The Fusion of Model-Based Systems Engineering (MBSE) and Model-Based Testing (MBT) Fortifies the Digital Thread.

UML Testing Profile (UTP) is the Epoxy Between Systems Engineering and Integration & Test

Presenter:
Tyler Jenkins (tyler.r.Jenkins@lmco.com)
Co-authors:
John Sweeney (john.m.sweeney@lmco.com)
Annemarie Kibbe (annemarie.l.kibbe@lmco.com)
David Harrison (david.r.harrison@lmco.com)
John West (john.west@lmco.com)

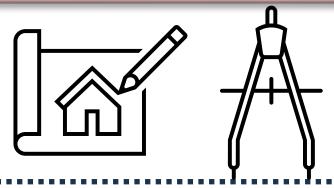
MOSA for Test & Eval Won't Work Without MBT & UTP

Historic State	Current & Future State (Leveraging MBT & UTP)
• Document based test, integration and evaluation	• Digital model assets that seamlessly connect system and test engineering disciplines
• Testing teams have different best practices to documentation	• Standardized approach to test architecture modeling
• Various teams having different test terminologies	• Common test ontology
• Multiple, disparate sources of truth	• Single source of truth which allows for reusable model libraries, test benches, interoperability and compatibility through data consistency
• Lack of standardized data exchange driving up costs	Experiencing an initial 5% to 10% cost reduction in integration and test activities • There is an up-front overhead of adoption with learning curve and training • Expected increase to 25% cost reduction as we reach economies of scale and trained workforce

BENEFITS AND VISION OF MODEL BASED TESTING WITH UTP

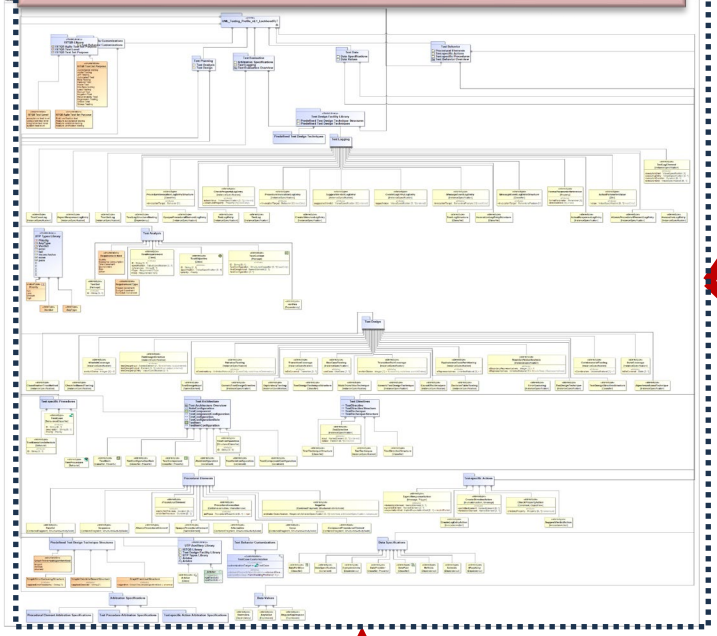
Transforming the Test Engineering Workflow

