Enhancing the Test and Evaluation Process

Incorporating Agile Development, Test Automation, and MBSE Concepts

Joshua Walker

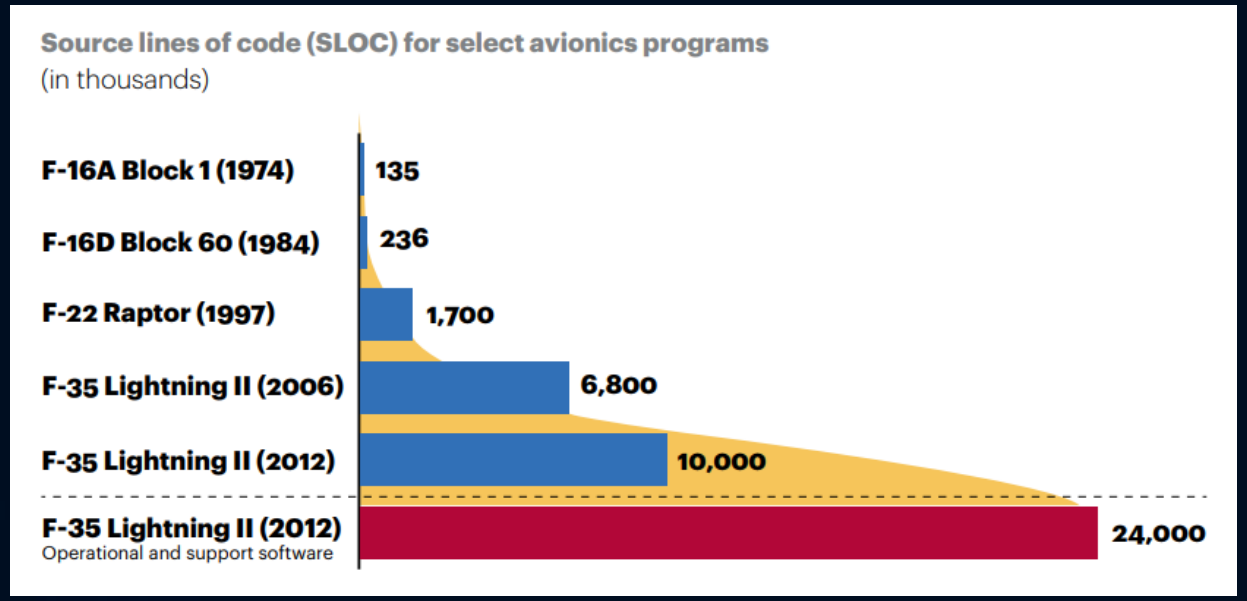
ITEA Symposium 2020
September 17, 2020
3:00-3:30pm

Agenda

- Introduction
- Statement of the Problem
- System Description
- Potential Solution
- Agile Transition
- Automated Testing
- MBSE Concepts
- Results
- Lessons Learned

Introduction

- As of 2007, “data from industry show that the size of the software for various systems and applications has been growing exponentially for the past 40 years“.[1]
  - F-35 fighter and support software totals over 24 million source lines of code (SLOC) as of 2012.[2]
- Test efforts account for 75-88 percent of total software development costs on large aerospace projects.[3]
- Exponential growth directly flows into the size and complexity of system test procedures.[4]



Statement of the Problem

- Increasing system complexity leads to test-related issues.
  - Long test execution times[5]
  - Confusing test procedures
  - Repetitive/duplicate test procedures
  - Incomplete requirements tracing[6]
  - Inability to focus regression testing on specific functionality[7]
- Overall product development is impacted.
  - Inability to quickly release new updates[8]
  - Inability to adapt to design changes[9]
  - Release of inadequately tested products[10]
  - Wasted test effort for components that did not change[11]
  - Long training times for engineers new to the system

Statement of the Problem

- The **need** and **opportunity** for optimization and innovation exists within the Test and Evaluation domain.

- Evidenced by:
  - Emergence of automated testing frameworks
  - Creation of iterative testing methodologies
  - Directed expansion and application of related concepts

System Description

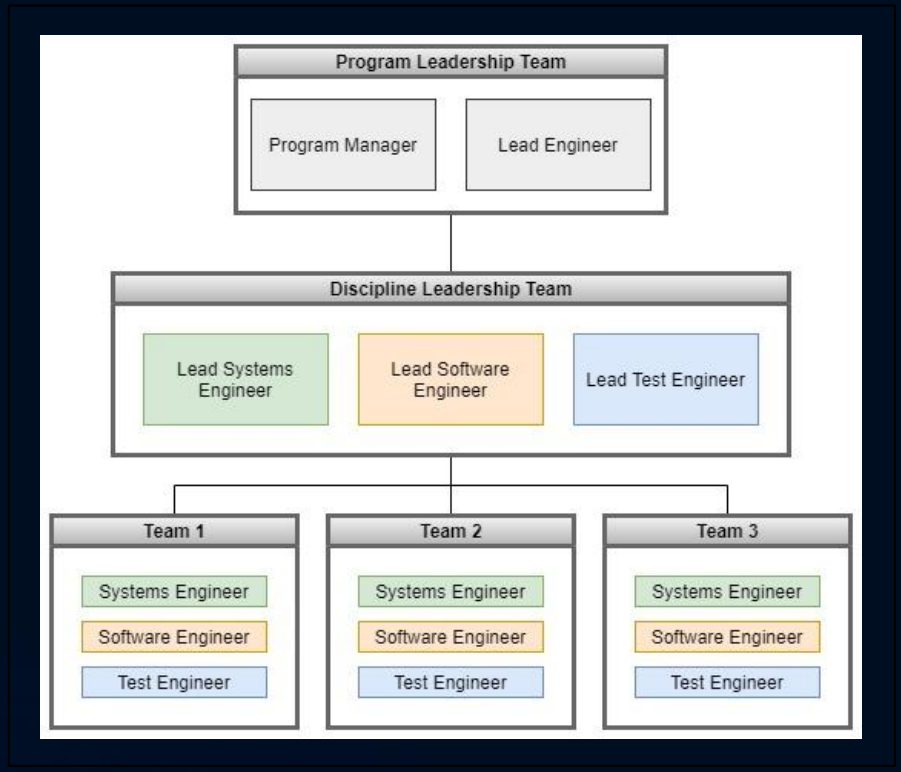
- Multi-Platform Embedded Electronic Warfare Management System
  - Interfaces with onboard and offboard aircraft systems
  - Receives and processes the threat environment
  - Provides the pilot/crew with a response solution for aircraft survivability
- Historical Development Process
  - Developed using a Waterfall development life cycle
  - Multiple parallel efforts (3-18 months) for each platform
- Challenges
  - Limited regression testing for smaller releases
  - Test execution times for full releases up to 18 weeks

