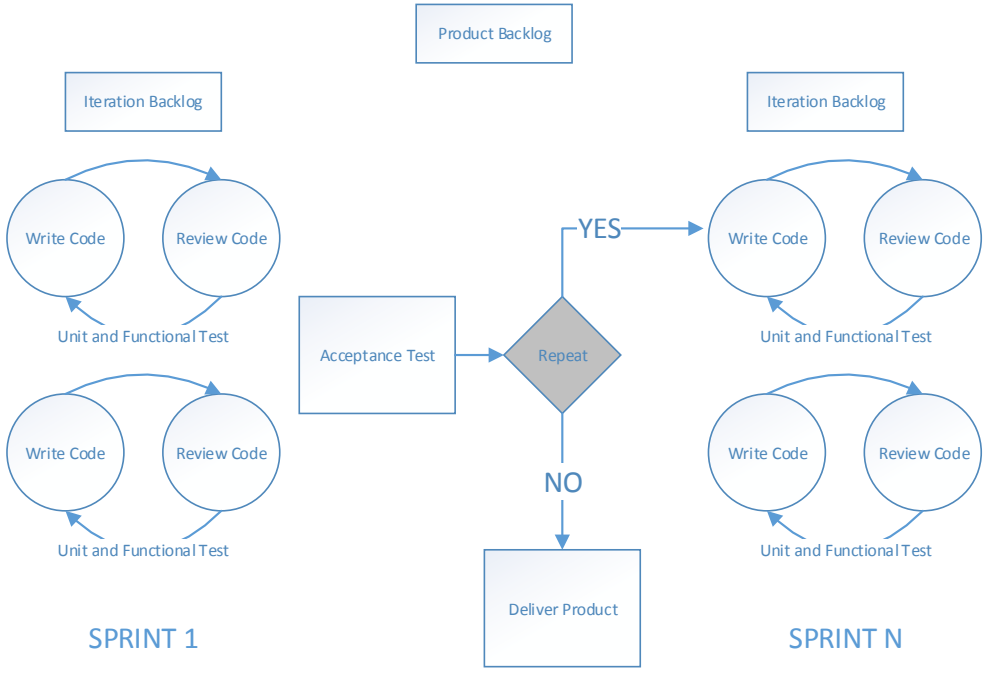

Figure 2: Simplified Agile Model

When safety and reliability are critical, however, requirements driven development is often relied on given its long track record of success and its alignment with established regulations. Agile V-model [3] [4] [5] (Fig. 3) software development lifecycle is a viable alternative that takes advantage of an incremental, iterative, and collaborative process while maintaining the rigorous verification and validation process based on requirements driven development.