System Design and Architecture

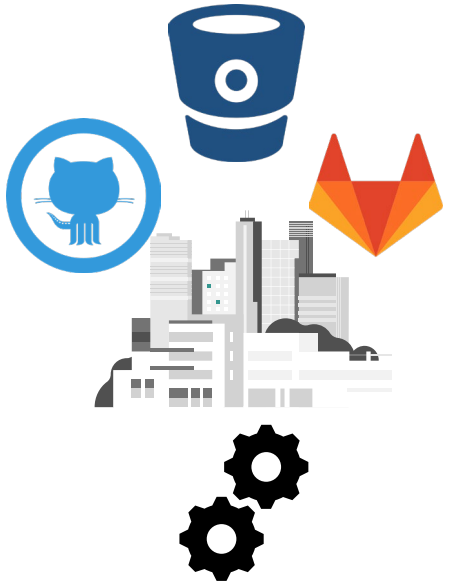


Requirements, CONOPs, Functional/Logical & Physical Architectures

Common Test Schema/Ontology



Plugin Suites/Translators



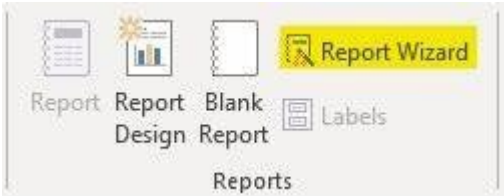
Test Scripts/Code

TAF Execution Environments

ATML

Test Configurations Test Execution Schedules

Report Generation



I&T CDRLs

Test Logs/Results

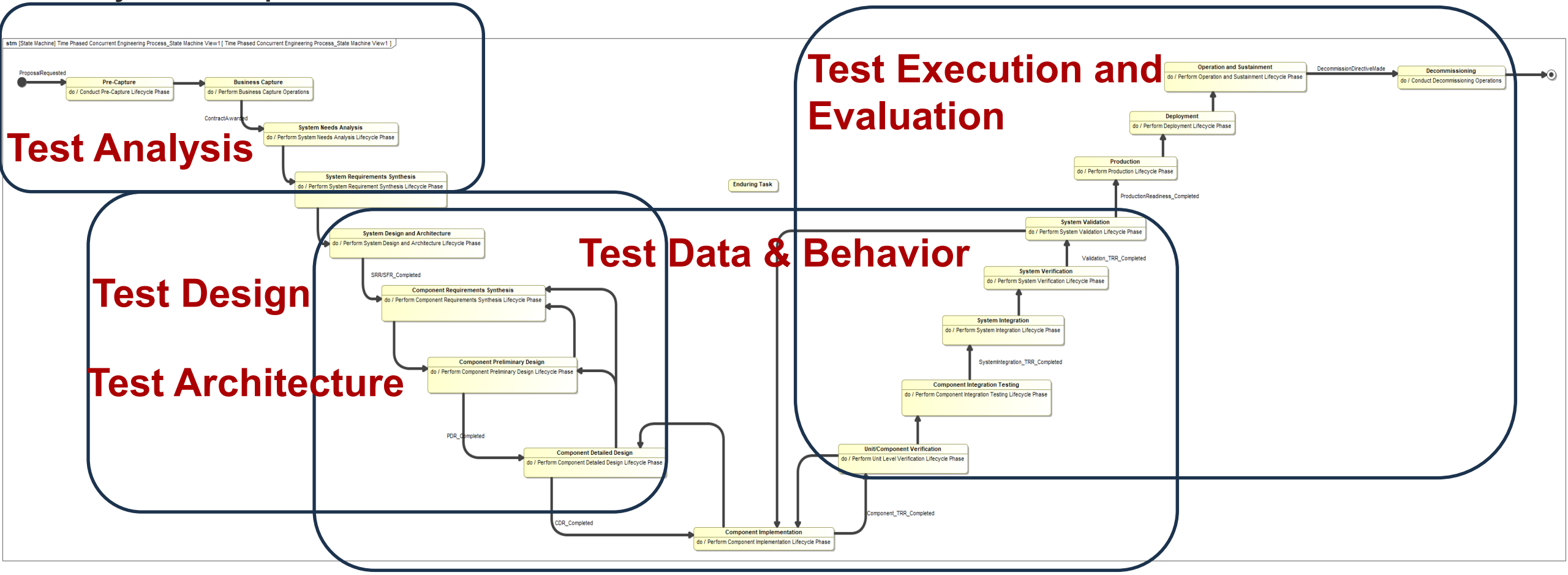
Test Results and Metrics

Defect Reports and Change Request/Orders

UML TESTING PROFILE V2.1 OVERVIEW

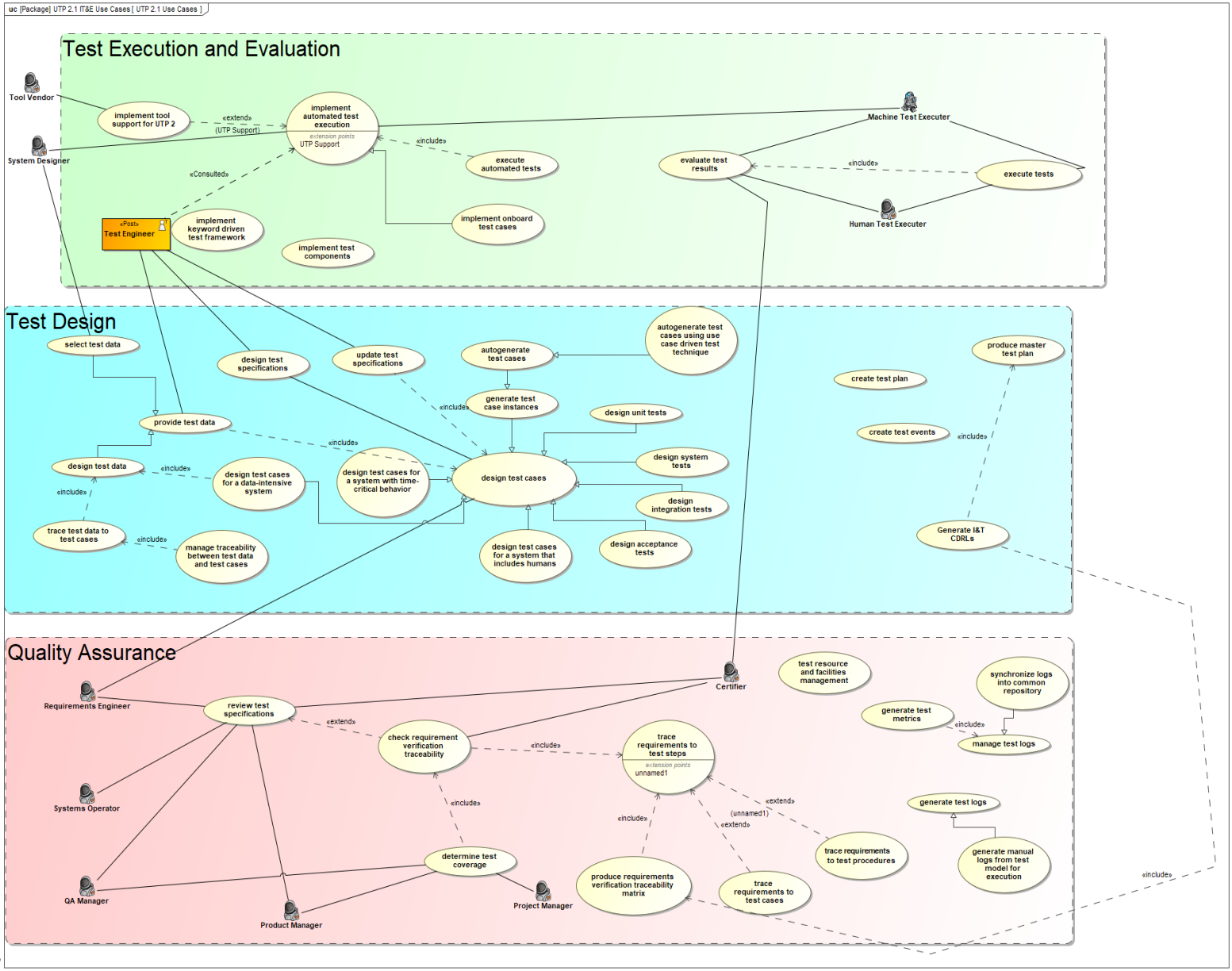
UTP v2.1 Overview- Use Cases

- UTP provides a foundation that addresses engineering processes within each phase of the system/product lifecycle (Missions Engineering through Operation and Sustainment).
- The old process takes months, the new process must take weeks/days (possibly hours) for DoD speed of relevance! The paperwork and data analysis is improved; amount of testing is decreased where possible using Scientific Test Analysis Technique.

