Raytheon

Steady Grip and Agile Footing

A Balanced Foundation for Automated System Testing

Integrated Defense System (IDS)

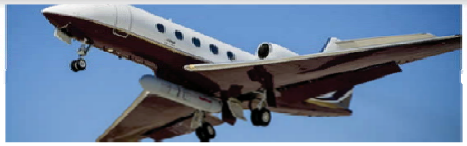
Peter Fontana
SVTAD Technical Staff
March 2, 2016

NON-EXPORT-CONTROLLED TECHNICAL INFORMATION: Artifacts determined to contain only Business Data and / or Technical Information that is publicly available in accordance with Raytheon policy 263-RP, External Communications and Public Release of Company Information, and does not contain Export-Controlled Technical Information (including those initiated in eTPCR) may be marked with the following:
"This document does not contain technology or technical data controlled under either the U.S. International Traffic in Arms Regulations or the U.S. Export Administration Regulations."

A technology and innovation leader specializing in defense, civil government and cybersecurity markets throughout the world.

- 2014 NET SALES: $23 BILLION
- 61,000 EMPLOYEES WORLDWIDE
- HEADQUARTERS: WALTHAM, MASSACHUSETTS

C5ISR



ELECTRONIC WARFARE

MISSILE DEFENSE

PRECISION WEAPONS

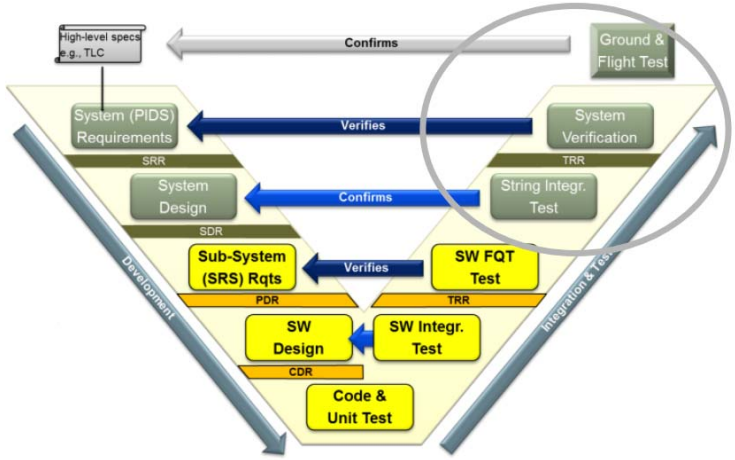
TRAINING & SERVICES

CYBER

Raytheon IDS SVTAD

Raytheon
Integrated Defense Systems

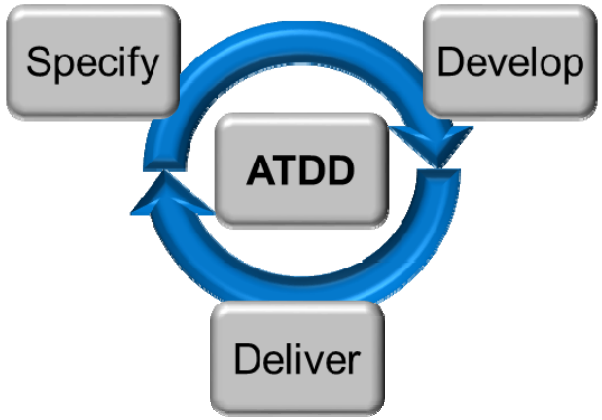
- The System Validation, Test and Analysis Directorate is responsible for the integration, verification, and validation of all Raytheon IDS products.
- Hundreds of staff from engineering and the factory to those deployed globally in the field
- We are responsible for the upper right of the system engineering “V”