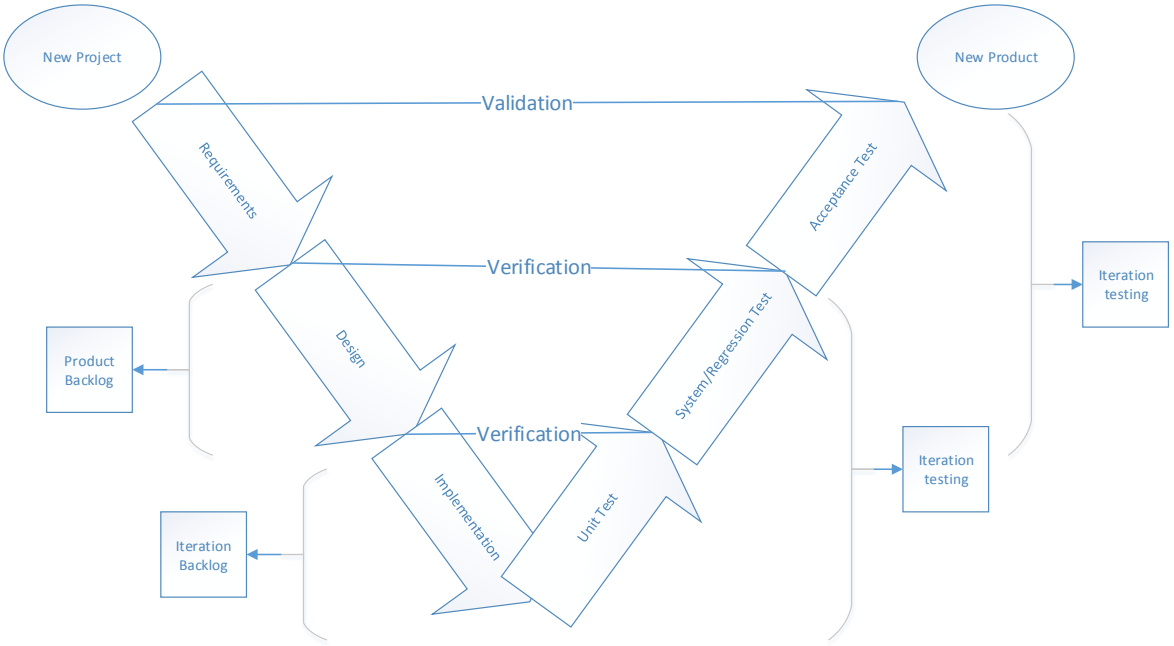

Figure 3: Simplified Agile V-Model

There is a place to deploy all types of software development lifecycles, including sequential ones. Many agile best practices have roots in V-model ideas [3] [5]. Embracing good ideas and being flexible, in any lifecycle, will allow the development teams to tailor, adapt, blend, and extend traditional and newer methods and techniques. This process is especially beneficial in industries with strong regulatory requirements like aerospace.

Agile V-model maintains the following ideas and practices:

- Independent testers
- Up-front design (retaining flexibility)
- Defect tracking
- Lifecycle tailoring (adaptability)
- Tasks outside iterations (automation, spacecraft level testing, etc.)

Agile V-model adapts or no longer uses the following ideas and practices:

- Sequential process
- Only one delivery at the end of the software development
- Cycle transition approved only by the validation organization

The agile V-model can be viewed as an enhancement of the test driven development concept by adding the acceptance/business case to complete the software development lifecycle. SSL's flight software development and validation follows an agile V-model adapted to requirements driven development and aligned with aerospace regulations.

1.2 Test Automation and Test Automation Frameworks

Test automation is the use of software to execute tests and then determine whether the actual outcomes are consistent with predicted outcomes. The shift from sequential to agile development allows the usage of test automation throughout the product life cycle. A test automation framework is a constructive blend of various guidelines, coding standards, concepts, processes, practices, modularity, reporting mechanism, user interfaces to support automation testing.

The test automation framework responsibilities are:

- Defining the format in which to develop and maintain the test definition and pass/fail criteria, including the capability to eliminate duplication of test cases

- Creating an interface to other systems responsible for requirements and pass/fail criteria definition (i.e. requirements driven development)
- Creating a mechanism to interface with the application under test
- Executing the tests
- Reporting results

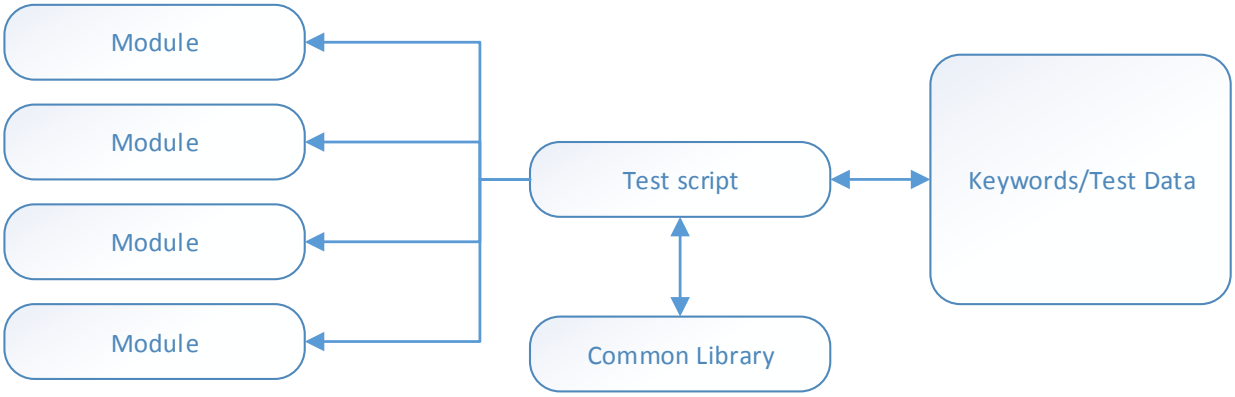

Figure 4: Simplified Hybrid Testing Framework

Throughout different industries several test automation frameworks [6] are used:

