

# ***U.S. Navy Automated Software Testing***

***Application of Standards to the  
Automated Test and Re-Test (ATRT) Effort***



***Object Management Group (OMG) Technical Meeting  
June 2007***

- **What is Automated Test and Re-test (ATRRT)?**
- **Current Test and Evaluation (T&E) Drivers**
- **ATRRT Approach, Scope, and Strategy**
- **Traditional T&E Process Factors**
- **ATRRT and the Software Development Life Cycle**
- **Types of Automated Test Tools**
- **STAF/STAX Example Framework**
- **Test Tool Implementation and Challenges**
- **Benefits and Challenges**
- **What is the Way Forward?**

# What is *Automated Test and Re-Test (ATRTR)?*



**ATRTR is a Naval Sea Systems Command (NAVSEA) effort whose objective is to leverage commercial automated testing “best practices” in U.S. Navy programs to**

- **Reduce overall system development testing costs**
- **Maintain or improve software product quality**
- **Shorten the software certification timeline**



# Current T&E Drivers

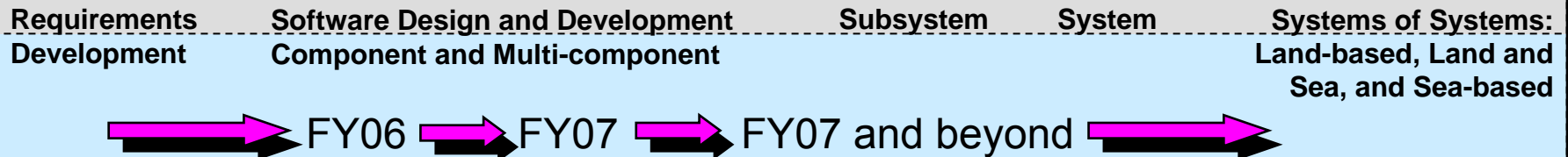
- **Mission-critical U.S. Navy software is tested and retested manually, multiple times.**
- **Manual testing is costly, and it is difficult to schedule and obtain resources (trained personnel, facilities, and equipment).**
- **Multiple deliveries of software and software systems are necessary.**
- **Technology insertions and software reuse require additional time-intensive and costly regression testing.**
- **Force-level and Net-centric capabilities have added complexity.**



# ATRT Approach and Scope



Existing U.S. Navy Development Cycle = Combination of Technical and Business Processes from U.S. Navy and Industry



**Approach:**

- Incorporate automated testing of software and systems
- Emphasize commercial tools and technologies
- Partner with existing automation efforts
- Emphasize standards and reuse

**Results:**

- Implement standards to support automated testing
- Develop automated testing architectures and frameworks
- Increase partnerships among industry and government for implementation of automated testing