Potential Solution

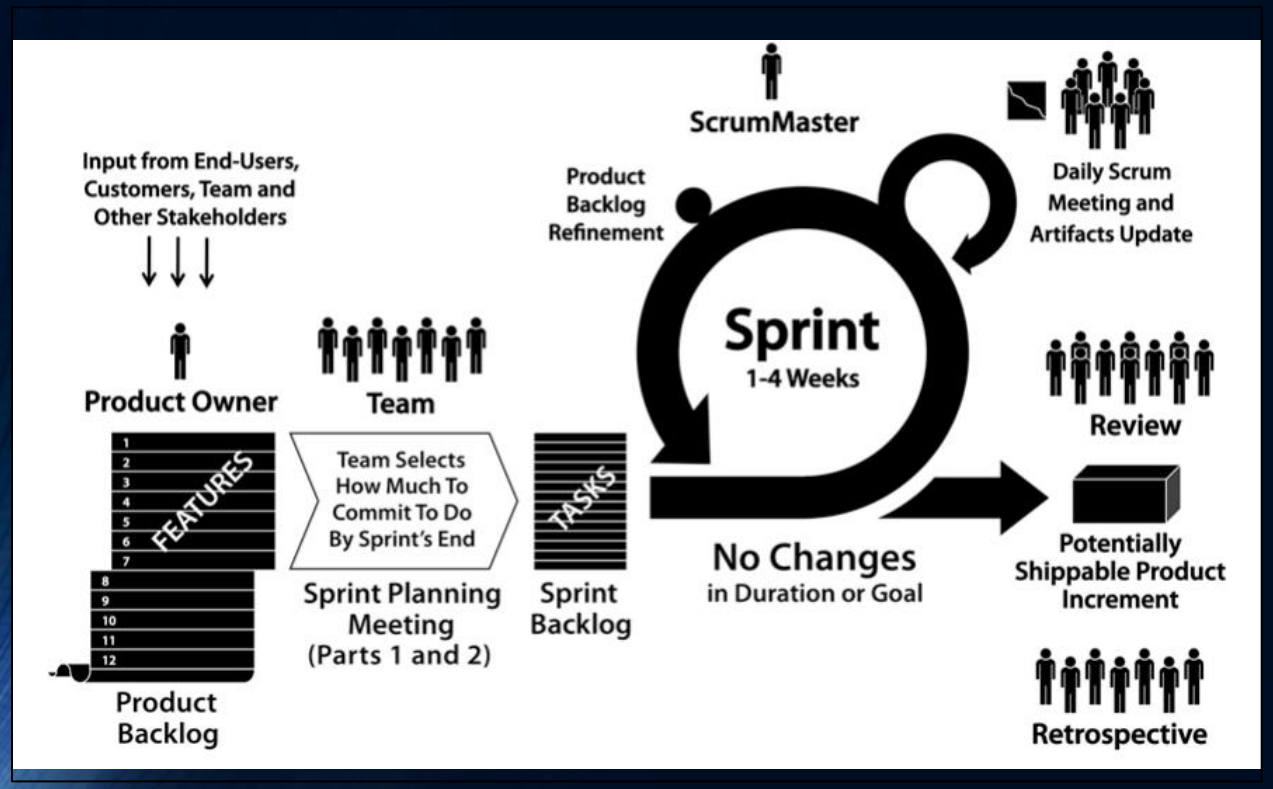
- Transition to an Agile development methodology
  - Restructures development process to complete features iteratively
  - Provides the ability to adapt to changing customer requests quickly[12]
- Implement automated testing
  - Decreases test execution times
  - Supports Agile process by rapidly testing prototypes[13]
- Apply Model-Based Systems Engineering (MBSE) concepts
  - Provides standardization of documentation to support regression testing
  - Helps manage and communicate the scope of necessary regression testing[14]

Agile Transition – Prerequisites

- Product Reorganization
  - Consolidate multiple system variants into a common releasable product
    - Document commonality of features among all platforms
    - Merge code that should be shared
    - Unify interface menu structures and displays
- Release Timelines
  - Align variant schedules to a single, yearly release cycle
  - Combine regression testing efforts to support all platforms
- Development Tools
  - Choose productivity tools that support Agile development
- Agile Training
  - Ensure strong leadership in Agile methodologies exists
  - Provide Agile training to team members
- Team Organization
  - Restructure from discipline-specific to cross-functional teams



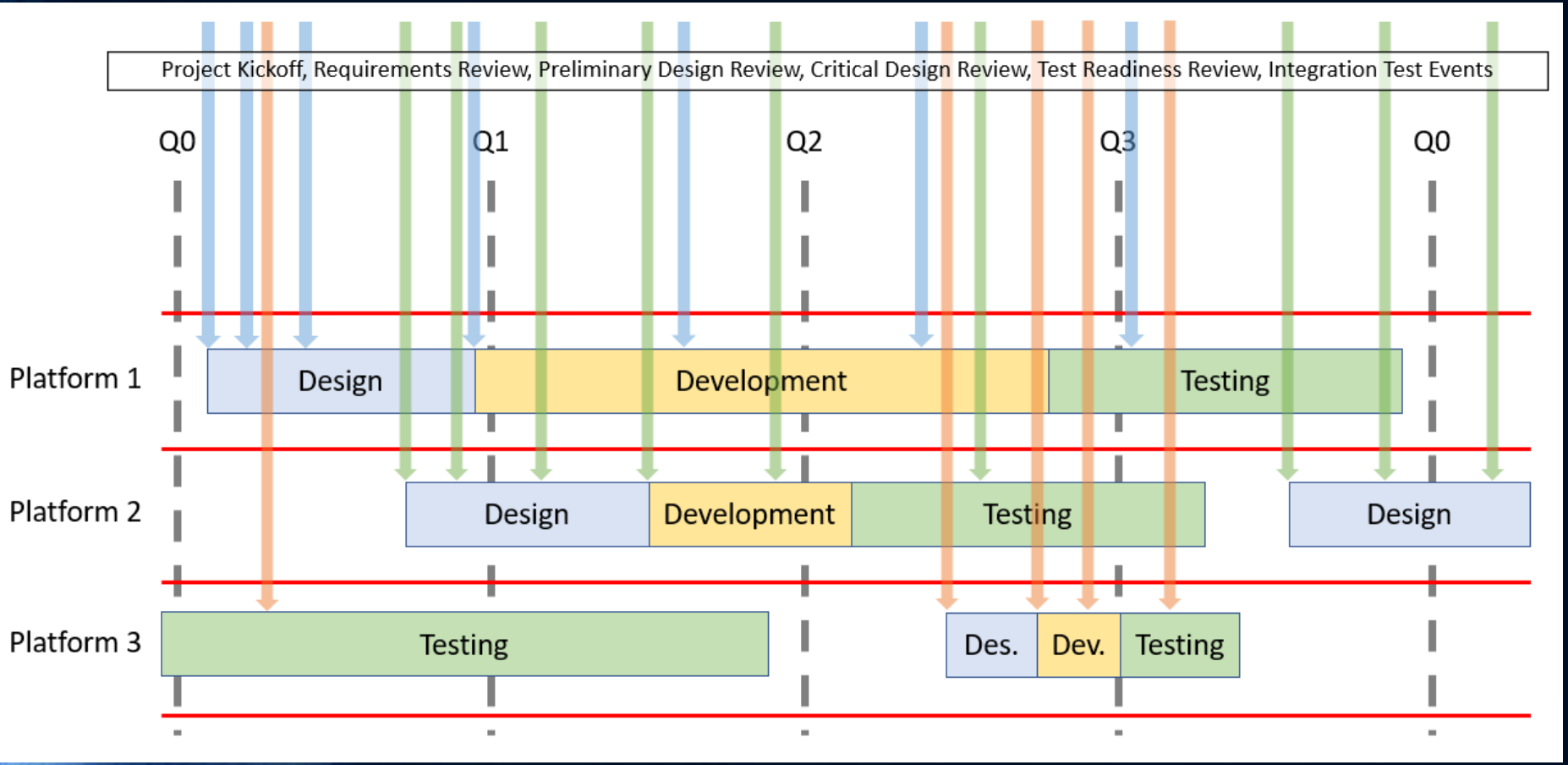
Agile Transition – New Process



Traditional Scrum Process[15]