TestForward

Raytheon
Integrated Defense Systems

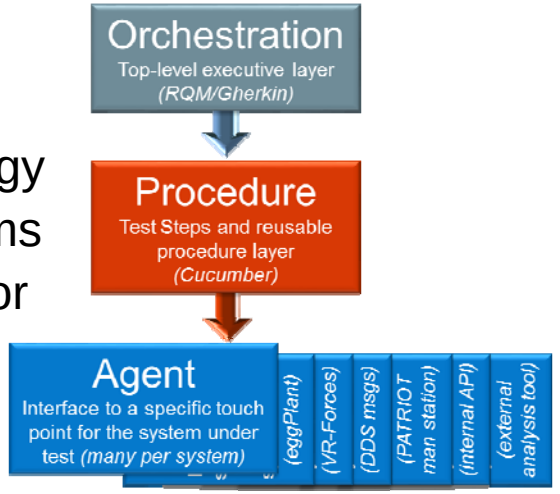
- To boost system quality and speed delivery, SVTAD is applying Acceptance Test Driven Development (ATDD) including
  - In-sprint collaboration of integration and test activities with development
  - Automated system verification testing at the mission thread level
- This initiative, *TestForward*, is driven by the confluence of Raytheon's
  - Development of Agile practices
  - Shift to mission thread-based testing
  - The push to SI&T test automation



Standard Approach Versus Adaptability

Raytheon
Integrated Defense Systems

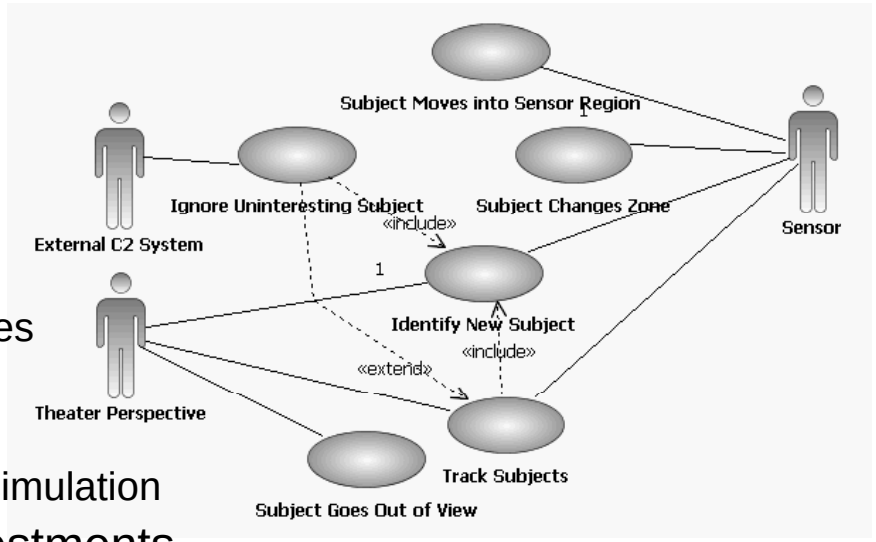
- Propagating *TestForward*/ATDD to dozens of active programs calls out for a common approach
  - Refine and adopt a single engineering method based on
    - Common management drivers
    - Sound engineering values
    - Proven automation principles
  - Build training and other learning aids once and use repeatedly
  - Deploy industry standard OTS automation technology
  - Share skills and tools configurations across programs
  - Build a basis of estimate and establish a template for project planning and management