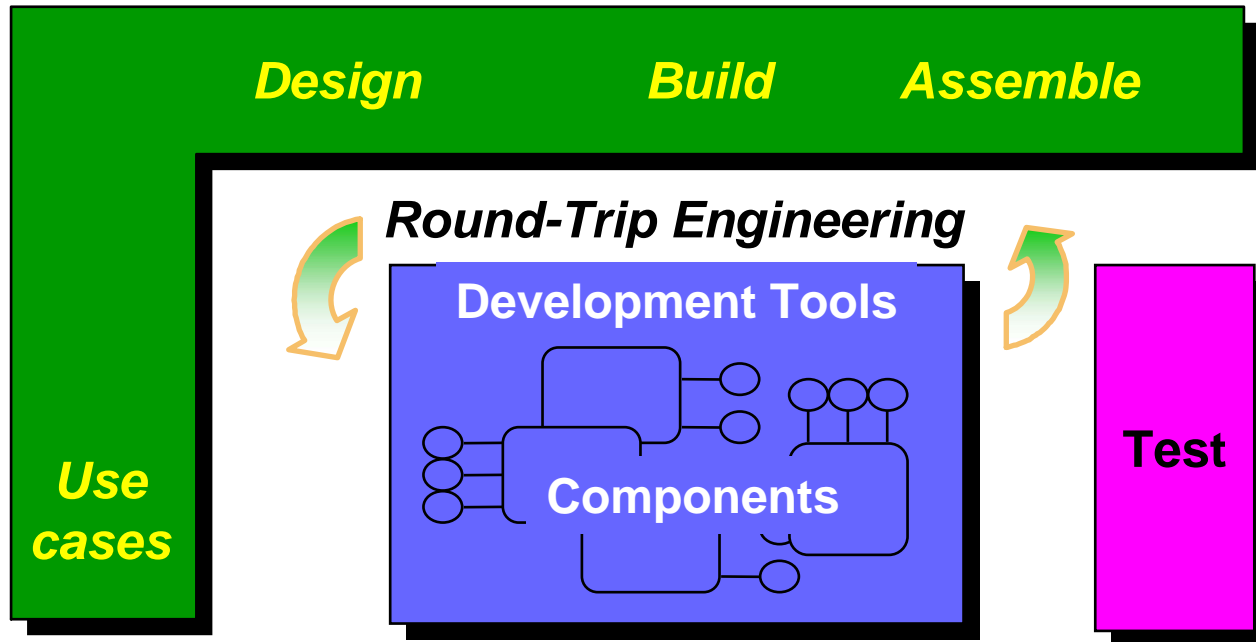


# *ATRT Definition of Automated Software Testing*

---



***Application and implementation of software technology throughout the entire Software Testing Life Cycle (STL) with the goal to improve STL efficiencies and effectiveness***

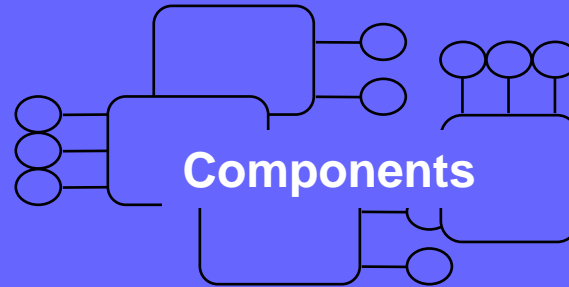


- Support for ActiveX, Java, Common Object Request Broker Architecture (CORBA), and Java 2 Platform, Enterprise Edition (J2EE)
- Support for Unified Modeling Language (UML)

Requirements Management and  
Process Automation

Visual Modeling

Development Tools

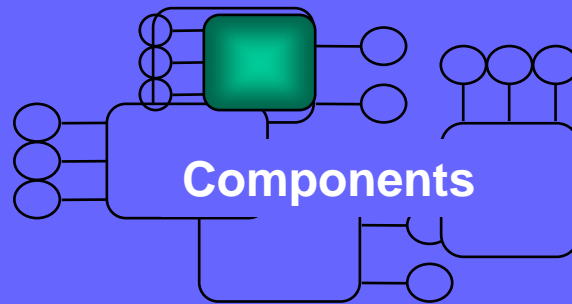


- 👉 Organizes, tracks, & controls requirements
- 👉 Requirements Traceability Matrix (RTM)