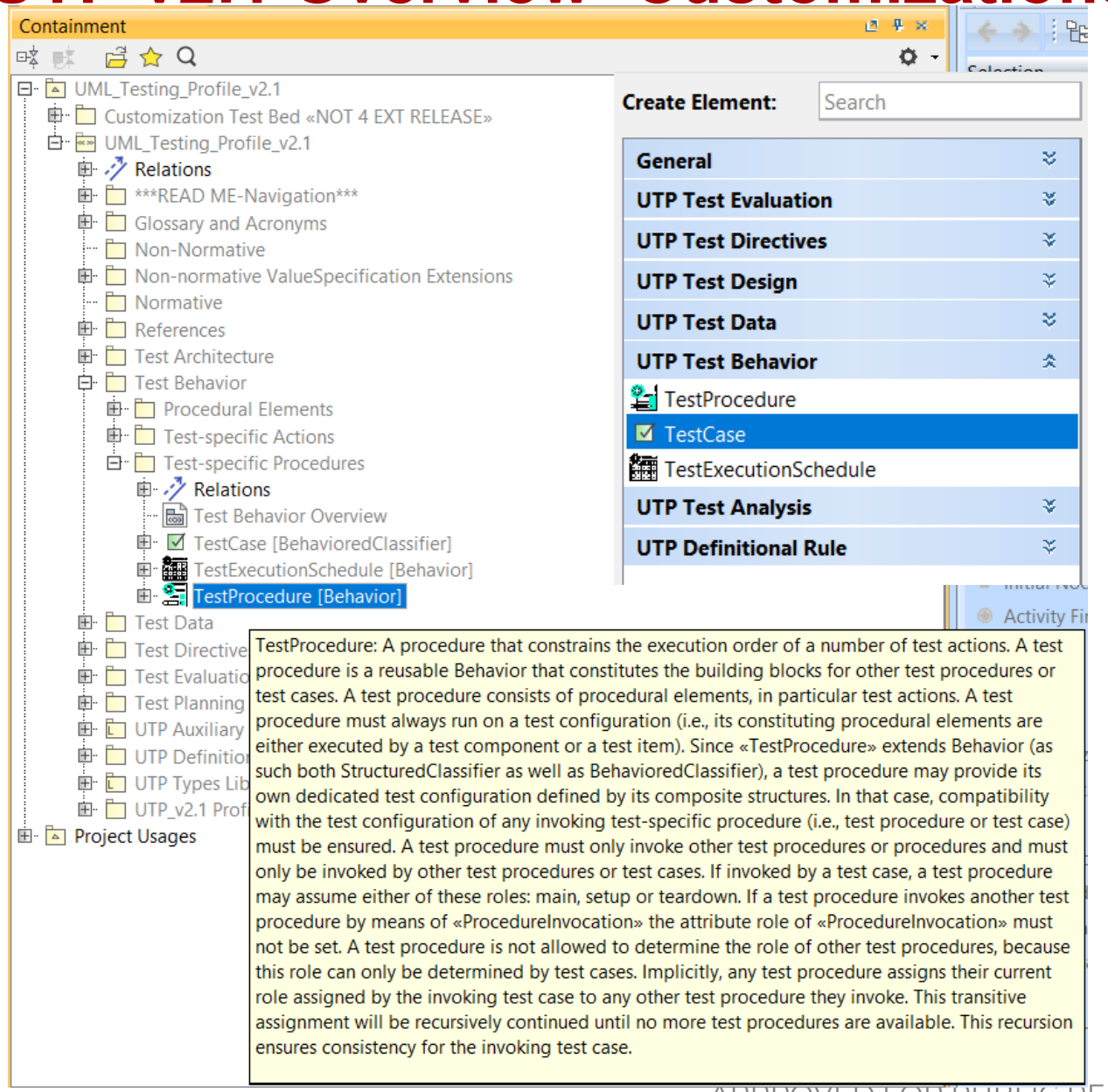


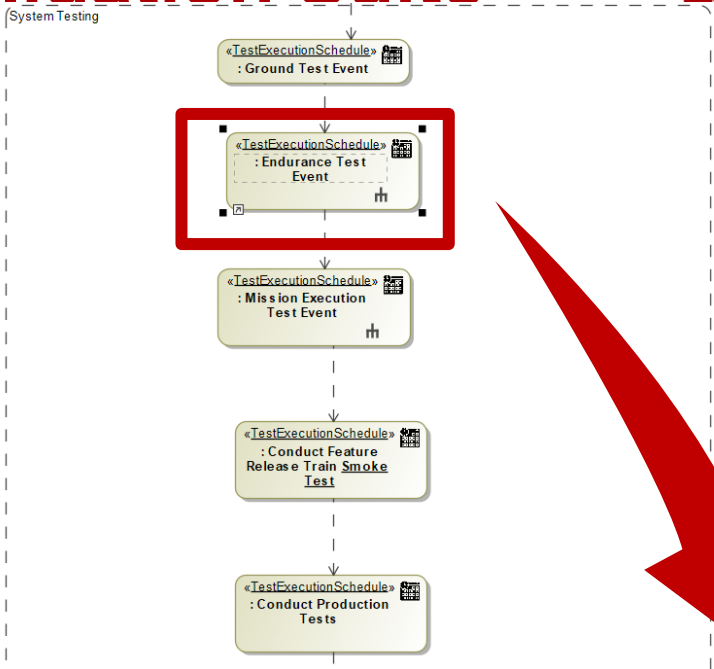
UTP v2.1 Overview- Use Cases

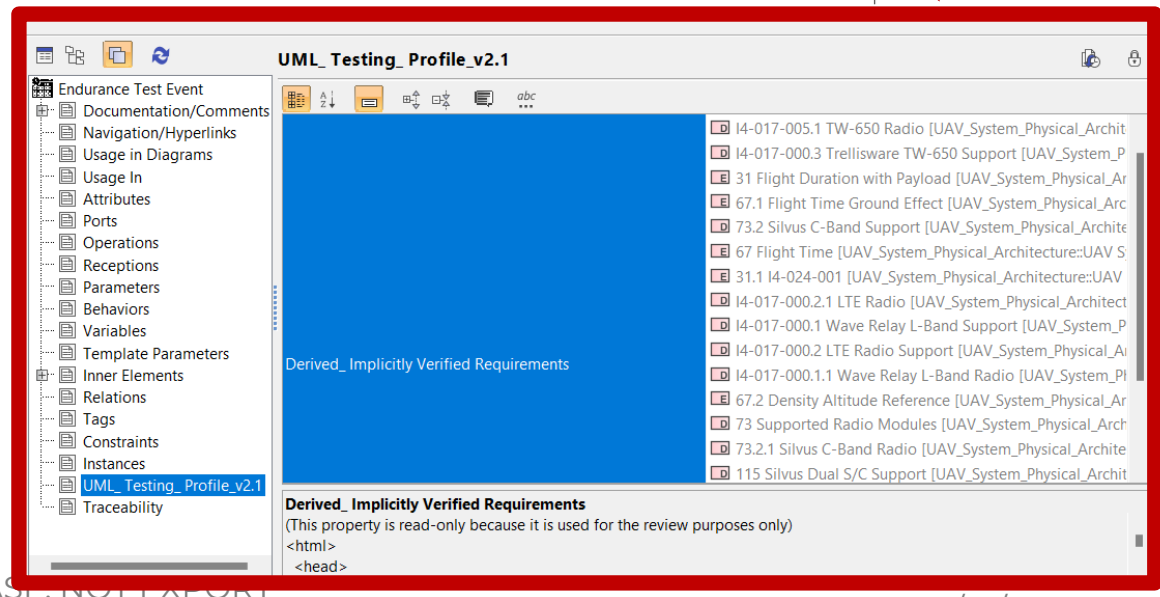
- Architecture centric CDRL generation
  - Master Test Plan, Master Test Flow, Test Reports, Test Procedures, RVTM
- Define acceptance criteria