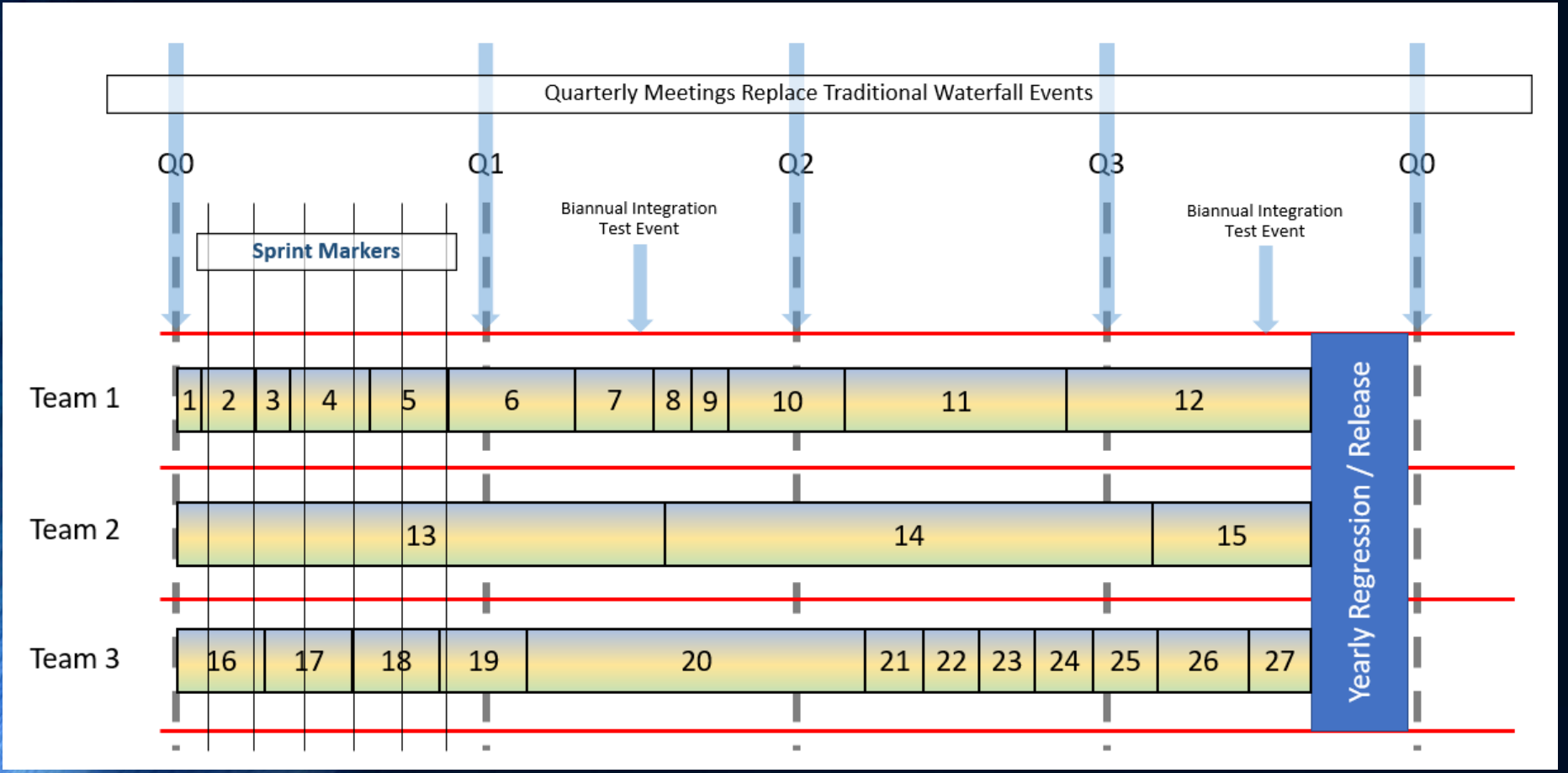
- Customized Scrum Process
  - Maintains most aspects of traditional Scrum, with some changes to support uniqueness of system and existing processes
- Main Differences
  - Scrum Master responsibilities shared by Lead Engineer and Discipline Leads
  - Multiple teams working independently from a shared backlog
  - Single Retrospective and Review meetings held with all teams
  - Weekly Backlog Refinement meetings used to prework future tasks
  - Yearly Regression period (planned for 6 weeks) held before final release to customer

Agile Transition – Original Waterfall Schedule



- Constant Customer Events
- All Features complete after Testing Phase
- Downtime for teams between platform releases
- High switching costs for team members

Agile Transition – New Agile Schedule

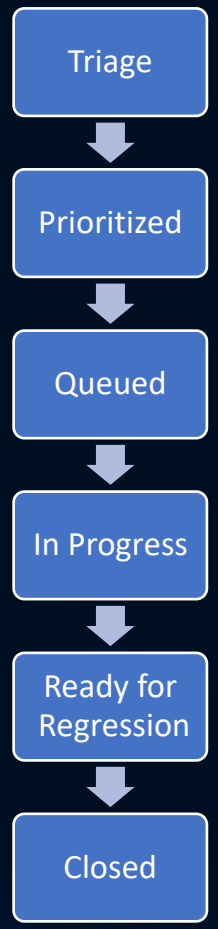


Numbered blocks = Epics

- Decreases Customer Events
- Provides opportunity for quarterly feedback/refocus
- Reduces downtime for teams
- Reduces task transition times

Agile Transition – Task Workflow

- Each Epic corresponds to a Feature/Change Request tracked by the customer.
  - Large additions that may take multiple Sprints to accomplish
  - Epics divided into Tasks; Tasks divided into Subtasks
    - Tasks accomplishable in a single Sprint
    - Subtasks required to finish the Task
  - Epics discussed and reprioritized at each Quarterly Meeting
    - Quarterly customer review to shift development focus mid-cycle

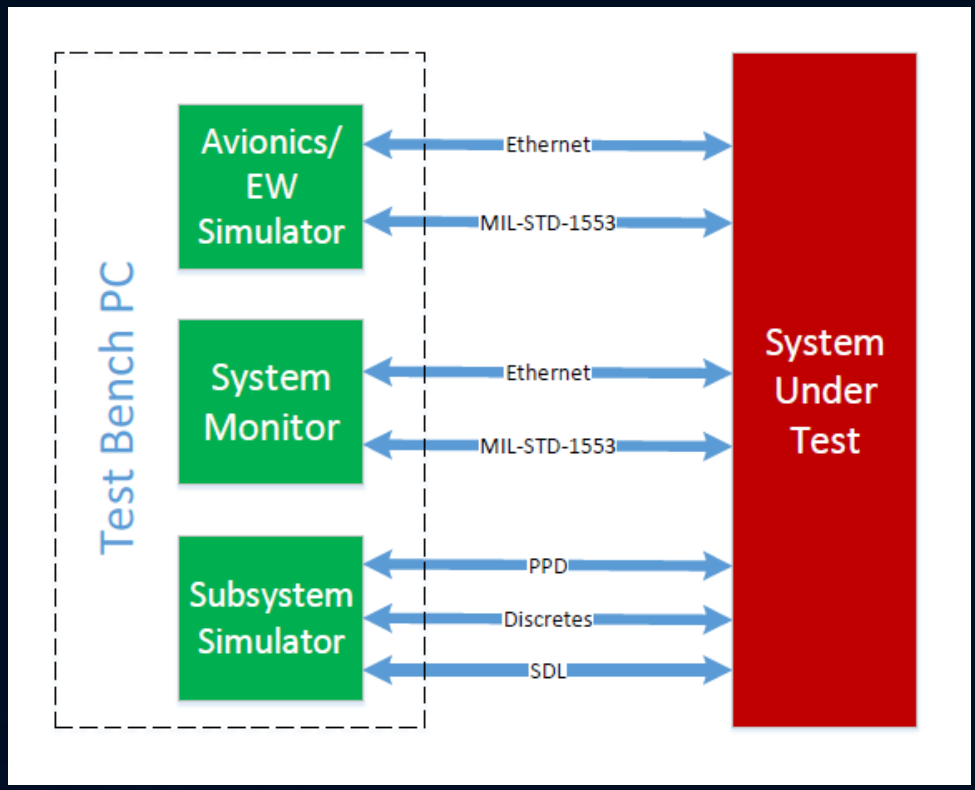


- Triage
  - Item has been submitted.
- Prioritized
  - Item has been prioritized against other items.
- Queued
  - Item has been refined enough to be assigned to a team's Sprint.
- In Progress
  - Item is being worked by a team.
- Ready for Regression
  - Item is completed and waiting to be formally tested during the Yearly Regression period.
- Closed
  - Item has been released.

Automated Testing – Initial Strategy

- Utilize existing functionality and components as much as possible to:
  - Minimize costs
  - Limit the impact to the system under test
  - Maintain the ability to execute existing manual testing
  - Provide an easier transition to automated testing
  - Provide automated testing capabilities faster