Requirements Management and  
Process Automation

Visual Modeling

Development Tools



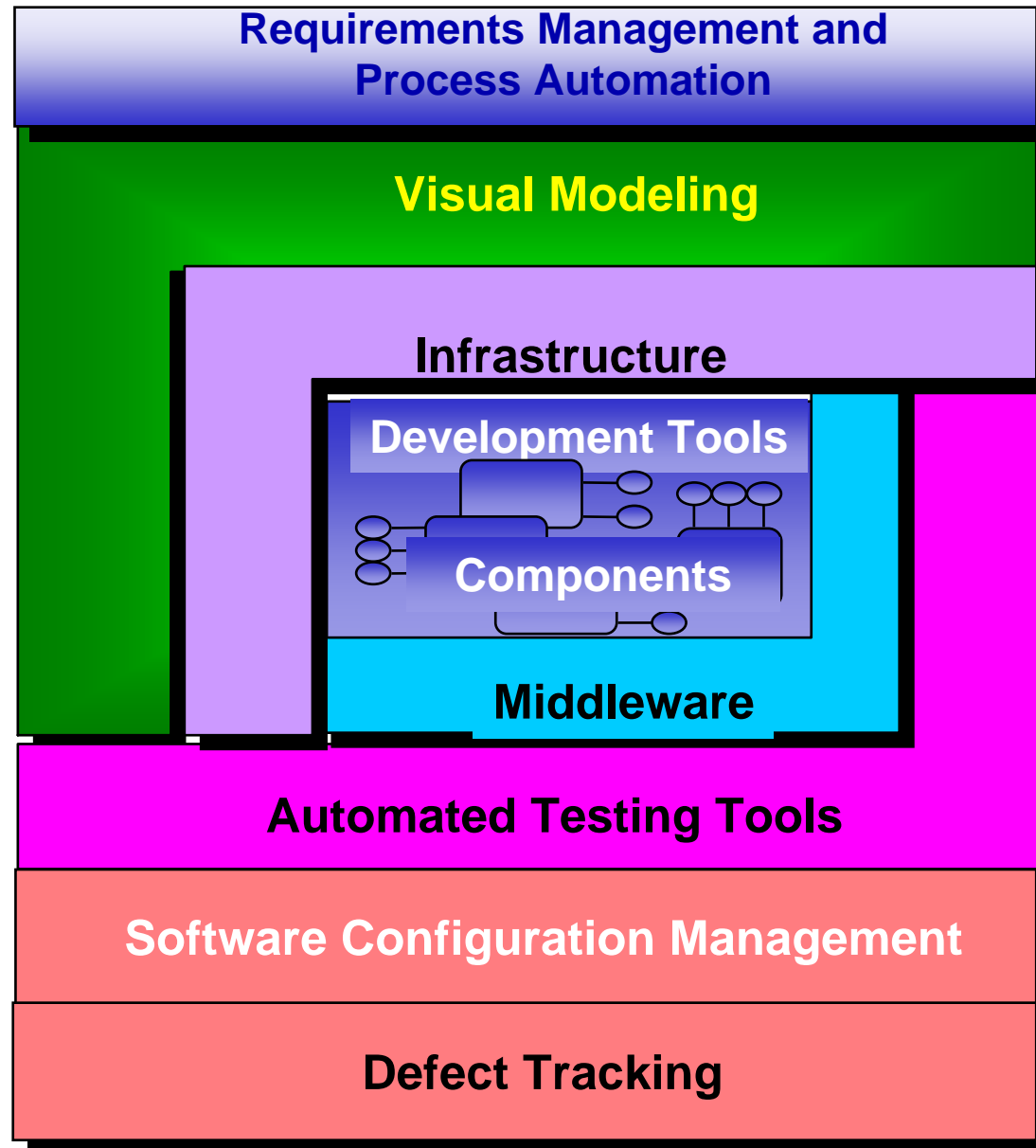
*Development  
Management  
Execution*

Automated Testing Tools

- ☞ Automates test cases using vendor-provided, open-source tools or in-house development

Additionally:

- ☞ Middleware
- ☞ Infrastructure
- ☞ Defect Tracking
- ☞ Configuration Management
- ☞ Memory Leak Detectors
- ☞ Performance Testing Tools
- ☞ Documentation Tools



Life-Cycle Phase	Type of Tool	Tool Description
<b>Business Analysis Phase</b>	Business Modeling	Records definitions of user needs and automates rapid construction of flexible, graphical, client-server applications
	Configuration Management	Baselines important data repositories
	Defect Tracking	Manages system life-cycle defects
	Technical Review Management	Facilitates communication, while automating the technical review/inspection process
	Documentation Generators	Automate document generation
<b>Requirements Definition Phase</b>	Requirements Management	Manages and organizes requirements; allows for test procedure design and test progress reporting
	Requirements Verifiers	Verify syntax, semantics, and testability
	Use Case Generators	Create use cases

Life-Cycle Phase	Type of Tool	Tool Description
<b>Analysis and Design Phase</b>	Database Design	Develops second generation enterprise client-server systems
	Structure Charts, Flowcharts, and Sequence Diagrams	Manage processes
	Test Procedure Generators	Generate test procedures from requirements, design, or data and object models
	Syntax Checkers/ Debuggers	Perform syntax checking and have debugging capability; usually available with built-in programming language compiler
<b>Programming Phase</b>	Memory Leak and Runtime Error Detection	Detects runtime errors and memory leaks
	Source Code Testing	Verifies maintainability, portability, complexity, and standards compliance
	Static and Dynamic Analyzers	Depict quality and structure of code
<b>Metrics Tools</b>	Code (Test) Coverage Analyzers or Code Instrumentors	Identify untested code and support dynamic testing
	Usability Measurements	Provide usability testing as conducted in usability labs