- Define test facility/range and test configurations needed to test the system
- Manage test data specifications and instances
- Build system to component level test flows
- Define and track test metrics



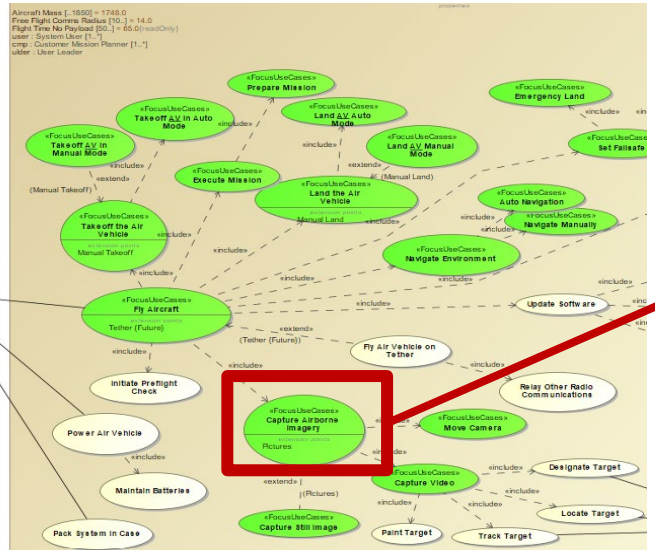
UTP v2.1 Overview- Customizations & Validation Suite