Standard Approach Versus Adaptability

Raytheon
Integrated Defense Systems

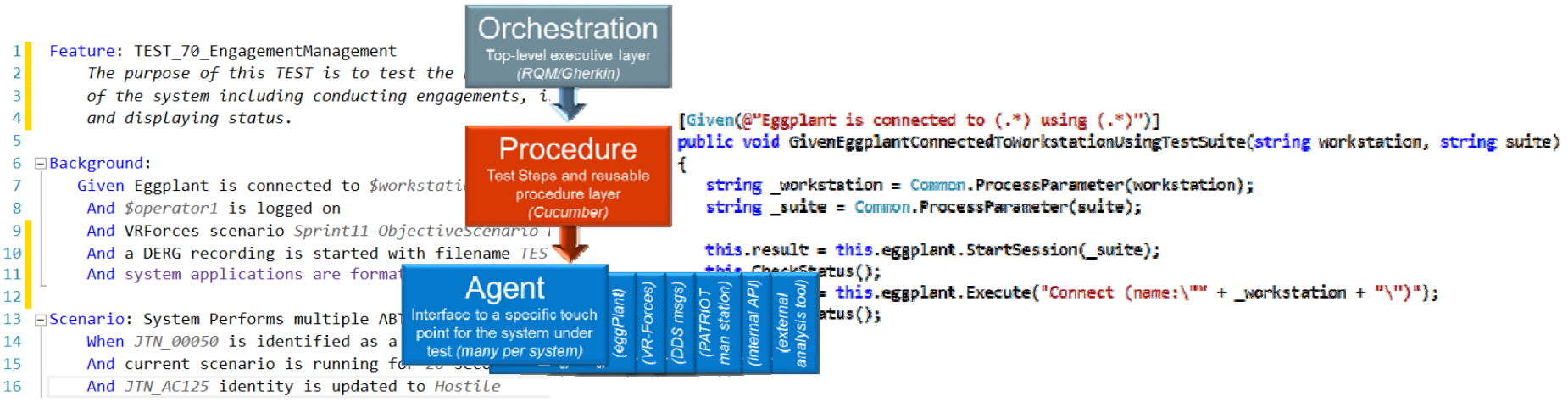
- BUT different programs can have very different needs
  - Different types of systems requiring different test techniques
    - GUI-based screen verification
    - High volume complex data-based analysis
    - Protocol-based behavioral sequence tracing
  - System test interfaces vary
    - User-level mouse and keyboard input
    - External messaging
    - Program-specific internal component interfaces
    - Data capture and marshaling
    - Information analysis
    - “Real world” target and other physical entity simulation
  - Legacy programs can have existing investments
    - Unique test tools
    - Large bodies of test scripts and data



Standard Approach Versus Adaptability

Raytheon
Integrated Defense Systems

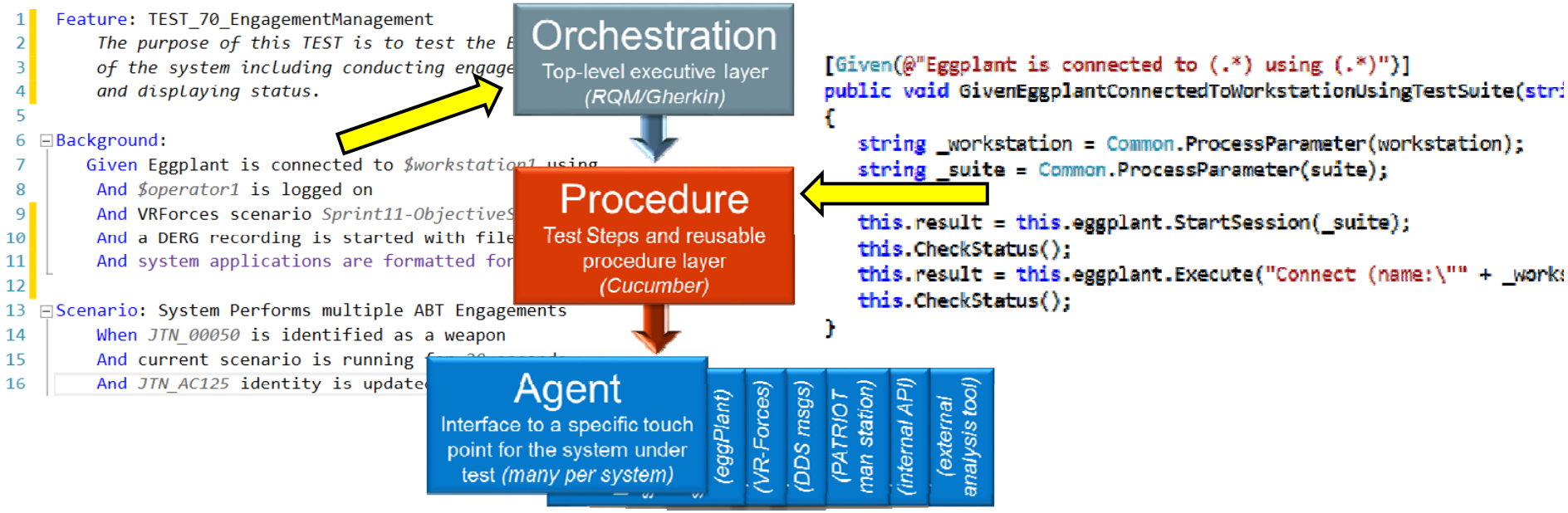
- There is a compelling need to both standardize and adapt
  - Deploy a standard ATDD method that can integrate program-specific interfaces and test techniques
  - Build on a standard automation framework based on a common scripting technology that can drive varying system interfaces through modular interfaces