# Types of Tools (Cont'd)

Life-Cycle Phase	Type of Tool	Tool Description
<b>Other Testing Life-Cycle Support Tools</b>	Test Data Generators	Generate test data
	File Compare Utilities	Find discrepancies between files that should be identical in content
	Simulation	Simulates application to measure for scalability, among other tasks
	Test Management	Tests management
	Network Testing	Monitors, measures, tests, and diagnoses performance across the entire network
	GUI Testing (Capture/Playback)	Conducts automated GUI tests; capture/playback tools record user interactions with online systems so they may be replayed automatically
	Load/Performance Testing	Conducts load/performance and stress testing
	Security Testing	Performs security testing and vulnerability scanning at the application or network level; plus debuggers will allow to check for security coding errors (source code checkers)

## Vendor-provided (Capture/Playback) Tool:

- Automated test tools mimic actions of the test engineer.
- During testing, the engineer uses the keyboard and mouse to perform some type of test or action.
- Testing tool captures all keystrokes and subsequent results, which are baselined in an automated test script.
- During test playback, scripts compare latest outputs with previous baseline.
- Testing tools have built-in, reusable test functions.
- Most test tools provide for non-intrusive testing; i.e., they interact with the “application-under-test” as if the test tool was not involved.

## In-House Software Development:

- Vendor-provided tools generate hard-coded values; test scripts are not reusable, nor do they implement software development best practices right out of the box; scripts need to be modified.
- Vendor-provided tools don't necessarily provide all testing features required; code enhancements are often required to meet testing needs.
- Vendor-provided tools are not necessarily compatible with system engineering environment, and software testing scripts need to be developed in-house.

# Automation Tasks - Example

**Automation Startup**

**System Setup**

**Testcase Execution**

**Testcase Output Analysis**

**Testcase Cleanup**

**Results Notification**

**Automation Completion**

**Execution**

**Monitoring**

**Synchronization**

**Resource Management**

# *Example - Software Testing Automation Framework (STAF)*

---



- **Open-source automation framework designed with reusable components, called services**
- **Makes it easier to create automated testcases and workloads**
- **Increases the efficiency, productivity, and quality of testing by improving the level of automation and reuse in individual test cases, as well as in the overall test environment**

## Peer-to-peer, Pluggable Test Automation Framework

- **History**

- Created at IBM Austin in 1998
- Corporate funded (via QSE) since 1999
- Open-sourced (on SourceForge) in 2001
- Licensed under the Common Public License (CPL)

- **User Community**

- 220+ IBM teams (world-wide)
- 180+ external companies
- 145k+ downloads of STAF and associated plug-ins since 4Q/2001
- Consistently ranked as one of the Top 100 projects on SourceForge.net

- **STAF externalizes its capabilities through services.**
- **A service provides a focused set of functionality such as logging, monitoring, or process invocation.**
- **STAF uses few system resources (install 10 – 30 MB, memory size 2.5 – 5 MB, idle 0 percent CPU).**
- **STAFProc is a process that runs on a machine (called STAFClient); it accepts requests and routes them to the appropriate service.**
- **STAFProc can run on a local machine or on a machine in the test environment – peer-to-peer environment.**

- **DIAG** provides diagnostics services *Internal ("DIAG")*
- **DELAY** provides a means to sleep a specified amount of time *Internal ("DELAY")*
- **ECHO** echoes back a supplied message *Internal ("ECHO")*
- **FILE SYSTEM** allows you to get and copy files across the network *Internal ("FS")*
- **HANDLE** provides information about existing STAF handles *Internal ("HANDLE")*
- **HELP** provides Help on STAF error codes *Internal ("HELP")*
- **MISC** handles miscellaneous commands such as displaying the version of STAF that is currently running *Internal ("MISC")*
- **PING** provides a simple is-alive message *Internal ("PING")*
- **PROCESS** allows you to start, stop, and query processes *Internal ("PROCESS")*
- **QUEUE** provides a network-enabled IPC mechanism for STAF Programs *Internal ("QUEUE")*
- **SEMAPHORE** provides network-enabled named event and mutex semaphores *Internal ("SEM")*
- **SERVICE** allows you to list services available on a machine and to examine the requests that have been submitted on a machine *Internal ("SERVICE")*
- **SHUTDOWN** provides a means to shutdown STAF and register for shutdown *Internal ("SHUTDOWN")*
- **Notifications** *Internal ("SHUTDOWN")*
- **TRACE** provides tracing information for STAF services *Internal ("TRACE")*
- **TRUST** interfaces with STAF's security *Internal ("TRUST")*
- **VARIABLE** provides a method for maintaining configuration and runtime data (variables) *Internal ("VAR")*