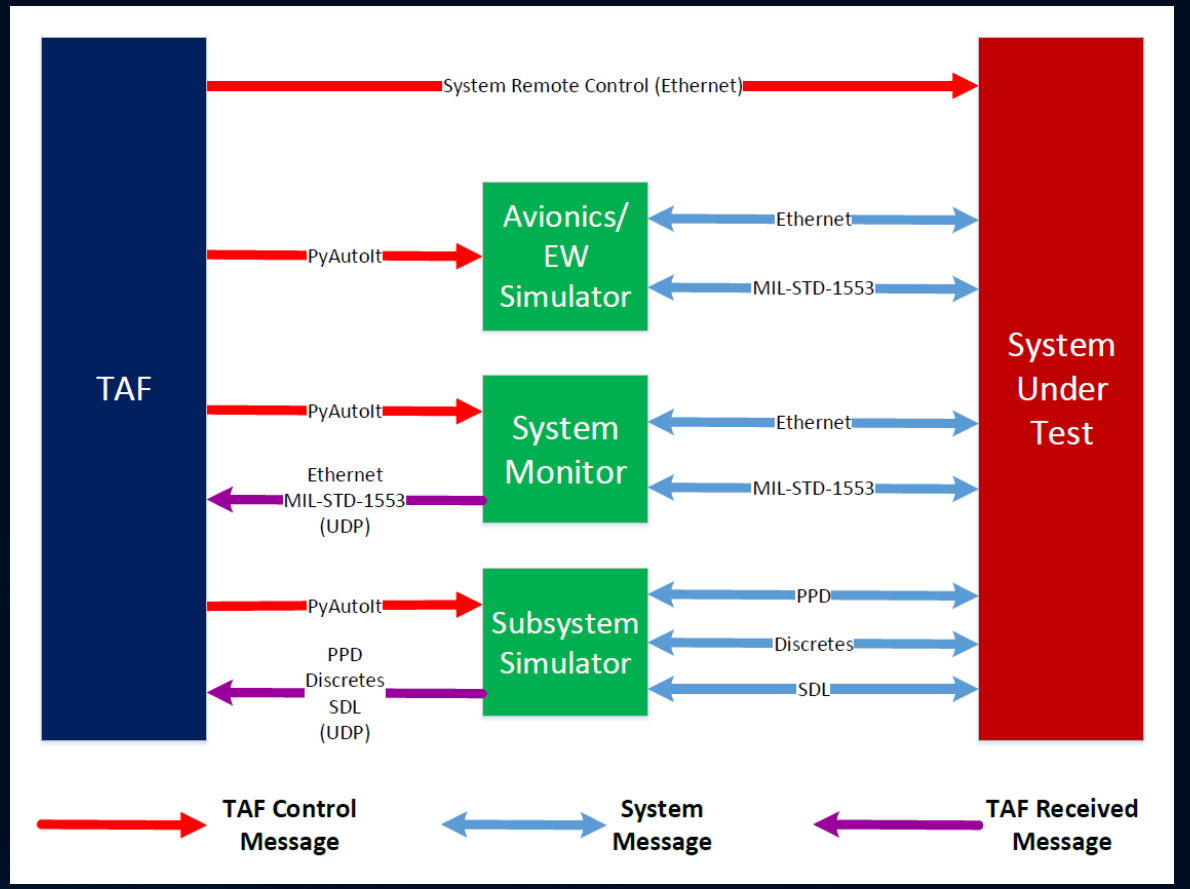
System Test Environment



Automated Testing – TAF Overview

- Test Automation Framework (TAF)
  - Uses open source Python module Robot Framework for core automated testing capabilities
  - Provides GUI-based control and embedded system control through developed libraries
  - Allows test steps to be written in natural language and translated into code
  - Calls Robot Framework “Keywords” as test steps (i.e., an action with input parameters)

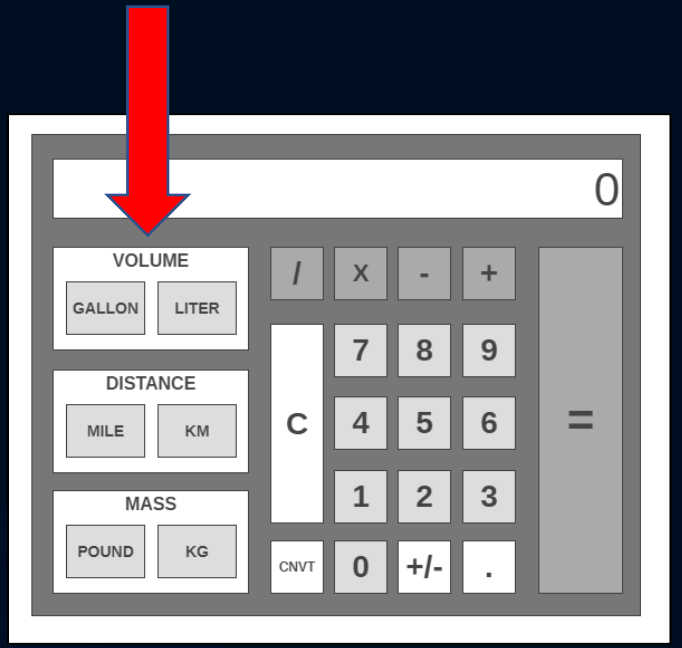
Automated Test Environment



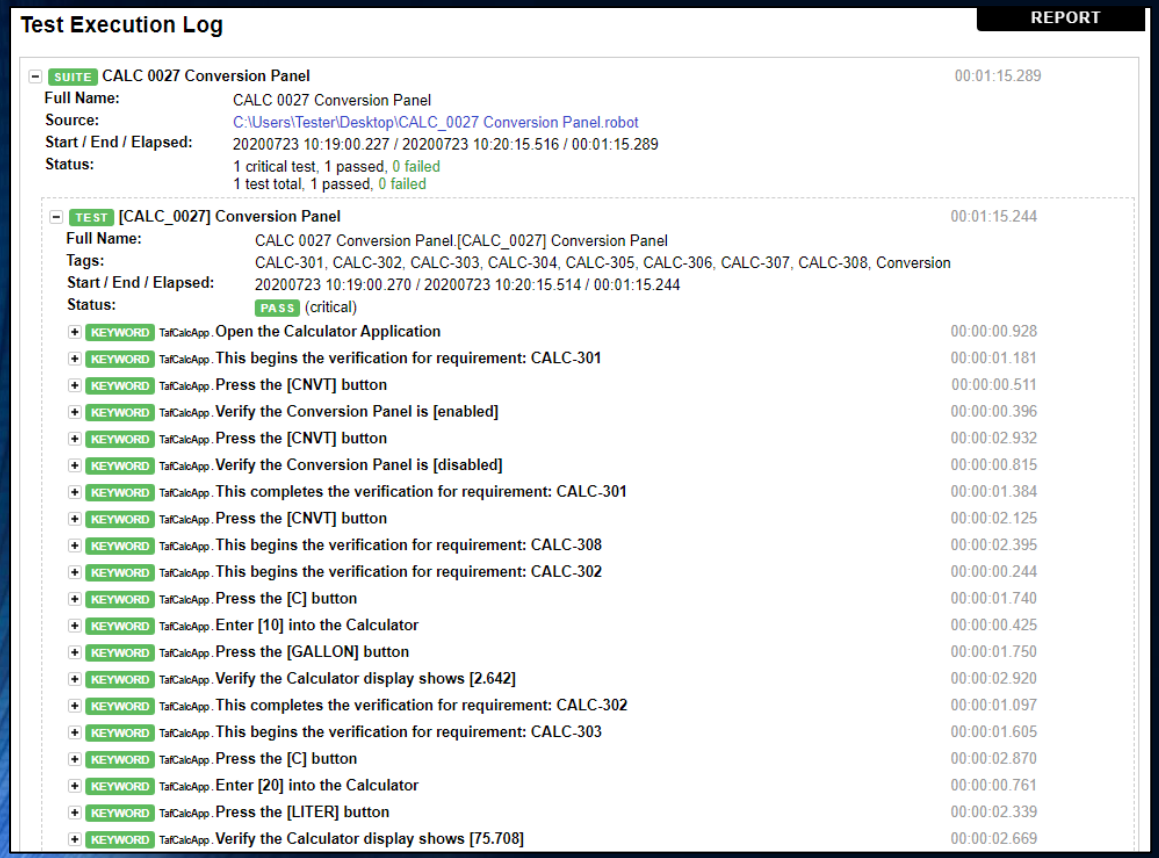
Automated Testing – TAF Test Case Example

```
*** Settings ***
Library TafCalcApp

*** Variables ***
@{REQUIREMENTS} CALC-301    CALC-302    CALC-303    CALC-304
...             CALC-305    CALC-306    CALC-307    CALC-308

*** Test Cases ***
[CALC_0027] Conversion Panel
    [tags]  @{REQUIREMENTS}     Conversion
    #The purpose of this test case is to open the Conversion Panel
    #and verify that each conversion button works as intended.

    Open the Calculator Application
    This begins the verification for requirement: CALC-301
    #Open the Conversion Panel
    Press the [CNVT] button
    Verify the Conversion Panel is [enabled]
    #Close the Conversion Panel
    Press the [CNVT] button
    Verify the Conversion Panel is [disabled]
    This completes the verification for requirement: CALC-301

    #Reopen the Conversion Panel
    Press the [CNVT] button

    This begins the verification for requirement: CALC-308

    #Convert to Gallons
    This begins the verification for requirement: CALC-302
    Press the [C] button
    Enter [10] into the Calculator
    Press the [GALLON] button
    Verify the Calculator display shows [2.642]
    This completes the verification for requirement: CALC-302

    #Convert to Liters
    This begins the verification for requirement: CALC-303
    Press the [C] button
    Enter [20] into the Calculator
    Press the [LITER] button
    Verify the Calculator display shows [75.708]
    This completes the verification for requirement: CALC-303

    #Convert to Kilometers
    This begins the verification for requirement: CALC-304
    Press the [C] button
    Enter [30] into the Calculator
    Press the [KM] button
    Verify the Calculator display shows [48.28]
    This completes the verification for requirement: CALC-304

    #Convert to Miles
    This begins the verification for requirement: CALC-305
    Press the [C] button
    Enter [40] into the Calculator
    Press the [MILE] button
    Verify the Calculator display shows [24.855]
    This completes the verification for requirement: CALC-305

    #Convert to Kilograms
    This begins the verification for requirement: CALC-306
    Press the [C] button
    Enter [50] into the Calculator
    Press the [KG] button
    Verify the Calculator display shows [22.68]
    This completes the verification for requirement: CALC-306

    #Convert to Pounds
    This begins the verification for requirement: CALC-307
    Press the [C] button
    Enter [60] into the Calculator
    Press the [POUND] button
    Verify the Calculator display shows [132.277]
    This completes the verification for requirement: CALC-307

    This completes the verification for requirement: CALC-308

    Close the Calculator Application
    [TEARDOWN]  Tear down the test case
```