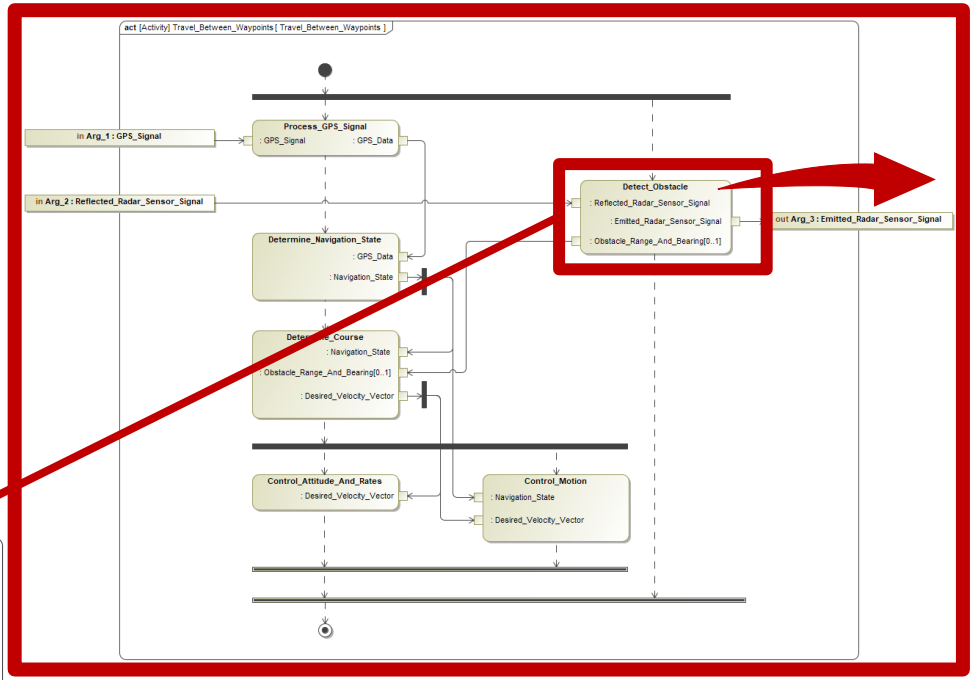


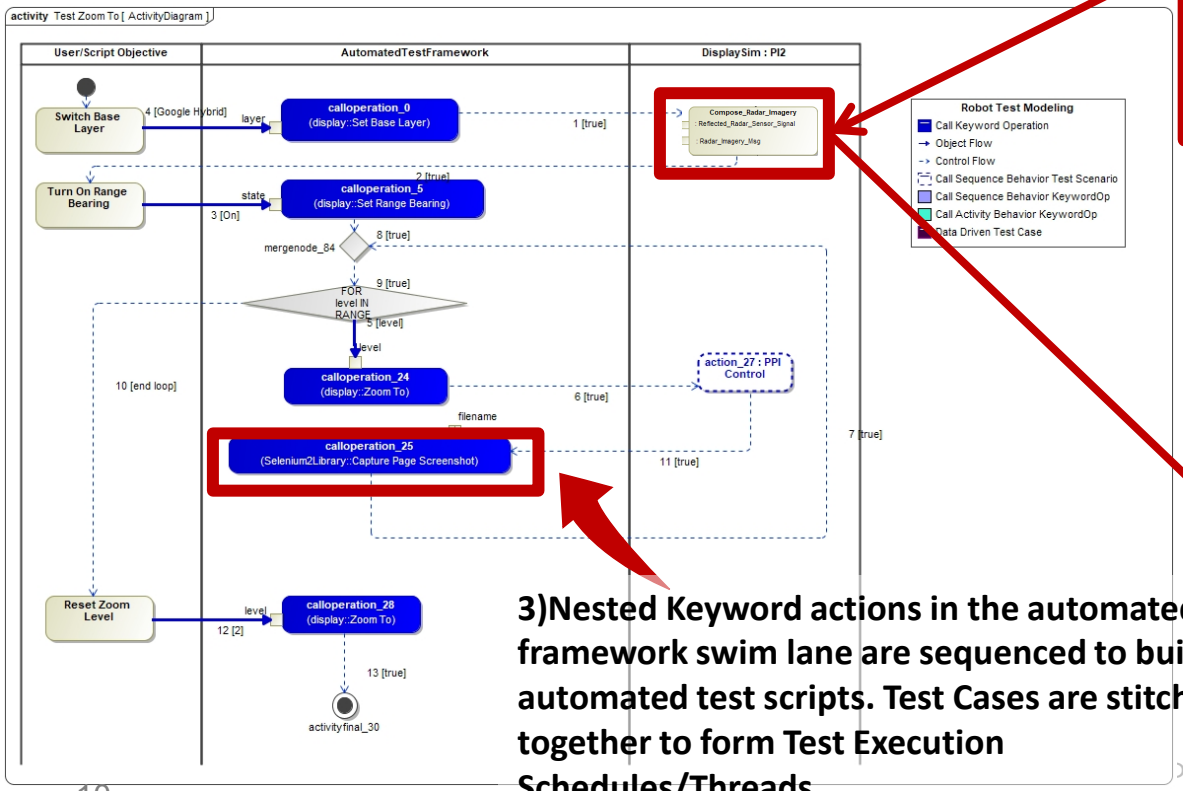
FUSION TO TEST AUTOMATION



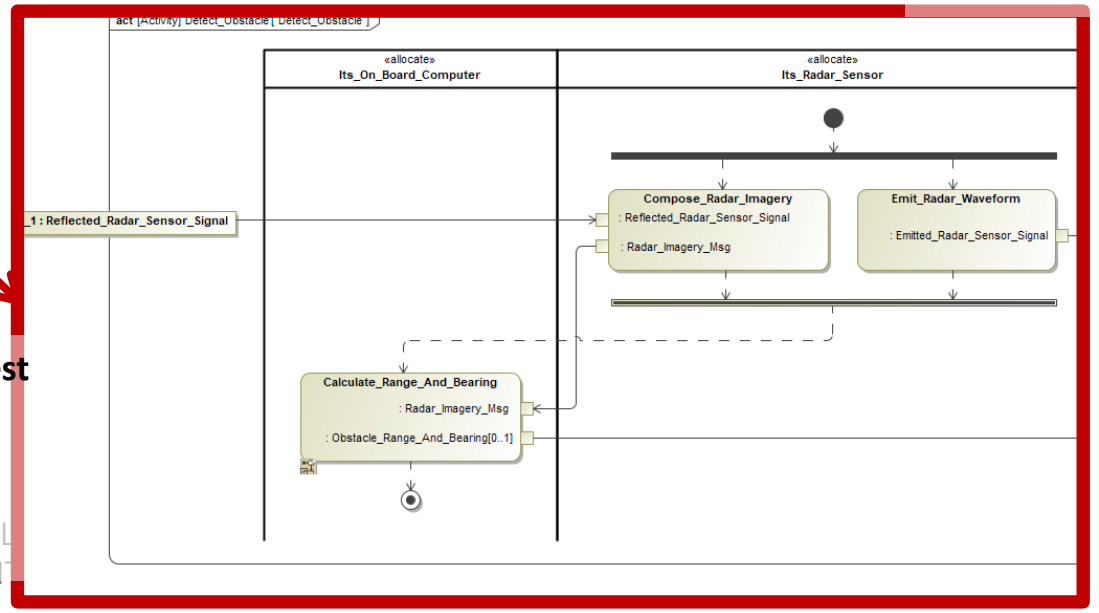
1)System Use Cases are refined by activities the system must perform to effectuate the use case.

2) System functions are used in test case diagrams to create traceability to system capabilities being verified.