- Module based testing framework
- Library architecture testing framework
- Data driven testing framework
- Keyword driven testing framework
- Hybrid testing framework
- Behavior driven development framework

Hybrid testing framework (Fig. 4) is a combination of more than one of the above mentioned frameworks. SSL's flight software development and validation uses a hybrid testing framework that maximizes the benefits of the various associated frameworks.
SSL flight software test automation framework attributes:

- Application independent
- Scalable and modular (easy to expand and maintain)
- Re-usable
- Improves testing efficiency

2 SSL FLIGHT SOFTWARE DEVELOPMENT AND VALIDATION WORKFLOW MANAGEMENT SYSTEM

2.1 Software Development Lifecycle

SSL designs and builds spacecraft to support a myriad of custom missions. The Flight Software Development and Validation Department uses a modular, layered architecture that has evolved over decades of orbital experience. SSL follows an incremental life-cycle process. The software validation employs successively higher fidelity testbeds to match the development maturity stages, ultimately culminating in spacecraft-level testing.
The large number of spacecraft that SSL designs and builds necessitates the adoption of a formal and rigorous process for requirement verification. This includes integration of new or modified code modules into a layered architecture, and validation through spiral testing such that the completed modules function as designed and perform in concert with the overall mission objectives.
As with the layered architecture that is the core of SSL's decades-rich orbital heritage spacecraft systems, the software test infrastructure is similarly structured to support verification of flight software design and implementation. The verification and validation environment consists of a variety of simulators, engineering models and higher-throughput test stations built on top of commercial-off-the-shelf (COTS) hardware.
SSL developed these higher-throughput test stations with configurations that are representative of the flight units:

- Commercial platform utilizes the same CPU architecture as the spacecraft flight processor
- Flight software applications compile the exact same code across platform configurations
- Differences are limited to minor board support package functionality including low-level I/O interfaces.

COTS test stations (Fig. 5) consist of a COTS Single Board Computer (SBC) that executes the flight software code and interfaces over a backplane to a second SBC. The second SBC executes software simulation modules which are representative of spacecraft hardware interfaces and spacecraft motion and dynamics. The hardware interface emulation includes data buses, sensors and actuators, command decoders, telemetry encoders, power control equipment, solar array driver equipment, and any other hardware present in the specific architecture of the spacecraft under test. The test station is configured/driven by a ground test software system that manages setup of the initial conditions of the test scenario and provides real-time stimulation.