- Simple test case to exercise conversion functions on a calculator application



Automated Testing – TAF Output Example



Test Execution Log

REPORT

SUITE CALC 0027 Conversion Panel 00:01:15.289

Full Name: CALC 0027 Conversion Panel
Source: C:\Users\Tester\Desktop\CALC_0027 Conversion Panel.robot
Start / End / Elapsed: 20200723 10:19:00.227 / 20200723 10:20:15.516 / 00:01:15.289
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

TEST [CALC_0027] Conversion Panel 00:01:15.244

Full Name: CALC 0027 Conversion Panel.[CALC_0027] Conversion Panel
Tags: CALC-301, CALC-302, CALC-303, CALC-304, CALC-305, CALC-306, CALC-307, CALC-308, Conversion
Start / End / Elapsed: 20200723 10:19:00.270 / 20200723 10:20:15.514 / 00:01:15.244
Status: PASS (critical)

KEYWORD	TafCalcApp. Open the Calculator Application	00:00:00.928
KEYWORD	TafCalcApp. This begins the verification for requirement: CALC-301	00:00:01.181
KEYWORD	TafCalcApp. Press the [CNVT] button	00:00:00.511
KEYWORD	TafCalcApp. Verify the Conversion Panel is [enabled]	00:00:00.396
KEYWORD	TafCalcApp. Press the [CNVT] button	00:00:02.932
KEYWORD	TafCalcApp. Verify the Conversion Panel is [disabled]	00:00:00.815
KEYWORD	TafCalcApp. This completes the verification for requirement: CALC-301	00:00:01.384
KEYWORD	TafCalcApp. Press the [CNVT] button	00:00:02.125
KEYWORD	TafCalcApp. This begins the verification for requirement: CALC-308	00:00:02.395
KEYWORD	TafCalcApp. This begins the verification for requirement: CALC-302	00:00:00.244
KEYWORD	TafCalcApp. Press the [C] button	00:00:01.740
KEYWORD	TafCalcApp. Enter [10] into the Calculator	00:00:00.425
KEYWORD	TafCalcApp. Press the [GALLON] button	00:00:01.750
KEYWORD	TafCalcApp. Verify the Calculator display shows [2.642]	00:00:02.920
KEYWORD	TafCalcApp. This completes the verification for requirement: CALC-302	00:00:01.097
KEYWORD	TafCalcApp. This begins the verification for requirement: CALC-303	00:00:01.605
KEYWORD	TafCalcApp. Press the [C] button	00:00:02.870
KEYWORD	TafCalcApp. Enter [20] into the Calculator	00:00:00.761
KEYWORD	TafCalcApp. Press the [LITER] button	00:00:02.339
KEYWORD	TafCalcApp. Verify the Calculator display shows [75.708]	00:00:02.669

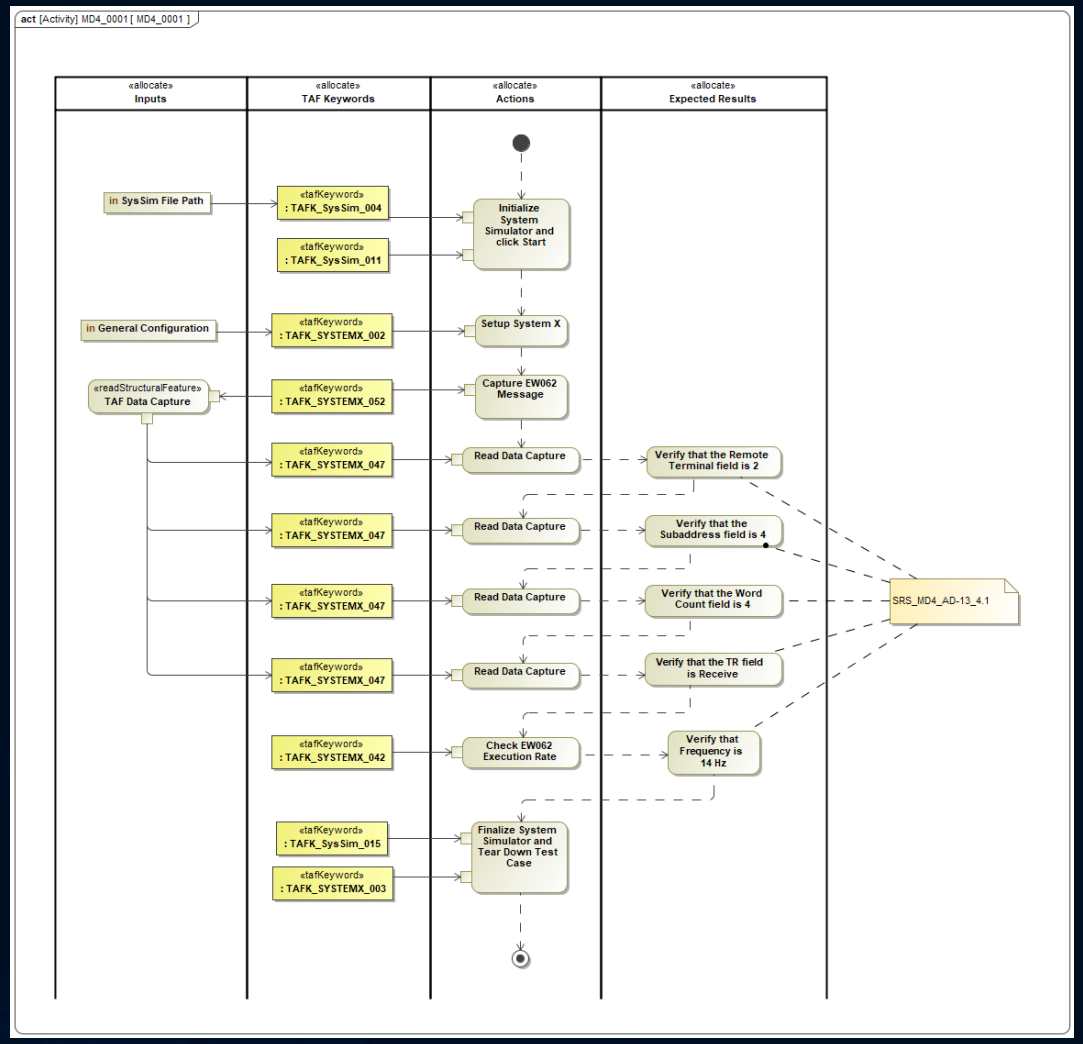
KEYWORD	TafCalcApp. This completes the verification for requirement: CALC-303	00:00:01.173
KEYWORD	TafCalcApp. This begins the verification for requirement: CALC-304	00:00:01.107
KEYWORD	TafCalcApp. Press the [C] button	00:00:02.270
KEYWORD	TafCalcApp. Enter [30] into the Calculator	00:00:02.738
KEYWORD	TafCalcApp. Press the [KM] button	00:00:02.200
KEYWORD	TafCalcApp. Verify the Calculator display shows [48.28]	00:00:01.615
KEYWORD	TafCalcApp. This completes the verification for requirement: CALC-304	00:00:00.904
KEYWORD	TafCalcApp. This begins the verification for requirement: CALC-305	00:00:00.283
KEYWORD	TafCalcApp. Press the [C] button	00:00:01.313
KEYWORD	TafCalcApp. Enter [40] into the Calculator	00:00:01.004
KEYWORD	TafCalcApp. Press the [MILE] button	00:00:01.894
KEYWORD	TafCalcApp. Verify the Calculator display shows [24.855]	00:00:01.058
KEYWORD	TafCalcApp. This completes the verification for requirement: CALC-305	00:00:02.068
KEYWORD	TafCalcApp. This begins the verification for requirement: CALC-306	00:00:02.708
KEYWORD	TafCalcApp. Press the [C] button	00:00:01.272
KEYWORD	TafCalcApp. Enter [50] into the Calculator	00:00:01.242
KEYWORD	TafCalcApp. Press the [KG] button	00:00:02.982
KEYWORD	TafCalcApp. Verify the Calculator display shows [22.68]	00:00:02.365
KEYWORD	TafCalcApp. This completes the verification for requirement: CALC-306	00:00:02.328
KEYWORD	TafCalcApp. This begins the verification for requirement: CALC-307	00:00:01.154
KEYWORD	TafCalcApp. Press the [C] button	00:00:01.121
KEYWORD	TafCalcApp. Enter [60] into the Calculator	00:00:00.565
KEYWORD	TafCalcApp. Press the [POUND] button	00:00:00.662
KEYWORD	TafCalcApp. Verify the Calculator display shows [132.277]	00:00:01.045
KEYWORD	TafCalcApp. This completes the verification for requirement: CALC-307	00:00:02.330
KEYWORD	TafCalcApp. This completes the verification for requirement: CALC-308	00:00:00.967
KEYWORD	TafCalcApp. Close the Calculator Application	00:00:01.129
TEARDOWN	TafCalcApp. Tear down the test case	00:00:02.632