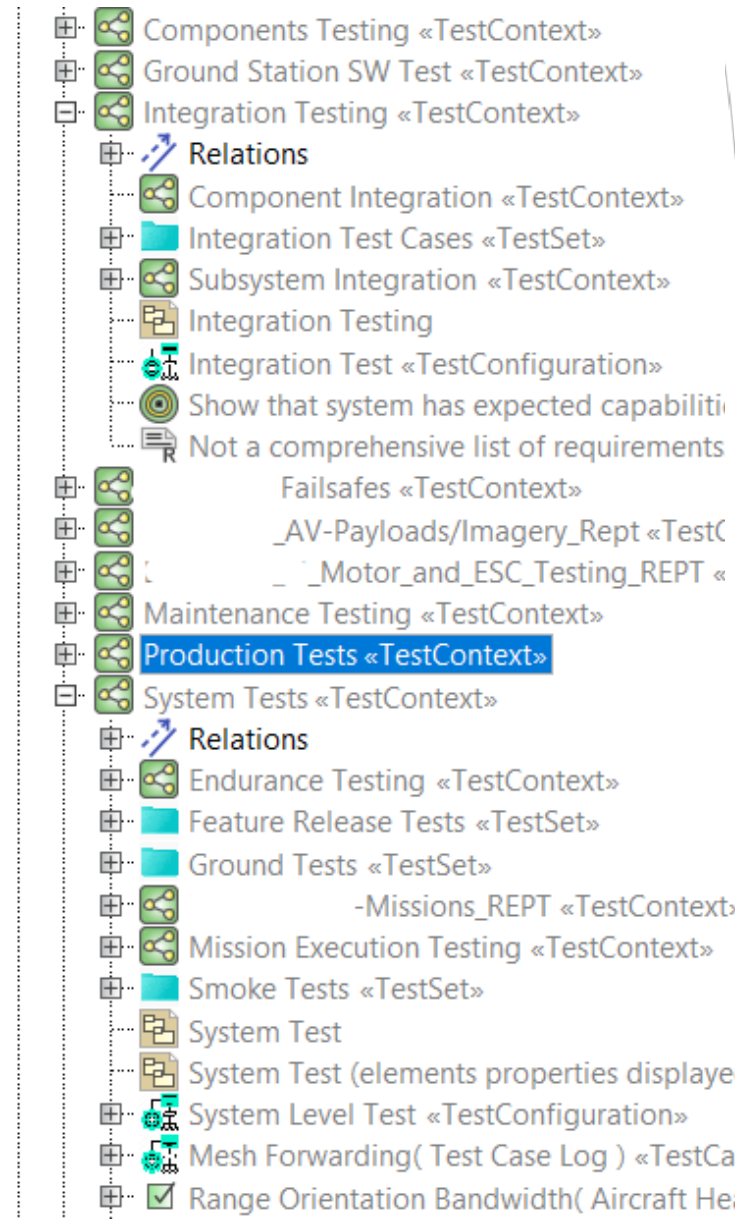


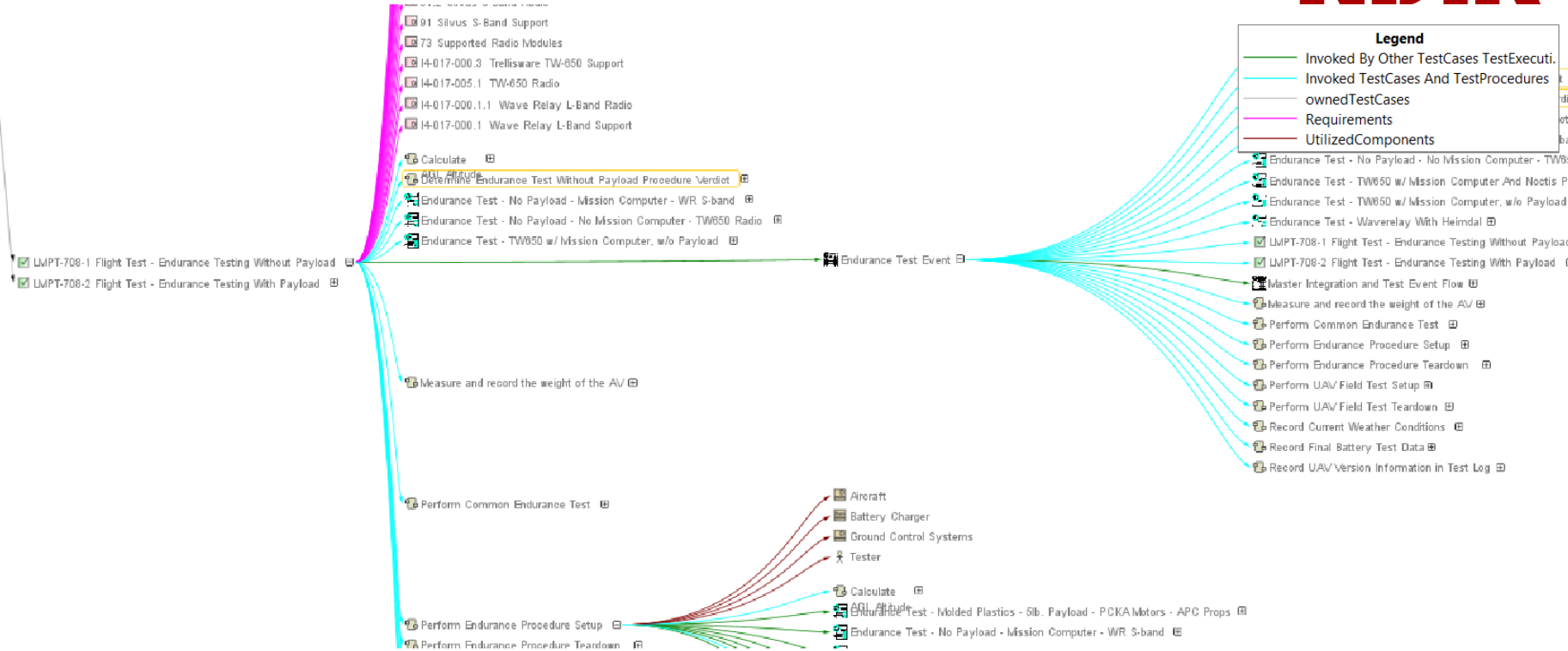
3)Nested Keyword actions in the automated test framework swim lane are sequenced to build automated test scripts. Test Cases are stitched together to form Test Execution Schedules/Threads.



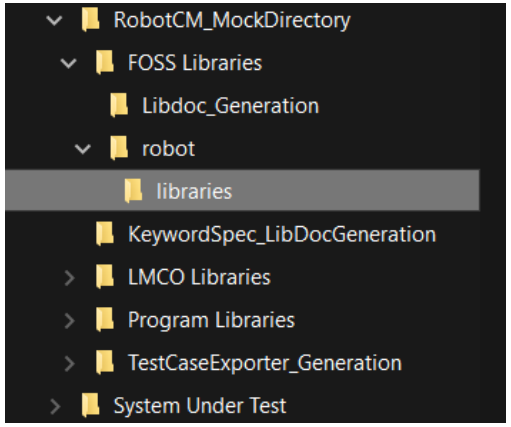
The Test Item/System Under Test (SUT) swim lane contains system functionality that the test case is verifying. SUT functions are the same functions created in the system design phase and used in the Use Case Threads. This method tracks test coverage to the system components (SUT swim lane) and/or the system component interfaces