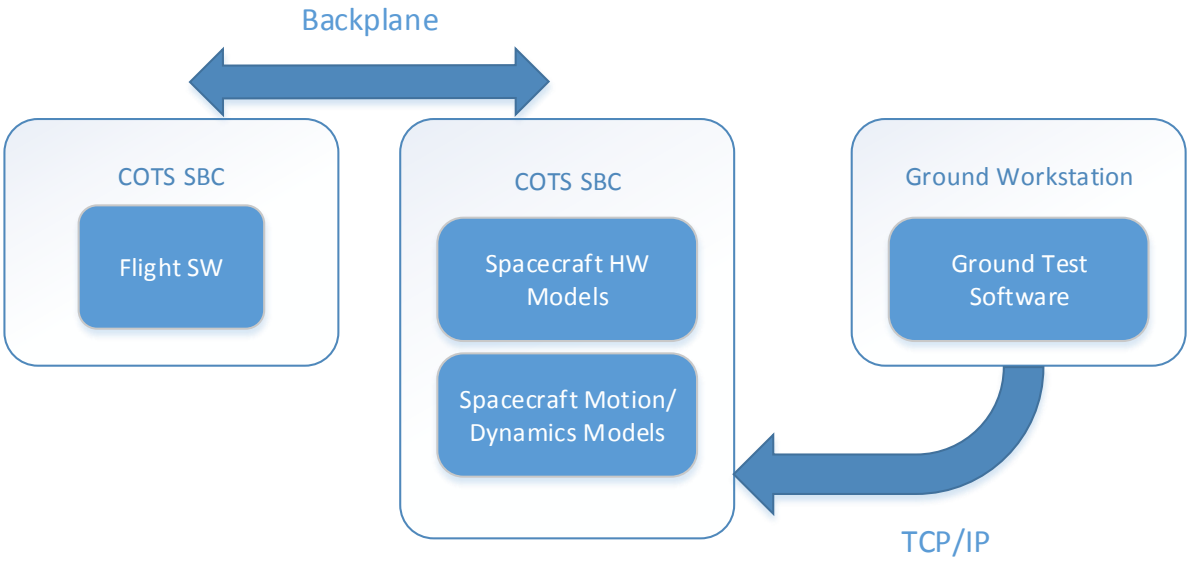

Figure 5: COTS Test Station Top Level Diagram

The advantages of the higher-throughput test stations are:

- Allows developers and testers to validate software and scripts to iron out any bugs before performing final validation
- Cost effectiveness, resulting in significant increase of the number of test stations that can be purchased
- Allows rapid test data collection due to 20x faster than real time capability to create more scalability in the test infrastructure.

2.2 SSL Flight Software Hybrid Testing Framework

Any type of testing framework can be implemented in the agile environment. However, short iteration cycles and rapidly changing requirements results in additional challenges to maintain the test automation suite. In the agile environments, testing plays a crucial role through the different phases of iterations. It involves continuous integration, unit testing, and constant regression testing. Agile is difficult to accomplish using testing frameworks if the test automation suite does not keep up with the pace of the development. SSL designed a set of tools that allows fast updates and easy maintenance of the test scripts and test data for each iteration.

Achieving maximum code and functionality coverage in the rapid requirements iteration cycle creates significant challenges using testing frameworks. SSL hybrid testing framework is developed as a modular, layered architecture that maximizes code reuse of heritage products, controls all changes to heritage baseline software, manages the development of new products, and ensures the integrity of the resulting product.

The components of the SSL hybrid testing framework (Fig. 6) are:

- Centralized system for version control, tracking inputs and test anomalies
- User interface to allow for the management and creation of individual test cases, including parametric pass/fail criteria

- A suite of tools for test data post processing, analysis and test report generation
- A system for test case execution and results data capture
- Common test results review user interface (UI) including common plotting tool that allows overlay comparison of the test results and truth model
- Programmatic interfaces to data repositories

Flight Software Validation Process

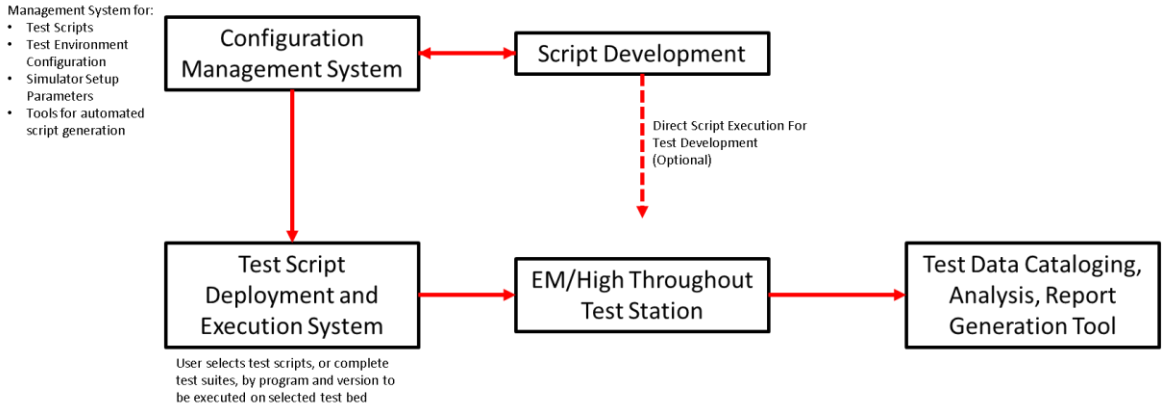

Figure 6: Simplified SSL Hybrid Testing Framework

The centralized system for version control (Fig. 7) provides a layer that can track versions of system components utilized in test cases across multiple version control entities (CVS, SVN, Git, etc.). This system is responsible for detecting when new versions of modules such as flight software, test scripts, or simulation software have been created and can automatically run regressions or mark as out of date test cases that are dependent on the changed modules. This system can also track failed test cases and generate detailed timelines of which versions encountered specific defects and in which versions those defects were resolved.