Standard Approach Versus Adaptability

Raytheon
Integrated Defense Systems

- Industry Standard Test Automation Framework
  - Multi-layered, federated – plug in various interface Agents

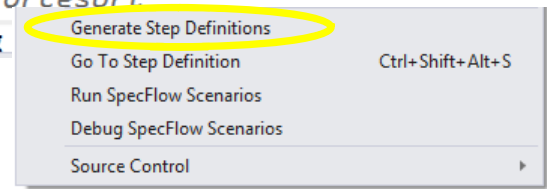


Central Test Language and Implementation

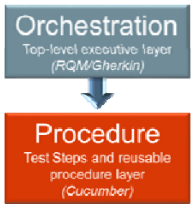
Raytheon
Integrated Defense Systems

- Map business-level Gherkin/Cucumber statements to Agent level commands:

```
Given Eggplant is connected to $workstation1 using $testSuite1
 And $operator1 is logged on
 And the VRForces agent is listening on $vrforcesUrl
 And VRForces scenario $scenario1 is running
```




```
[Given(@"Eggplant is connected to (.*) using (.*)")]
public void GivenEggplantConnectedToWorkstationUsingTestSuite(string workstation, string suite)
{
    string _workstation = Common.ProcessParameter(workstation);
    string _suite = Common.ProcessParameter(suite);

    this.result = this.eggplant.StartSession(_suite);
    this.CheckStatus();
    this.result = this.eggplant.Execute("Connect (name:\"" + _workstation + "\")");
    this.CheckStatus();
}
```



Agents – Modularity and Adaptability

Raytheon
Integrated Defense Systems

- An ***Agent*** is a software component that interfaces to one aspect of the System Under Test
  - Provides services to the test procedure to stimulate the SUT, query for state information and gather aspect-specific data
  - Embodies reusable FOSS communication services (HTTP/REST) to provide both location-independence and platform-independence

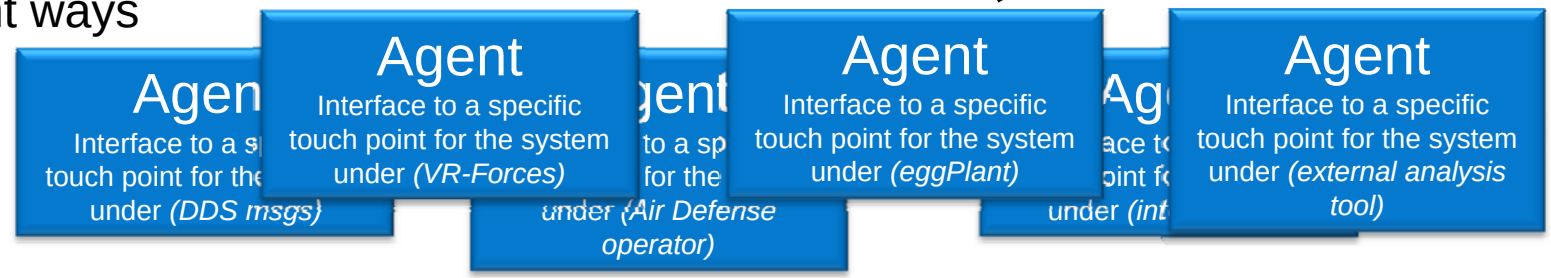
- This federated architecture - Cucumber procedures driving distributed Agents - is a key enabler to achieving
  - Flexible test and deployment topologies via Agent communications
  - Platform Independence, decoupling test procedures from Agent implementation
  - Decoupled interfaces - add/modify/reuse individual Agents independently

Agents – Modularity and Adaptability

Raytheon
Integrated Defense Systems

- Technology adaptation is through the Agents
  - Adapt different point tools, like TestPlant eggPlant or HP UFT
  - All agents conform to a common test script interface standard
    - Cucumber-based
    - Robust
    - Simple
  - SUT interface with different systems in different ways