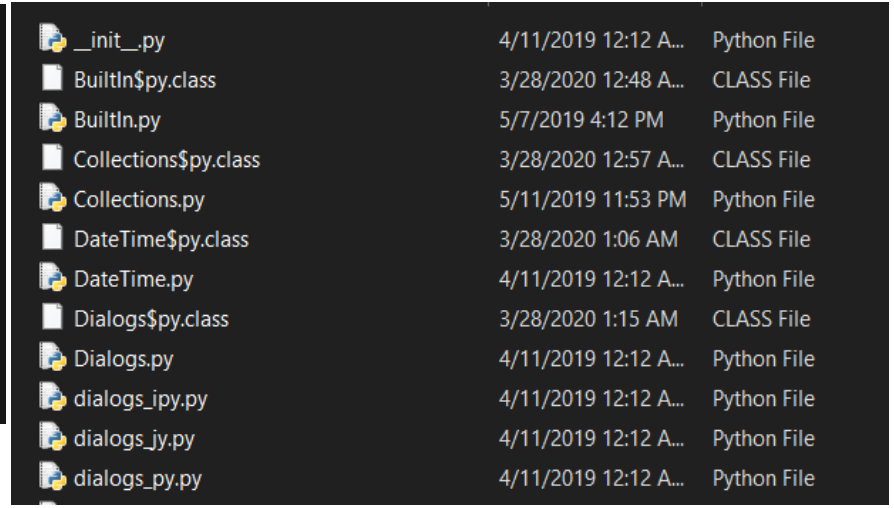
Reusable Test Architectures & Libraries





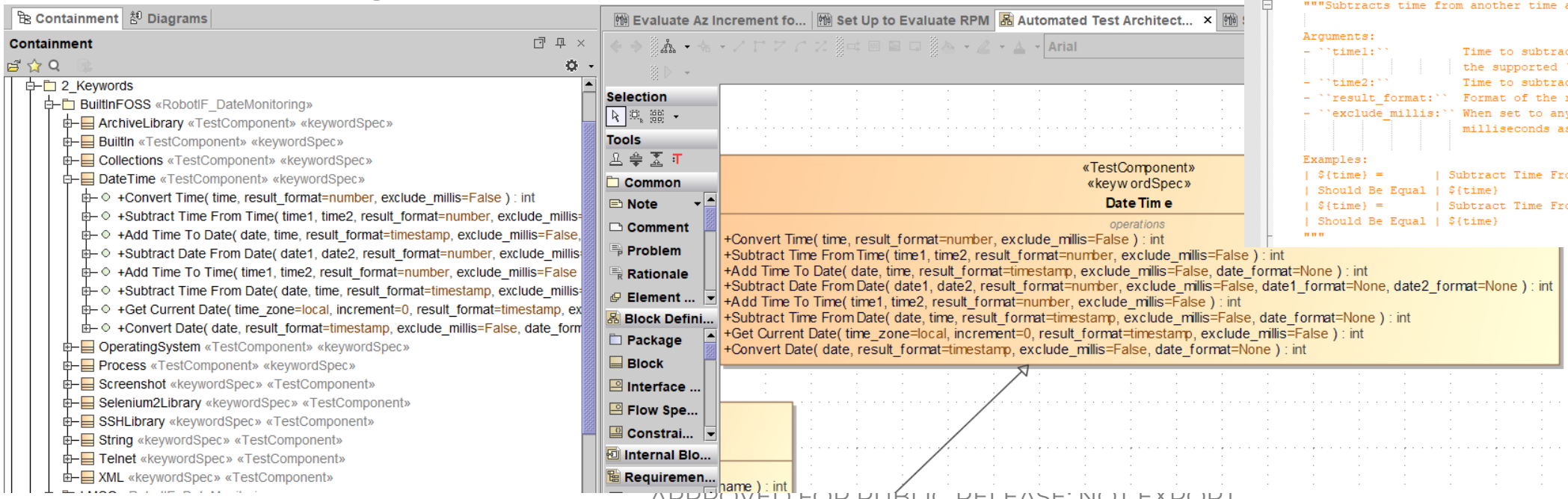
- **Left side image: Reusable test libraries from component and integration test levels can be combined feature release tests, user acceptance testing, usability tests, smoke test and regression tests.**
- **Above image: Impact Analysis Traceability view captures system elements (i.e., requirements, use cases, interfaces, components) impacted by test cases. Less time is spent making updates to tests, more time spent verifying and trying to break the system.**