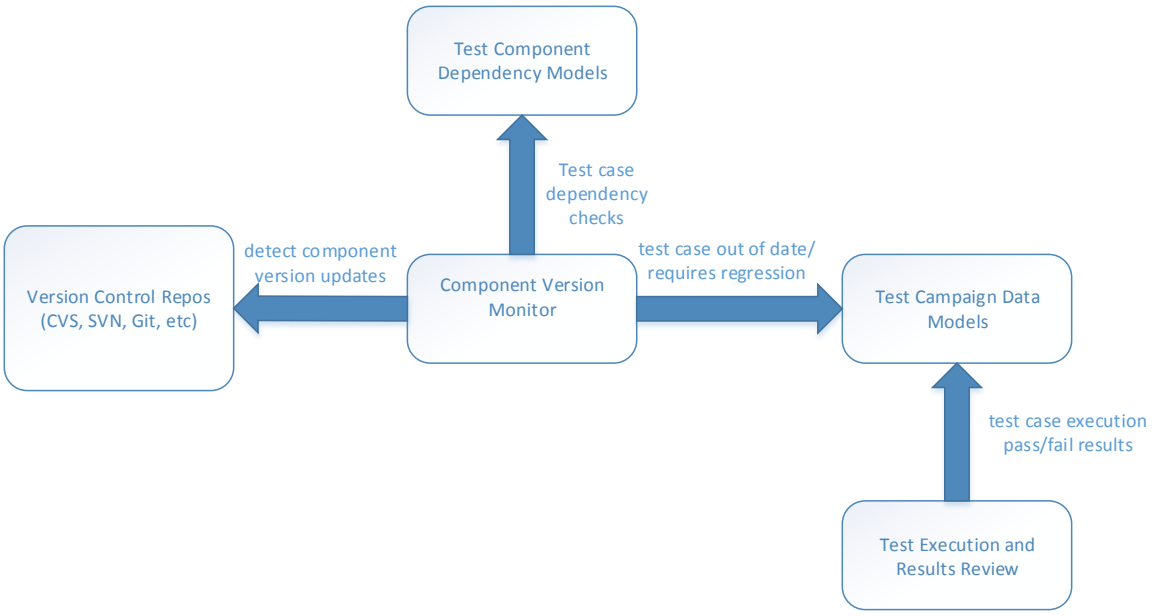

Figure 7: Centralized System for Version Control

The test case management system (Fig. 8) provides a UI for creation of new test cases and management of test components required to execute the test (such as the name of the test script file). This system also allows the user to submit pass/fail criteria information and define how the results should be presented to the user (overlay plots, system event timelines, etc.).

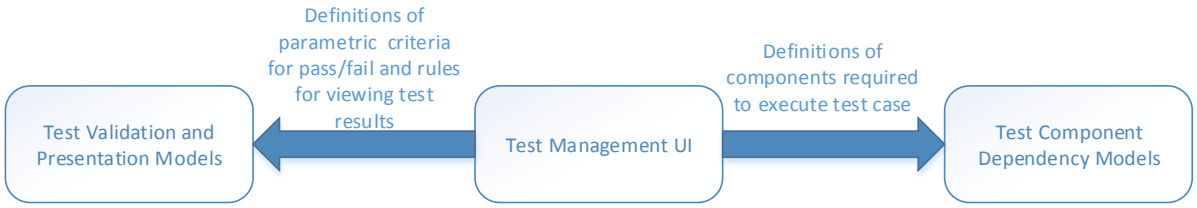

Figure 8: Test Case Management System

The test executive (Fig. 9) provides capability for individual test case or groups of tests to be submitted for batch execution in the test environment. This includes managing and allocating available lab hardware, properly setting up the test harness and configuration for each specific test case under execution, capturing results data including dynamic data series and timestamped system events, and storing the captured data to the test results database.

