STAT COE-Report-05-2021
Case Number: 88ABW-2021-0478

Applying Scientific Test and Analysis Techniques within the DevSecOps Lifecycle

Authored by:

William F. Rowell, PhD

Steven C. Oimoen, PhD

Darryl K. Ahner, PhD, PE

15 April 2021

The goal of the STAT COE is to assist in developing rigorous, defensible test strategies to more effectively quantify and characterize system performance and provide information that reduces risk. This and other COE products are available at www.afit.edu/STAT.

STAT Center of Excellence
2950 Hobson Way – Wright-Patterson AFB, OH 45433

Table of Contents

Executive Summary

Intelligent application of scientific test and analysis techniques (STAT) to Department of Defense (DoD) weapon and business systems has proven to be a key component to enable effective, efficient, and rigorous test & evaluation (T&E) supporting decision making at all levels. Implementation of Development, Security and Operations (DevSecOps) and related modern software development methodologies continues to grow within the acquisition of software intensive DoD systems. Similarly, interest in leveraging STAT in the DevSecOps T&E environment is rising. This paper provides best practices for applying STAT to T&E of software intensive systems using DevSecOps and includes insights into appropriate STAT application techniques.

Keywords: Agile, DevSecOps, scientific test and analysis techniques, automated testing

Introduction

Scientific test and analysis techniques (STAT) are deliberate, methodical techniques (processes and procedures) used to develop effective, efficient, and rigorous test strategies yielding defensible results. STAT applies not only to all phases and types of Department of Defense (DoD) testing (developmental, operational, integrated, and live fire testing) but also to testing performed in programs following any of the six DoD adaptive acquisition framework pathways (urgent capability, middle-tier, major capability, software, defense business systems, and services). STAT encompasses such processes and procedures as design of experiments (DOE), reliability, software test coverage, automated software testing, observational studies, and survey design used within the context of a larger decision support framework. Each technique's suitability depends on the specific test objective(s).

This paper provides best practices for applying STAT techniques to T&E of software intensive systems using the Development, Security and Operations (DevSecOps) lifecycle. First, the paper covers background information on the DevSecOps lifecycle and the core STAT process used to guide the implementation of STAT techniques to the testing process. Then, the paper discusses four areas of DevSecOps testing where STAT can be applied (non-functional requirements, risk-based approach to software test coverage, software development process metrics, and automated software testing). The final section (conclusion) summaries the application of STAT within the DevSecOps lifecycle.

Background

DevSecOps Lifecycle

DoD Instruction (DoDI) 5000.02, Operation of the Adaptive Acquisition Framework, defines six acquisition pathways: Urgent Capability, Middle Tier, Major Capability, Software, Defense Business Systems (DBS), and Defense Acquisition of Services. The Software Acquisition Pathway's purpose is to facilitate rapid and iterative delivery of software capability to the user via modern software

development practices such as Agile Software Development, DevSecOps and Lean Practices (DoDI 5000.02 Operation of the Adaptive Acquisition Framework, 2020).

DoDI 5000.87, Operation of the Software Acquisition Pathway, defines DevSecOps as follows:

> *An organizational software engineering culture and practice that aims at unifying software development, security, and operations. The main characteristic of DevSecOps is to automate, monitor, and apply security at all phases of the software lifecycle: plan, develop, build, test, release, deliver, deploy, operate, and monitor. In DevSecOps, testing and security are shifted left through automated unit, functional, integration, and security testing – this is a key DevSecOps differentiator since security and functional capabilities are tested and built simultaneously*. (DoD Instruction 5000.87 Operation of the Software Acquisition Pathway, 2020)

This paper focuses on DevSecOps as a software engineering practice with a well-defined software lifecycle shown in Figure 1 (DoD Enterprise DevSecOps Reference Design Version 1.0, 2019).