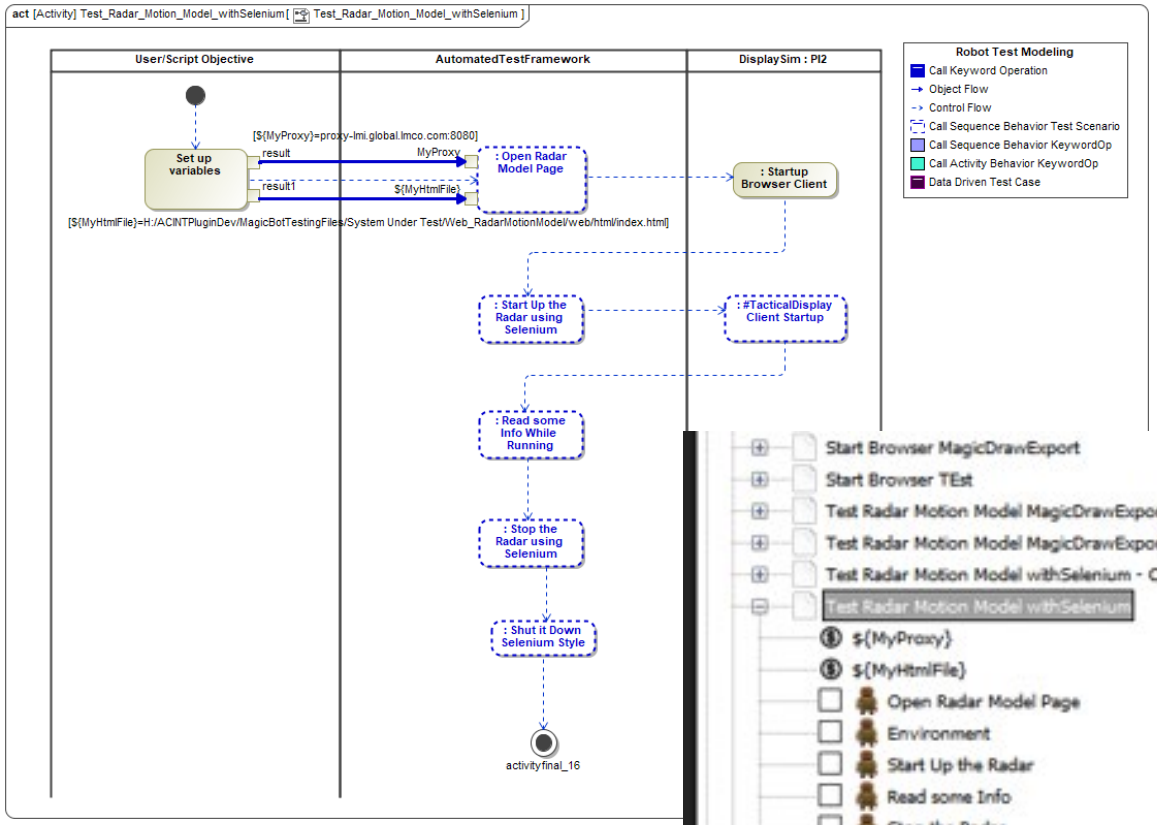


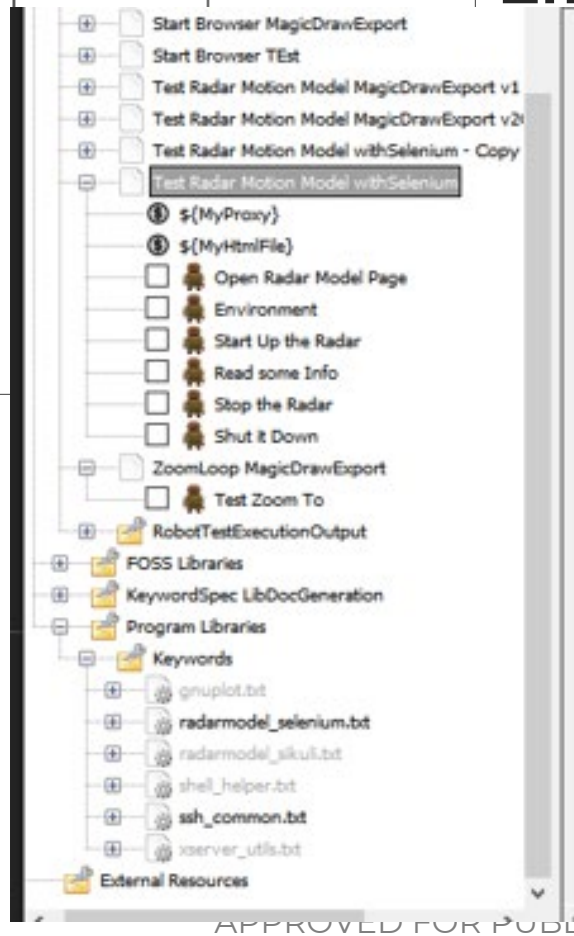
```
def get_current_date(time_zone='local', increment=0,
                     result_format='timestamp', exclude_millis=False):
    """Returns current local or UTC time with an optional increment.
    if time_zone.upper() == 'LOCAL':
        dt = datetime.now()
    elif time_zone.upper() == 'UTC':
        dt = datetime.utcnow()
    else:
        raise ValueError("Unsupported timezone '%s'." % time_zone)
    date = Date(dt) + Time(increment)
    return date.convert(result_format, millis=is_falsy(exclude_millis))

def convert_date(date, result_format='timestamp', exclude_millis=False,

def convert_time(time, result_format='number', exclude_millis=False):

def subtract_date_from_date(date1, date2, result_format='number',

def add_time_to_date(date, time, result_format='timestamp',

def subtract_time_from_date(date, time, result_format='timestamp',

def add_time_to_time(time1, time2, result_format='number',

def subtract_time_from_time(time1, time2, result_format='number',
                            exclude_millis=False):
    """Subtracts time from another time and returns the resulting time.

    Arguments:
    - ``time1:``          Time to subtract another time from in one of
                          the supported `time formats`.
    - ``time2:``          Time to subtract in one of the supported `time formats`.
    - ``result_format:``  Format of the returned time.
    - ``exclude_millis:`` When set to any true value, rounds and drops
                          milliseconds as explained in `millisecond handling`.

    Examples:
    | ${time} =       | Subtract Time From Time | 00:02:30 | 100      |
    | Should Be Equal | ${time}                 | ${50}    |
    | ${time} =       | Subtract Time From Time | ${time}  | 1 minute | compact |
    | Should Be Equal | ${time}                 | - 10s    |
    """
```

UTP helps create a standard way to synchronize test component implementation (e.g., automated test keywords) from multiple vendors. This bridges code base and architecture.



Exported test cases are translated into executable scripts for a targeted Test Automation Framework Execution Environment's syntax.






```
1  *** Settings ***
2  Documentation      Launch the Browser and Log In
3  Library            OperatingSystem
4  Library            Selenium2Library
5  Resource           ../Program Libraries/keywords/radarmodel_selenium.txt    #Resource
6
7  *** Variables ***
8  ${MyProxy}
9  ${MyHtmlFile}      H:/                    TestingFiles/System Under Test/Web_RadarMoti
10
11 *** Test Cases ***
12 Open Radar Model Page
13     # ${rc}    ${HowManyBrowsersAlready}=    Run And Return Rc And Output    xvininfo -r
14     # ${FullFilePath}=    Run    ls `pwd`/${MyHtmlFile}
15     ${MainBrosver}=    Open Browser    file://${MyHtmlFile}    browser=chrome    optio
16     Capture Page Screenshot
17     Title Should Be    Radar Motion Model
18     Log Title
19
20 Environment
21     Log Variables
22
23 Start Up the Radar
24     radarmodel_selenium.Start Radar
25     Sleep    6
26
27 Read some Info
28     Read Infobox
29     Read Azimuth
30
31 Stop the Radar
32     radarmodel_selenium.Stop Radar
33     Sleep    2
34
35 Shut it Down
36     Close All Browsers
37
```

Next Steps

- **No MBSE vendor currently offers UTP 2.1 implementation**
  - Lockheed Martin is releasing to industry the Dassault Cameo 2021x, 2022x and IBM Rhapsody UTP v2.1. Expecting to host on OMG's website. Target release is beginning of 2025.
- **Lockheed Martin plans to actively participate in the future evolution and releases of the UTP Specification**
  - Investigating the impacts SysML 2.0 will have on Cameo and Rhapsody UTP implementations.
- **Industry adoption**
  - Formalization in the DoD and DAU (Joint Capabilities Integration and Development System, **Model Based** Defense Acquisition System)
  - Critical that industry participates in the OMG UTP task force
  - Review the non-normative UTP implementation developed by Lockheed and released by the task force
  - Compliance from test execution tool and test management tool vendors

QUESTIONS?