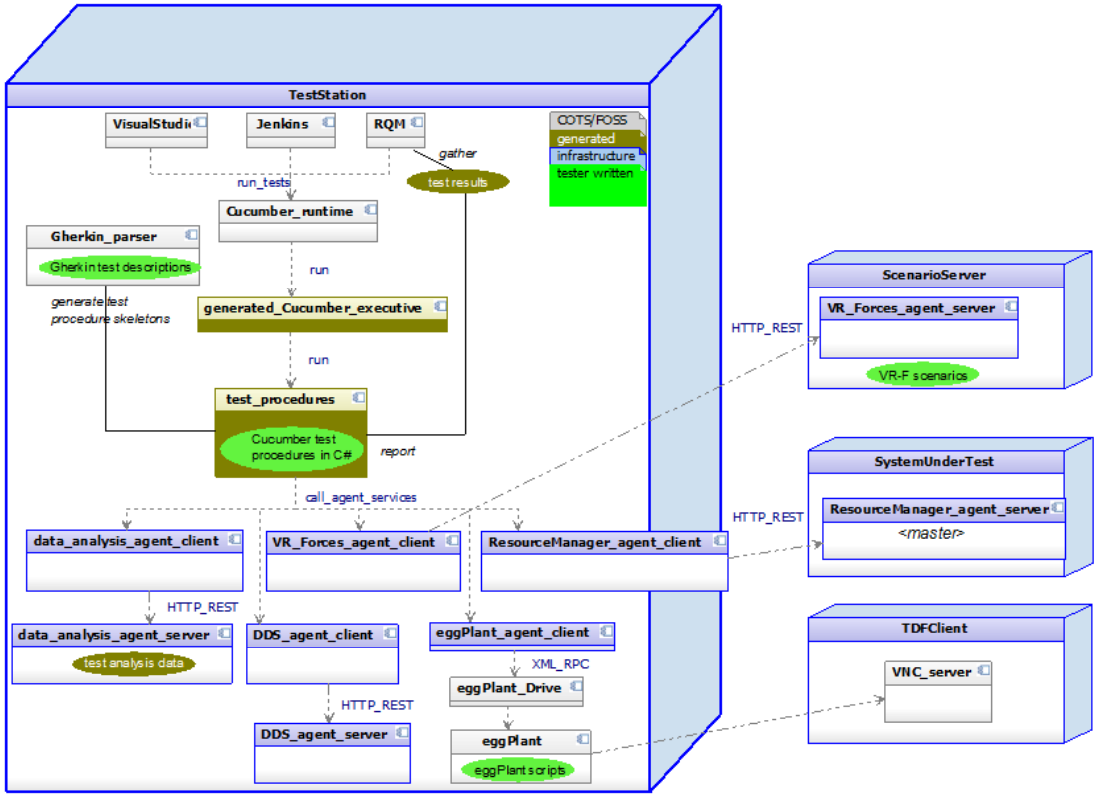
```
[Given(@"Eggplant is connected to (.*) using (.*)")]
public void GivenEggplantConnectedToWorkstationUsingTestSu
{
    string _workstation = Common.ProcessParameter(workstati
    string _suite = Common.ProcessParameter(suite);

    this.result = this.eggplant.StartSession(_suite);
    this.CheckStat();
    this.result = s.eggplant.Execute("Connect (name:\""
    this.CheckSta();
}
```



Agents – Modularity and Adaptability

Raytheon
Integrated Defense Systems

- Example Test Framework and Agent deployment:



Results

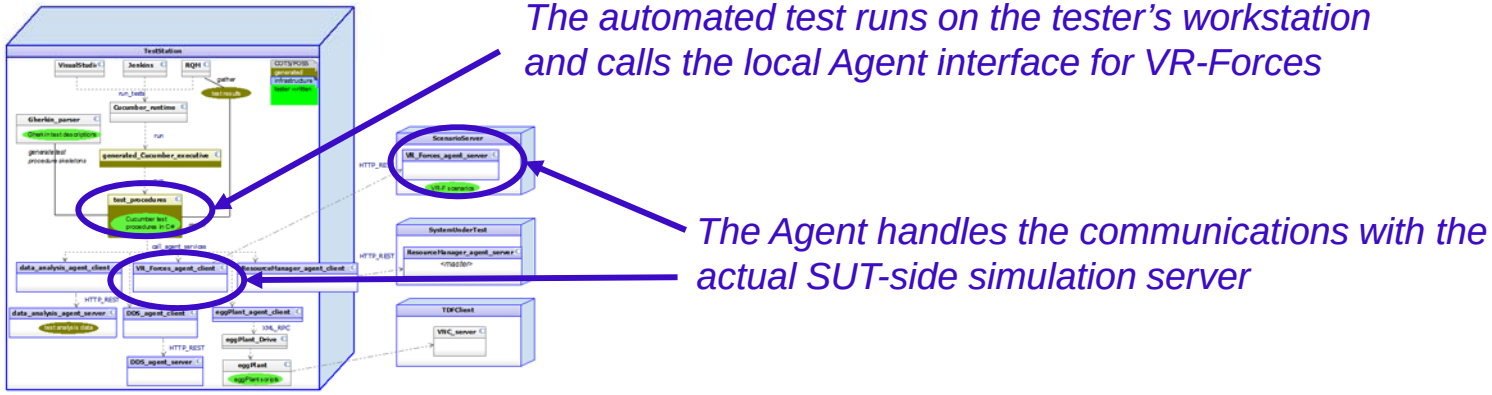
Raytheon
Integrated Defense Systems

- How well does this approach work?
  - Automation portability and reuse
  - Common system interfaces
  - Unique system interfaces
  - Legacy Automation
  - Alternative point of contact technologies

Results – Automation Portability

Raytheon
Integrated Defense Systems

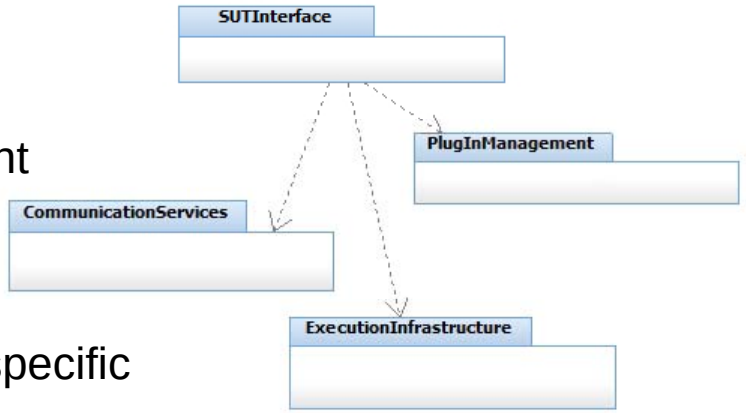
- Gherkin/Cucumber portability
  - Agent connectivity approach mitigated the need for native test automation – only the far (server) half of the Agent is integrated with SUT
  - Cucumber has integrations for over a dozen languages/environments from Java and C++ to Ruby and TCL.
    - Gherkin scripts developed in Eclipse JDT Cucumber on Linux can connect to legacy subsystems still implemented in Jovial on embedded processors



Results – Automation Portability

Raytheon
Integrated Defense Systems

- Agent portability
  - Test-side Agents (clients) work from a common architecture, platform and toolkit
    - Built and run in test programming environment (Eclipse JDT/Java, Visual Studio/C#)
    - Use many FOSS components: REST, JSON
    - Strong reuse from program to program
  - SUT-side Agents require much more program-specific adaptation
    - Some don't have FOSS HTTP/REST or JSON available
    - Some have limited or proprietary communications available
    - Some are complicated by security needs to limit or eliminate testability software from tactical deployments



Results – Automation Reuse

Raytheon
Integrated Defense Systems

- Gherkin/Cucumber Scripting
  - Some test steps exercise standard interfaces (Agents) in standard ways
    - **Given** Health and Monitoring Logging started at *Warning* level
    - **When** Built In Test for *Warm Start* initiated
  - Some steps are program specific, exercising unique interfaces
    - **When** I log in as *Air Defense* Operator at Console *4*
    - **Then** the Protected Zone Alerts are automatically displayed

```
[Given(@"Eggplant is connected to (.*) using (.*)")]
public void GivenEggplantConnectedToWorkstationUsingTestSuite(string workstation, string suite)
{
   string _workstation = Common.ProcessParameter(workstation);
   string _suite = Common.ProcessParameter(suite);

   this.result = this.eggplant.StartSession(_suite);
   this.CheckStatus();
   this.result = this.eggplant.Execute("Connect (name:\"" + _workstation + "\")");
   this.CheckStatus();
}
```