# STAF External Services

- **CRON** calls into STAF services at a specified time interval *External (Java)*
- **EMAIL** allows you to send email messages *External (Java)*
- **EVENT** provides a publish/subscribe notification system *External (Java)*
- **EVENTMANAGER** allows you to call STAF services when a specified Event occurs *External (Java)*
- **HTTP** allows you to make HTTP requests that can be grouped together in a session *External (Java)*
- **LOG** provides a full-featured logging facility *External (C++)*
- **MONITOR** allows a test case to publish its current running execution status for others to read *External (C++)*
- **RESOURCE POOL** allows you to manage exclusive access to pools of elements, e.g. VM User IDs or Software Licenses *External (C++)*
- **STAX** provides an XML-based execution engine *External (Java)*
- **ZIP** provides a means to zip/unzip/list/delete PKZip/WinZip compatible archives *External (C++)*

Executable code for external STAF services resides outside of STAFProc, for example in a Java jar file, a C++ DLL file, or a Rexx script file.

# ***STAX (Streaming API for XML)***

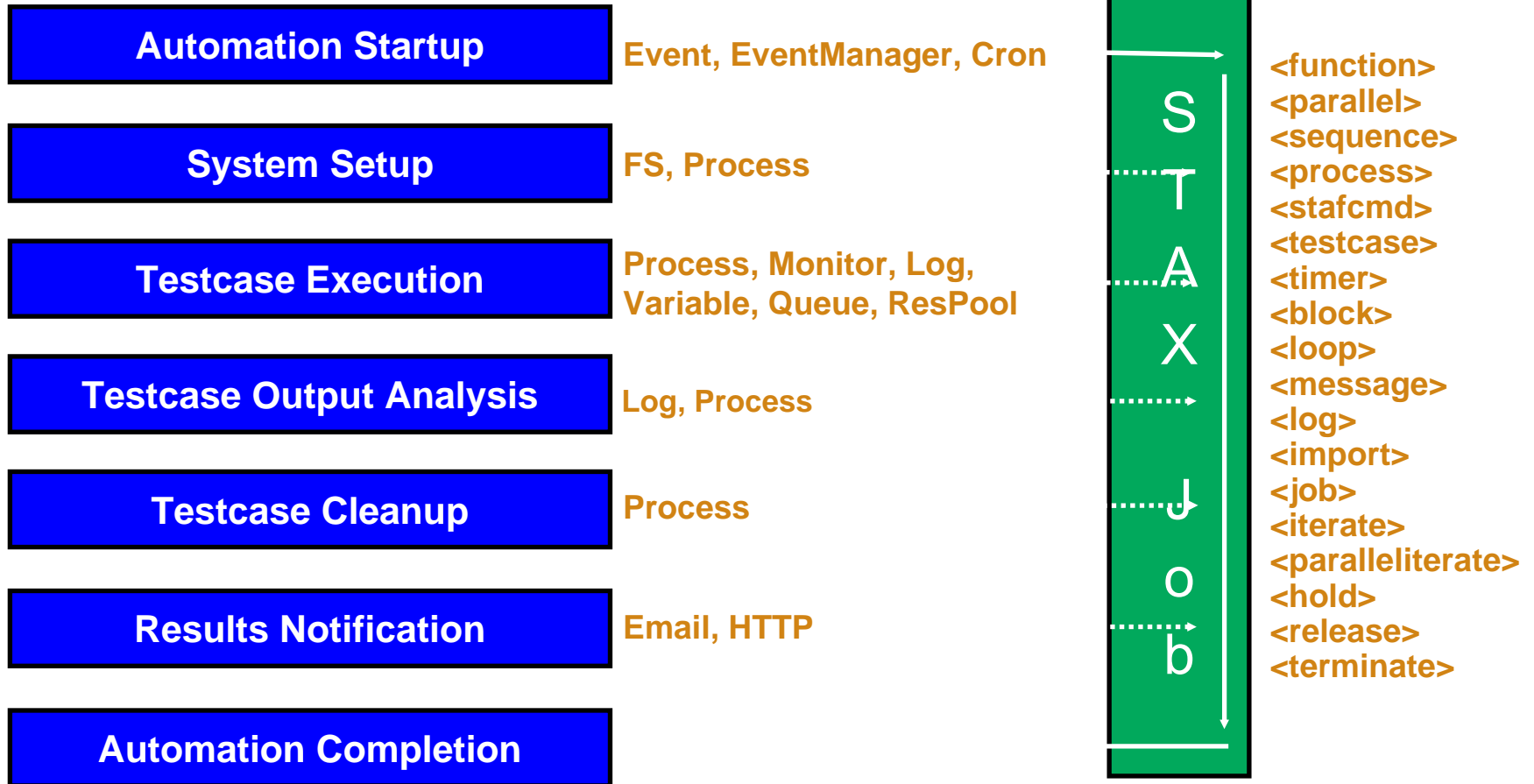
- **STAX is an automation system used to automate and monitor the entire test environment, including:**
  - System/product setup
  - Test case distribution
  - Test case execution
  - Test case results collection and analysis
- **STAX consists of:**
  - A programming language designed for automation; programs written in this language are called jobs
  - An execution engine (similar to an interpreter for other programming/scripting languages), which manages the execution and runtime behavior of the job
  - A GUI application, called the STAX Monitor, which provides a dynamically updated view of executing jobs
  - A tool (STAXDoc) used to generate documentation for STAX jobs