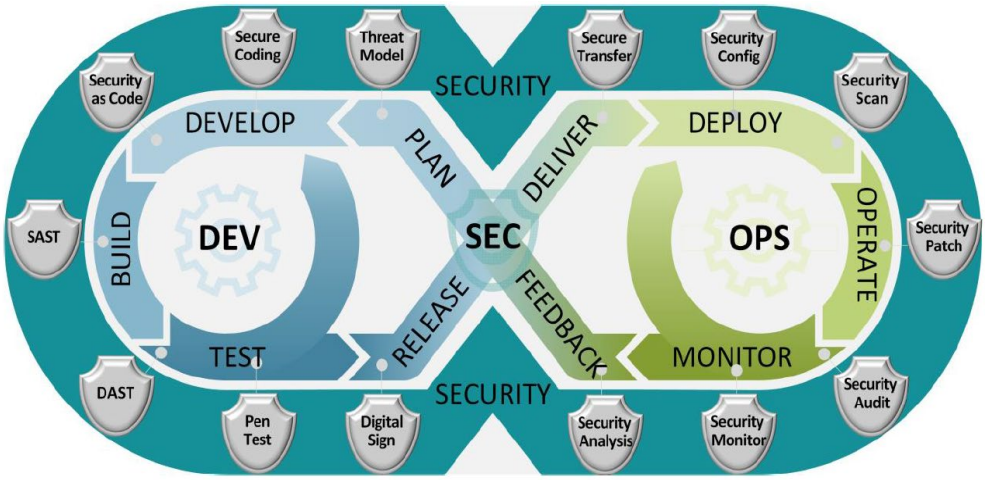

Figure 1. DevSecOps Software Lifecycle

A key assumption of this paper is that the Develop phase of the DevSecOps lifecycle uses an Agile software development process. Agile's key characteristics (flexible planning, evolutionary development, quick delivery, continuous improvement, and responsiveness to change) are consistent with the DevSecOps approach. Figure 2 shows a diagram of a notional Agile software development process.

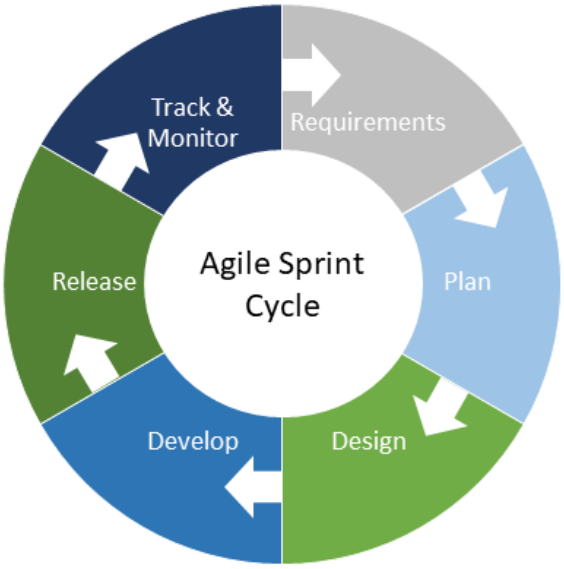

Figure 2. Notional Agile Software Development Process

STAT Process

The STAT Process, as illustrated in Figure 3, is designed to incorporate a high-level system engineering approach to the scientific method to form rigorous test strategies that apply STAT methods and techniques to better inform and manage T&E risks. This process enables STAT practitioners to identify STAT-applicable candidate requirements and associated test objectives, develop appropriate T&E strategies and test designs, and enhance the data analysis that results from such planning. An in-depth discussion and primer on the STAT process is available in the STAT Center of Excellence (COE) best practice "Guide to Developing an Effective Test Strategy" (Burke et al, 2019).

Disciplined adherence to the structured STAT process is essential to overall effectiveness and efficiency of a T&E program. In particular, the STAT process focuses on developing a set of clear test objectives that provide the demand signal for test resources throughout the testing process. Test objectives derived in this manner accurately reflect program goals because they:

- Are derived from requirements and clearly stated
- Reflect focus and purpose of testing and evaluation
- Define scope of follow-on testing
- Form a basis for measures of performance (responses) which are specific, unbiased, measurable, and of practical consequence
- Provide a basis for selecting more powerful analytic capability to inform results