MBSE Concepts – Objectives

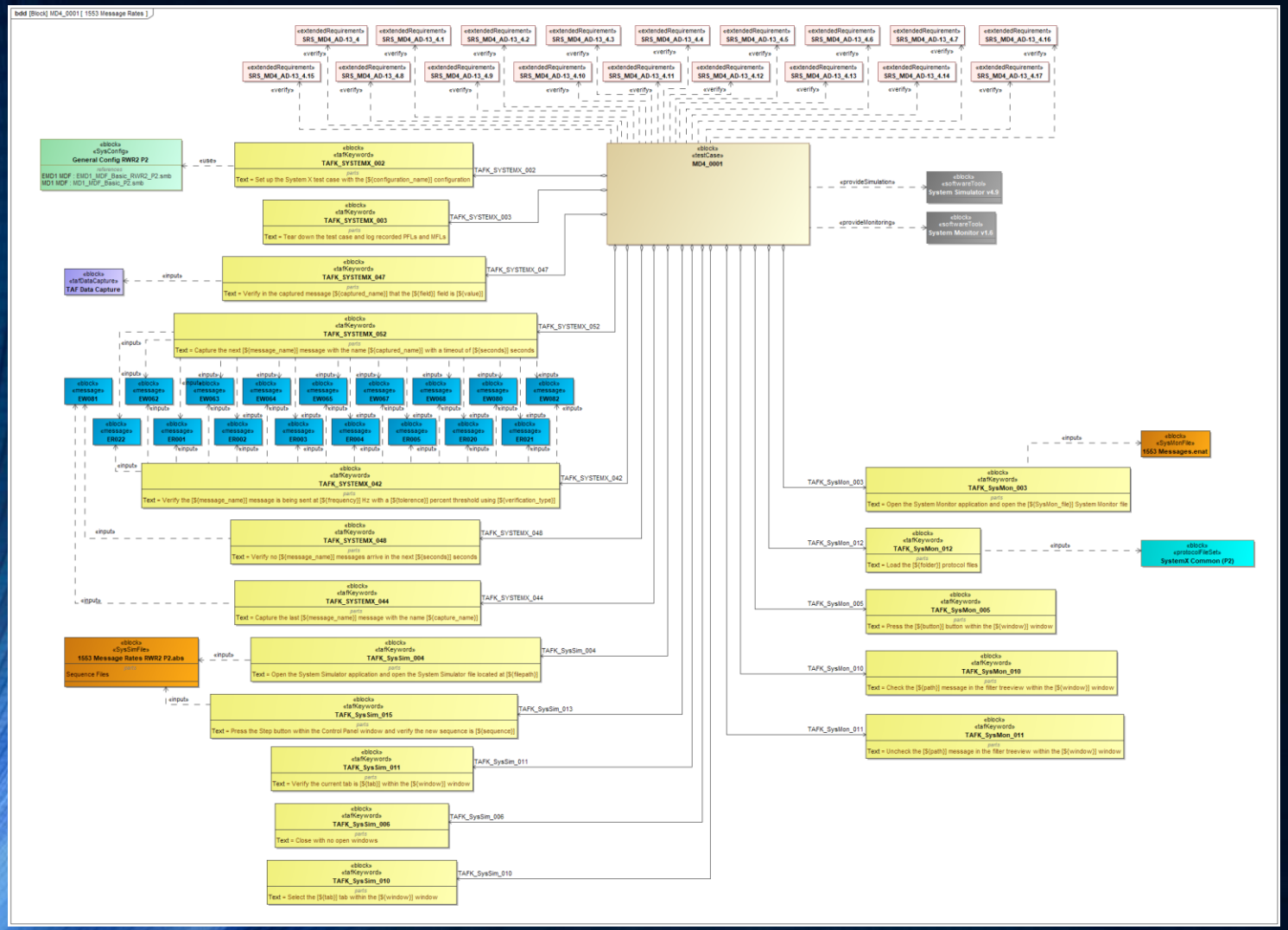
- Main goal is to promote the organization and structure of test artifacts in a standard, formalized way.
  - Improve the structure and clarity of test case documentation
  - Improve rapid comprehension of test procedure content[16]
  - Reestablish a formal method of requirements traceability[17]
  - Provide a method of assessing impact of upstream changes[18]
  - Increase understanding of interdependencies between test cases
  - Facilitate easier review of test planning documentation
  - Provide a method to quickly identify test cases for targeted regression testing
- Process is not a solution for model-based testing, but rather model-based documentation.

MBSE Concepts – Test Planning

- Test Engineer performs Test Planning within the model using an Activity Diagram.
  - Formalizes structure of test plan for easier review and understanding
  - Provides intended flow of the test case to be developed
  - Captures traditional Actions and Expected results tied to Requirements, but also shows expected TAF Keywords and input parameters