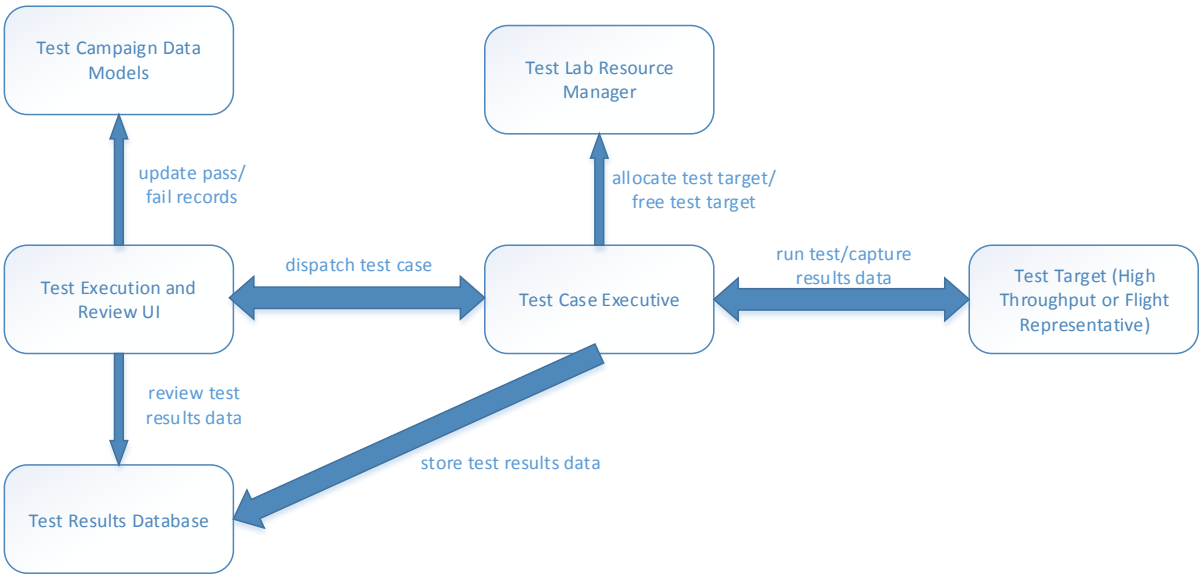

Figure 9: Test Executive

Test data post processing tools allow for the accumulation of automated test results for subsequent analysis. As more test data is collected across many different systems, human-in-the-loop test data review may be decreased by employing pattern recognition algorithms that can perform auto-validation of test results. Post processing may also include tools that generate information that allows for more efficient presentation of the test results to a human reviewer. A tool that automatically time-aligns data samples from a test case with samples from a predict data set is an example. Post processing tools are also used to generate automated test reports to support deliverables to external organizations as well as customers. All test results are kept in repositories that have programmatic query Application program interfaces (APIs) to support easy access for custom analysis. This facilitates future data-mining activities that may need to be performed for an unforeseen reason. An example may be a spacecraft anomaly investigation that requires sophisticated analysis of system performance data captured during system development and test phases.

The test results review UI (Fig. 10) provides a user-friendly means for test engineers internal to the software organization as well as external organizations such as Systems Engineering to review test case results and provide pass/fail assessment. The common plotting tool generates time-aligned overlay plots of specific data points in the test case with predicted profiles. This interface is optimized for human-in-the-loop analysis of system performance as compared to a model-generated truth dataset. Predicted (truth) profiles can be generated from higher level design models (such as dynamic simulations in Matlab), from a previous test instance in the case of regression test, from a similar test performed on a different spacecraft, or a combination of these sources.