Results – Automation Reuse

Raytheon
Integrated Defense Systems

- Actual Gherkin reuse is not considered significant
  - Likely to be program specific even when using cross-program interfaces/agents – “stream of consciousness”
  - The Cucumber level of abstraction is where the programming work happens
- Cucumber Step reuse is more significant
  - Simple modularization and parameterization
    - Reduces cloning
    - Supports binding multiple Gherkin steps to same Cucumber Step Implementation
  - The Agent interface is where the complexity lies - common Agents boost Cucumber reuse

```
[Given(@"Eggplant is connected to (.*) using (.*)")]
public void GivenEggplantConnectedToWorkstationUsingTestSuite(string workstation, string suite)
{
   string _workstation = Common.ProcessParameter(workstation);
   string _suite = Common.ProcessParameter(suite);

   this.result = this.eggplant.StartSession(_suite);
   this.CheckStatus();
   this.result = this.eggplant.Execute("Connect (name:\"" + _workstation + "\")");
   this.CheckStatus();
}
```

Results – Common system interfaces

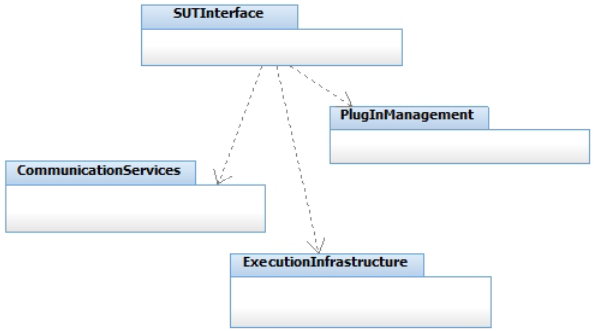
Raytheon
Integrated Defense Systems

- The Agent interface approach encapsulates each unique interface of a system to be tested
  - DDS messaging
  - GUI
  - SNMP Device Control and Status
  - Standardized system instrumentation data (track info, health, performance)
- Systems that share common interfaces and subsystems also share Agents
  - Agent client and server code is reused
  - Cucumber implementation of common Agent requests is carried over and adapted

Results – Unique system interfaces

Raytheon
Integrated Defense Systems

- The total cost of establishing test automation for a program can be substantially driven by the need to build Agent interface software
  - Many Systems have unique interface needs for SI&T
  - Agents can be specific to a program
    - Subsystem-specific API (Radar Data Reduction, Track Correlation, Network Security)
    - Data reduction Agent for unique data
    - Command/query interface unique hardware interface
  - Some programs are the first to automatically test a common interface, and have to build it



Results – Legacy Automation

Raytheon
Integrated Defense Systems

- Legacy programs typically aren't using an adaptable, modular test automation framework
  - Many have scattered and ad-hoc automation
  - Even widely automated programs often used a simplistic automation approach
    - Solve one program's needs
    - Often organically grown by "midnight hero" efforts
  - Ongoing maintenance, ECPs and Phase N+1 program awards can stretch rigid and fragile automation
- Measured, careful steps forward
  - Retrofit the Cucumber framework
  - Encapsulate effective legacy automation with Agents
  - Selective, to preserve existing capability
  - Provide a growth path forward

Results – Alternative point of contact technologies

Raytheon
Integrated Defense Systems

- Some points of contact for the SUT are serviced by COTS or other existing technologies:
  - *GUI*: TestPlant's eggPlant, or HP's Unified Functional Testing
  - *Target Generation*: MAK's VR-Forces, or program-specific
- The Agent provides a consistent interface to these alternatives
- This frees each program to choose the alternative the meets their needs best:
  - Capability
  - Cost
  - Availability
  - User preference

Summary

Raytheon
Integrated Defense Systems

- Our *TestForward* approach is explicitly tasked to both standardize and adapt:
  - Deploy a standard ATDD method that readily integrates program-specific interfaces and test techniques
  - Build on a unified automation framework and common scripting technology that drive varying system interfaces through modular Agents