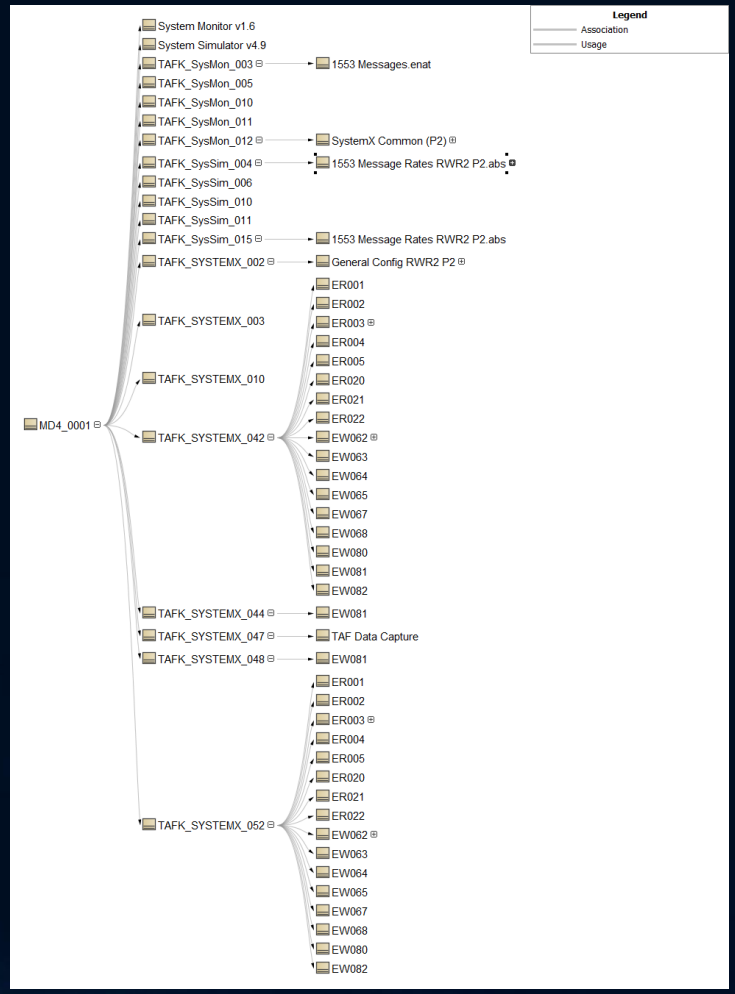
MBSE Concepts – Test Documentation

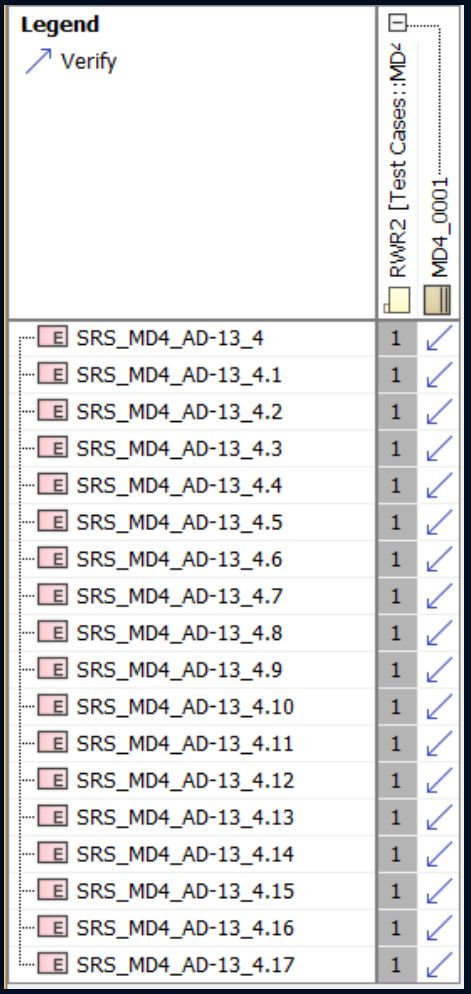


- Test Engineer documents important aspects of the developed test case in a Block Definition Diagram.
  - Provides a view of shared components (e.g., test files, system messages, TAF Keywords, external tools, other inputs) between all test cases
  - Provides the ability to analyze impact of upstream changes

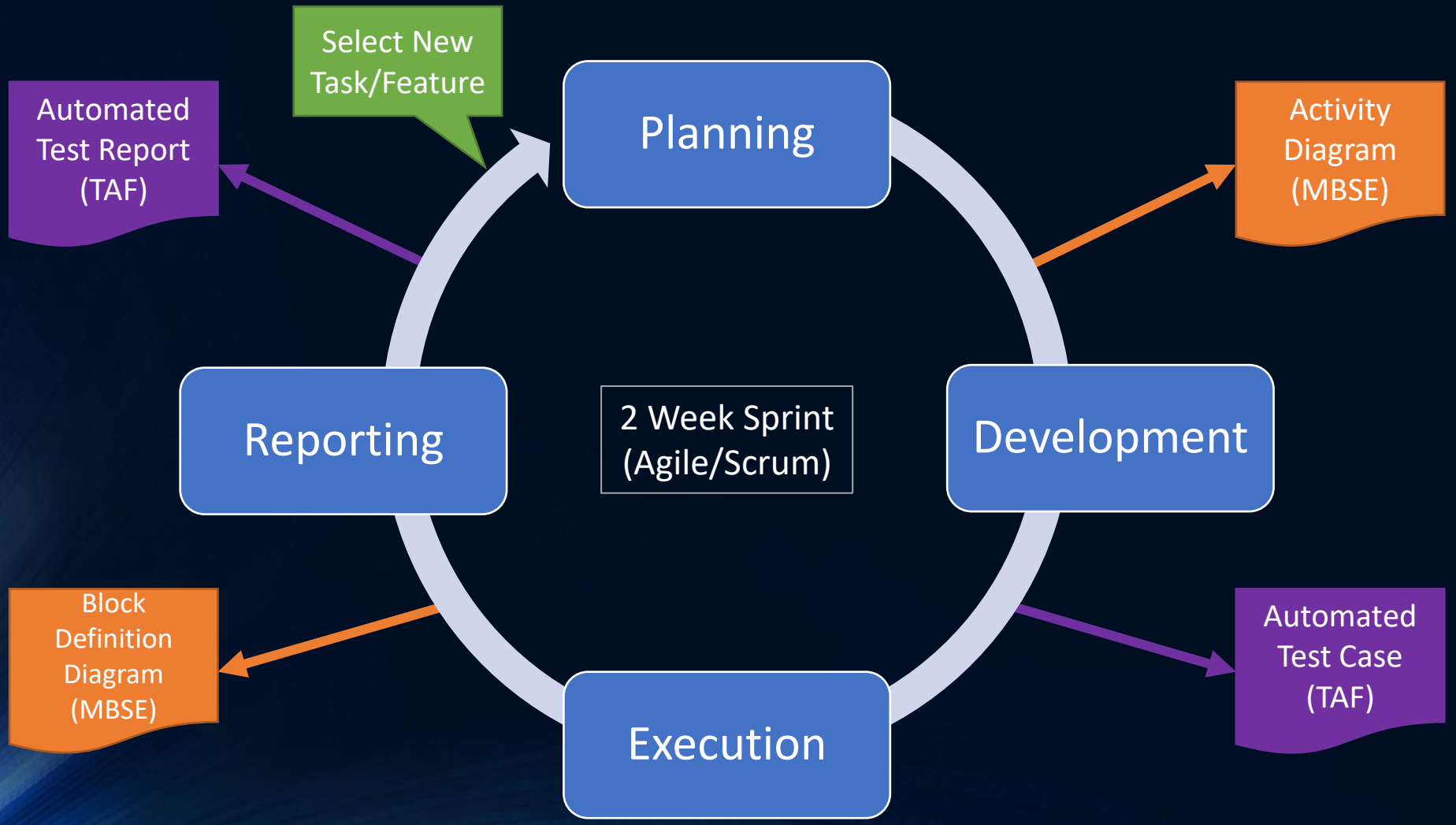
MBSE Concepts – Using the Model

- Test Case Summaries
- Requirements Traceability
- Impact Analysis
  - Relation Map
  - Lookup Tables
  - Dependency Matrices
- Data Export