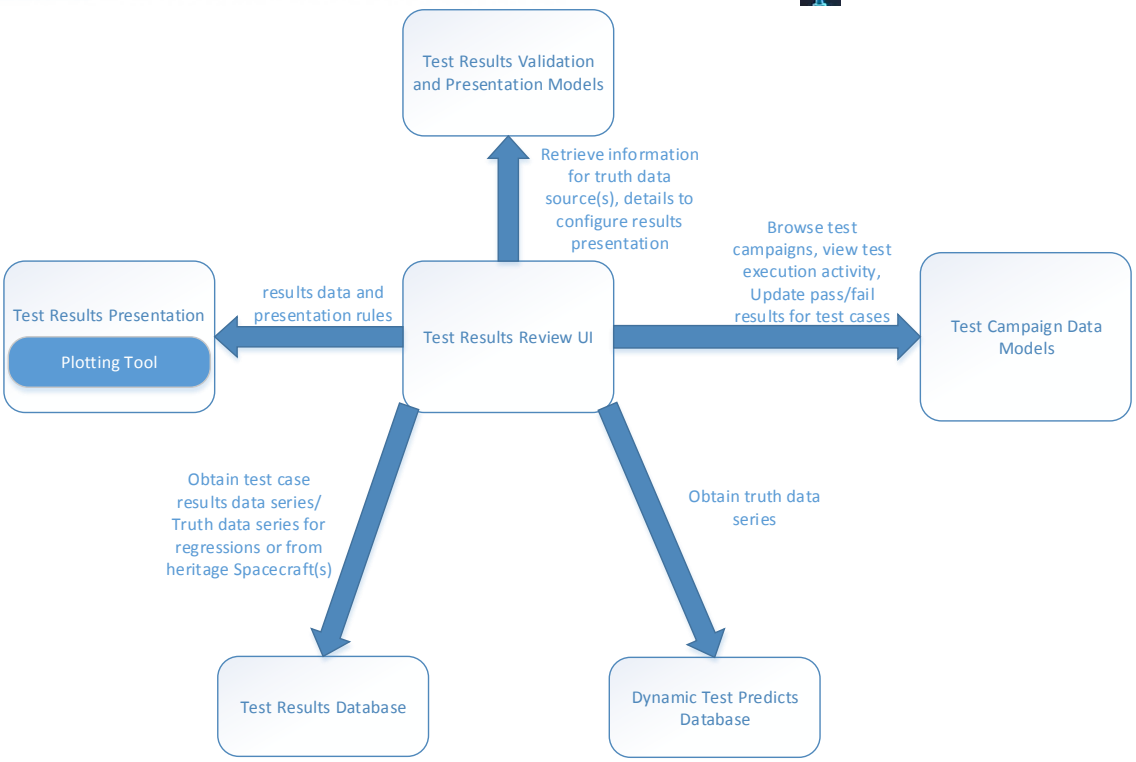

Figure 10: Test Results Review UI

3 CONCLUSION

SSL flight software engages in a cyclical test-fix-test process that subjects the code to increasing levels of functional simulation of spacecraft operating system and application interaction. The process successively adds levels of stress to the software performance evaluation by starting with the application logic but then adds in and verifies the telemetry and command (T&C) interfaces and data bus communication protocols. Addition of more hardware models into the test loop raises the complexity of the test environment to increasingly flight-like conditions and documents the maturation of the code modules as implementation progresses. Final exit from the development test-fix-test loop occurs after successful performance in stress tests with the actual hardware wherever and whenever possible. Where actual hardware is not possible or practical, the most flight-like engineering model hardware and simulator test fixtures are available.
SSL Flight Software Development and Validation Workflow Management System provides the following key attributes:

- Maximizes data organization, ease of navigation and transparency
- Creates traceability by formalizing and tracking quality of inputs
- Improves interfaces to organizations that consume flight software validation reports
- Creates centralized system for version control and tracking inputs required for flight software development and validation
- Automates validation reporting

This rigorous verification and validation process used by the SSL Flight Software Development and Validation Department yields software products that have undergone both a full validation with a high-fidelity testbed and spacecraft-level test resulting in increased quality.

REFERENCES

1. T. Weilkiens, J. G. Lamm, S. Roth, M. Walker; 2016; *Model-Based System Architecture*; First Edition. John Wiley & Sons, Inc.
2. K. Schwaber, M. Beedle; 2002; *Agile Software Development with Scrum*, Prentice Hall

3. M. Monteleone; 2017; "A Proposal for an Agile Development Testing V-Model"; http://www.modernanalyst.com/Resources/Articles/tabid/115/ID/1967/A-Proposal-for-an-Agile-Development-Testing-V-Model.aspx
4. E. Johnson; 2014; "How to implement the Agile-Waterfall Hybrid | V-model vs Waterfall"; www.intland.com/blog/agile/how-to-implement-the-agile-waterfall-hybrid
5. R. Black, 2017; "Agile V Model: Oxymoron or Best Practice?"; www.linkedin.com/pulse/agile-v-model-oxymoron-best-practice-rex-black
6. Software Testing Help; 2017; "Most Popular Test Automation Frameworks with Pros and Cons of Each – Selenium Tutorial"; http://www.softwaretestinghelp.com/test-automation-frameworks-selenium-tutorial-20/