# Technologies Used in STAX

---

- **STAF** – provides the infrastructure on which STAX builds. The full power of STAF and its services is exposed for use within STAX jobs.
- **XML** – is the basis for STAX programming language, providing a built-in structure for jobs and a set of tools for constructing jobs, such as XML (aware) editors and XSLT.
- **Python** – is integrated with Python to provide a rich and accessible data model; this allows access to existing Python libraries.
- **Java** – allows access to existing Java classes/libraries, providing another source for reuse.

# End-to-end Automation with STAF and STAX



- **Technical**

- **Problem complexity increases in later phases of testing**
  - No tool can replace the “operational user” stressing the system.
  - System-level testing may involve non-determinism.
- **Tool maturity**
  - Tool availability lessens as complexity of testing increases.
  - Automated tools industry is not mature (standardization).

# *What is the Way Forward?*

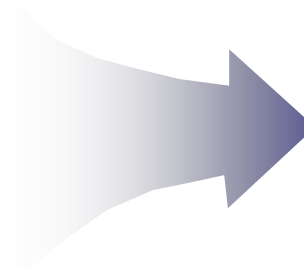
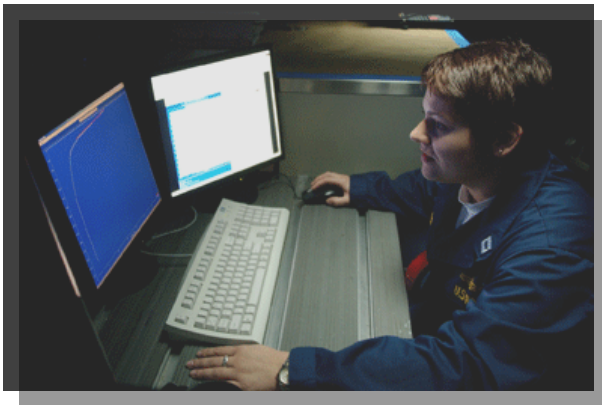
---

**As the ATRTR business and technical process evolves and matures, collaborate with vendors, U.S. Navy industry community, and other interested partners to**

- Continue to define and develop the ATRTR architecture and framework
- Leverage technical and business concepts from the U.S. Navy Open Architecture effort
- Identify and develop standards, as applicable

# Expected Results

- Implement standards to support automated testing
- Develop automated testing architectures and frameworks
- Increase partnerships among industry and government for implementation of automated testing



**Software delivered to the fleet faster, better, and at lower life-cycle costs!**