Test Engineer Activities



Results – Automation Capability

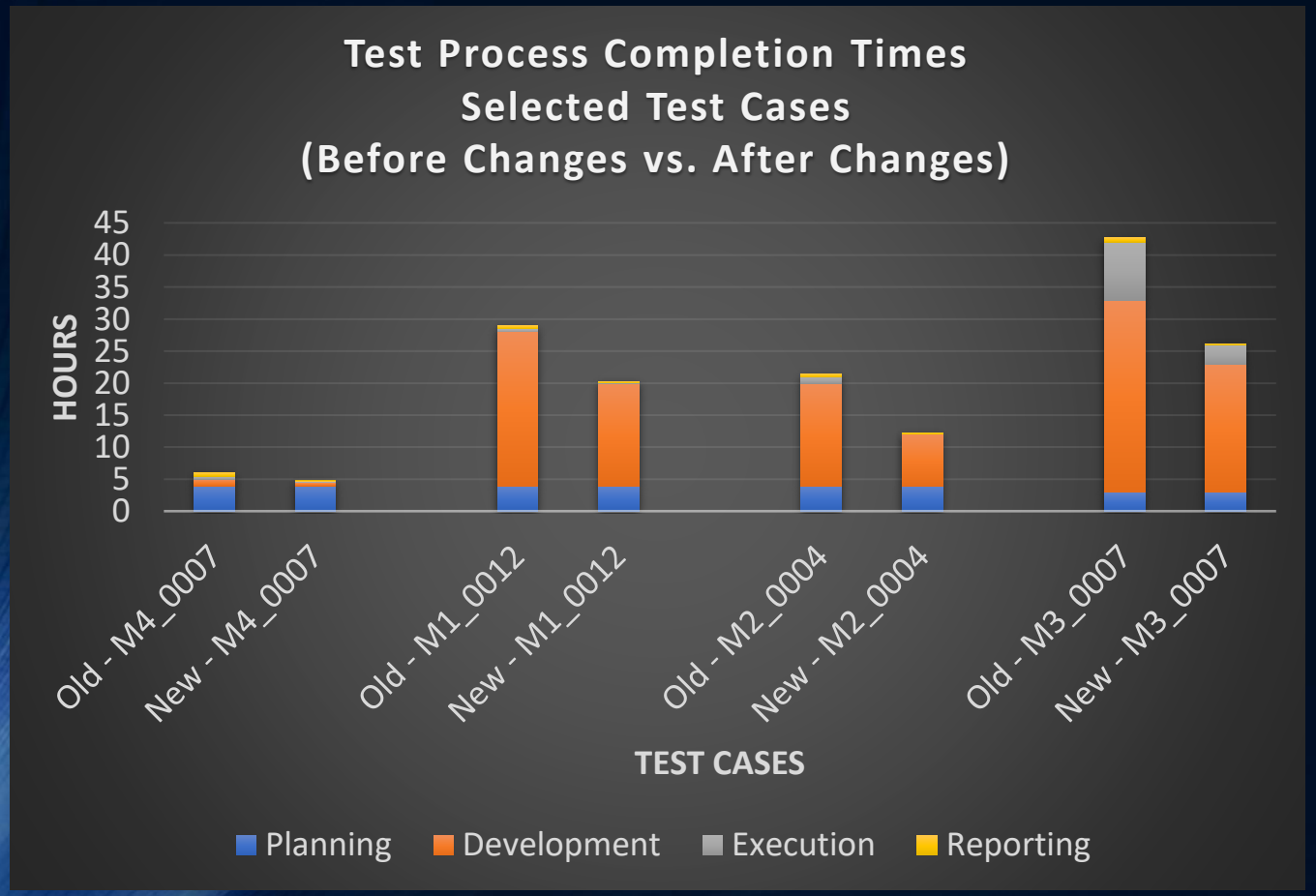
Component	CR1 (2019) Capability	CR2 (2020) Capability	CR1 (2019) Completed	CR2 (2020) Completed
System Tests	75%	85%	5%	20%
Module 1 Tests	90%	95%	5%	15%
Module 2 Tests	75%	75%	20%	50%
Module 3 Tests	95%	95%	75%	75%
Module 4 Tests	75%	90%	50%	80%
Module 5 Tests	100%	100%	5%	6%

- Automated Test Cases Completed
  - Common Release 1 (2019): 76
  - Common Release 2 (2020): 165+
- Three supported platforms included in Common Release
- New features included in both Common Releases (i.e., ongoing development)

Results – Test Execution Time Savings

- Common Release 1 (2019)
  - Reduced Regression Period to 15 weeks
  - Saved approximately 3 weeks with new process
- Common Release 2 (2020)
  - Reduced Regression Period to 12 weeks (estimated)
  - Saved approximately 6 weeks (estimated) with new process
- Mission Data Tool
  - Automated approximately 2,200 pages of test procedures (95%)
  - 2019 Release: 6-week manual test reduced to 2 days
  - 2020 Release: 7-week manual test reduced to 2 days

Results – Test Process Times



- Test Planning – **No Difference**
  - While the planning process is different, proper planning still requires dedicated engineering time.
- Test Development – **Reduced by 42%**
  - New process increases speed of developing test procedures and allows easier reuse of procedures.
- Test Execution – **Reduced by 81%**
  - Automated test procedures execute faster than a human can possibly run them.
- Test Reporting – **Reduced by 69%**
  - Generated test logs are easily transferred to a formal test report.

Results – Defect Identification

- Common Release 1 (2019)
  - 23 new defects identified and accepted in final release
  - 100% of new issues discovered by new process found **before** Test Execution Phase
- Common Release 2 (2020)
  - Similar results expected
  - Regression Phase not completed yet

New process led to defects being found earlier in the development process.

Results – Qualitative

- Test Procedures
  - Improved flow/organization
  - Increased maintainability
  - Reduced confusion
  - Reduced variability of test approach
- Training Times
  - Reduced new engineer training time
  - Increased new engineer productivity
- Product Quality
  - Enabled quicker release of new features
  - Increased confidence of release quality
  - Provided systematic method of impact analysis
  - Identified defects much earlier in the process
- Team Morale
  - Reduced burnout caused by traditional test methods

Lessons Learned

- Agile
  - Be patient and expect to change the process to suit the needs of the product
  - Find opportunities for fun/promoting team comradery
  - Automate as much of the process as possible (e.g., autobuilding documents)
- Test Automation
  - Treat test automation code/framework like a real product
  - Treat test cases like code
  - Start small and advertise results
- MBSE Concepts
  - Reduce unnecessary details to keep model flexible/robust
  - Automate as much as possible (e.g., autobuilding model elements from a test case)
  - Document and enforce a standardized approach

Questions/Discussion

References

1. Lyu, M. R. (2007). Software reliability engineering: A roadmap. *2007 Future of Software Engineering*, 153-170. https://doi.org/10.1109/FOSE.2007.24
2. Hagen, C., Kearney, A. T., Hurt, S., & Williams, A. (2015). Metrics That Matter in Software Integration Testing Labs. *CrossTalk*, 24.
3. West, T. D., & Blackburn, M. (2017). Is digital thread/digital twin affordable? A systemic assessment of the cost of DoD's latest Manhattan Project. *Procedia Computer Science*, *114*, 47-56. https://doi.org/10.1016/j.procs.2017.09.003
4. Jorgensen, P. C. (2018). *Software testing: a craftsman's approach*. CRC Press.
5. Bucur, S., Ureche, V., Zamfir, C., & Candea, G. (2011). Parallel symbolic execution for automated real-world software testing. *Proceedings of the sixth conference on computer systems,* 183-198. https://doi.org/10.1145/1966445.1966463
6. Siegemund, K., Thomas, E. J., Zhao, Y., Pan, J., & Assmann, U. (2011). Towards ontology-driven requirements engineering. *Workshop semantic web enabled software engineering at 10th International Semantic Web Conference (ISWC)*, *Bonn, Germany*.
7. Meister, D. (2014). *Human factors testing and evaluation*. Elsevier.
8. Black, R. (2014). *Advanced software testing-vol. 2: Guide to the Istqb Advanced Certification as an Advanced Test Manager*. Rocky Nook.
9. Boehm, B. (2011). Some future software engineering opportunities and challenges. *The future of software engineering,* 1-32. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15187-3_1

References

10. Garousi, V., & Zhi, J. (2013). A survey of software testing practices in Canada. *Journal of Systems and Software, 86(5),* 1354-1376. https://doi.org/10.1016/j.jss.2012.12.051
11. Ekelund, E. D., & Engström, E. (2015). Efficient regression testing based on test history: An industrial evaluation. *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME),* 449-457. https://doi.org/10.1109/ICSM.2015.7332496
12. Olsson, H. H., Bosch, J., & Alahyari, H. (2013). Customer-specific teams for agile evolution of large-scale embedded systems. *2013 39th Euromicro Conference on Software Engineering and Advanced Applications,* 82-89. https://doi.org/10.1109/SEAA.2013.43
13. Hametner, R., Winkler, D., & Zoitl, A. (2012). Agile testing concepts based on keyword-driven testing for industrial automation systems. *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society,* 3727-3732. https://doi.org/10.1109/IECON.2012.6389298
14. Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability, 22(2)*, 67-120. https://doi.org/10.1002/stvr.430
15. Sutherland, J., & Schwaber, K. (2007). The scrum papers. *Nuts, bolts and origins of an Agile process.*
16. Hart, L. E. (2015). *Introduction to model-based system engineering (MBSE) and SysML* [Chapter meeting presentation]. Delaware Valley INCOSE Chapter Meeting, Mount Laurel, New Jersey. https://www.incose.org/docs/default-source/delaware-valley/mbse-overview-incose-30-july-2015.pdf
17. Vidal, E. J., & Villota, E. R. (2018). Sysml as a tool for requirements traceability in mechatronic design. *Proceedings of the 2018 4th International Conference on Mechatronics and Robotics Engineering,* 146-152. https://doi.org/10.1145/3191477.3191494
18. Mazeika, D., Morkevicius, A., & Aleksandraviciene, A. (2016). MBSE driven approach for defining problem domain. *2016 11th System of Systems Engineering Conference,* 1-6. https://doi.org/10.1109/SYSOSE.2016.7542911