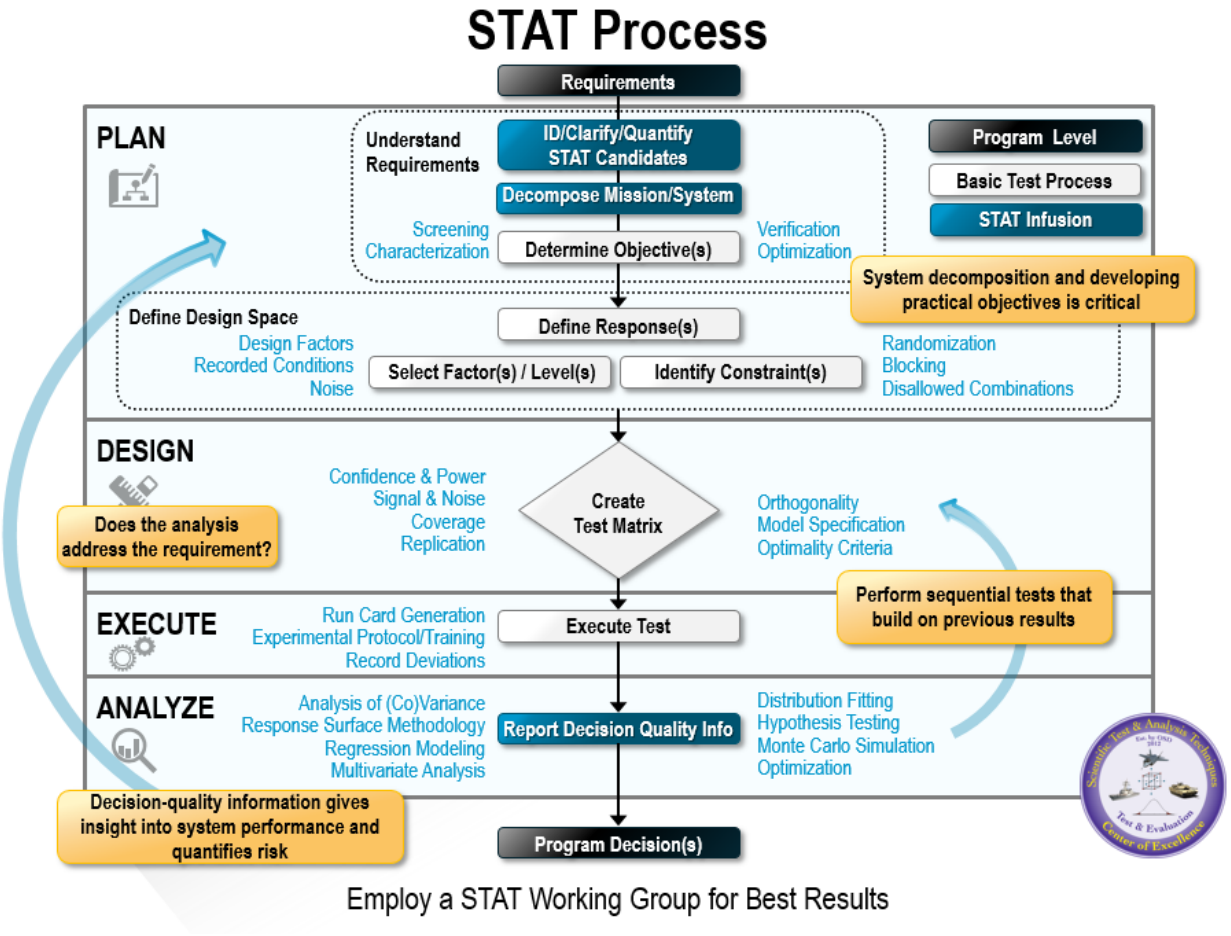

Figure 3. STAT in the T&E Process

Non-functional Requirements (NFRs)

In the Scaled Agile Framework (SAFe®) methodology, which is widely used in DoD software acquisition, NFRs define system attributes such as performance, scalability, reliability, maintainability, usability, and security (Nonfunctional Requirements, 2021). NFRs serve as constraints or restrictions on system design across different backlogs. For DoD systems, NFRs take a variety of forms: Key Performance Parameters (KPPs), Key System Attributes (KSAs) or System Performance Measures (SPMs). Regardless of the specific labels for these types of requirements used by the selected Agile methodology, NFRs are frequently encountered in DoD programs. Therefore, NFRs must be tested rigorously to provide decision makers with the evidence required to verify requirements have been met satisfactorily.

Because of the inherently random nature of processes involving performance, scalability, and reliability, STAT techniques such as Design of Experiments (DoE) and statistical methods are particularly useful in rigorously verifying these requirements. The sections below describe how to apply STAT to these three NFRs.

Performance

In this section, we will do an end-to-end walkthrough of an example of applying STAT to a performance NFR. The STAT planning phase information consists of a response time requirement for retrieving a report, test objectives, a response, constraints, and factors and their associated levels. Note that the test strategy involves a sequence of three test objectives concluding with rigorous verification of the requirement. The test environment is representative of the expected operational environment. If the factor-screening testing identifies any significant factors, set these factors in the requirement verification tests to levels expected to occur in the operational environment. Characterizing response times over the range of levels of significant test factors provide insights into where in the factor space we may encounter performance problems. Table 1 provides test information and Table 2 identifies the STAT planning information (factors and levels).

Table 1. Test Information

Item	Description
Requirement	Report from 80 percent of all queries will be displayed to user within 12 second or less once query is submitted
Test Objectives	1: Screen factors to identify statistically significant factors (those that influence report retrieval response time) 2: Characterize report retrieval response time over range of test factor values 3: Rigorously verify satisfaction of the requirement
Response	Elapsed time from when user submits request until report is displayed to user
Constraints	None
Factors and Levels	See Table 2

Table 2. STAT Test Planning Information – Factors and Levels

Levels/Factors	Report Type	Report Size (MB)	User Location	Download Speed (Mbps)
Level 1	1	5	A	10
Level 2	2	20	B	40

After testing Test Objectives 1 and 2, a summary of key results is shown in Table 3 below.

Table 3. Test Results

Test Objective	Results
1: Factor Screening	• Report Type and User Location are statistically significant • User location is likely to have a negative impact on satisfying requirement • No significant factor interactions
2: Factor/Response Characterization (Predicted Response)	• Significantly increases when User Location is "B" • Exceeds response time when User Location is "B" and Report Type = "2"
3: Test Design	• Report Type and User Location proportions must mirror expected operational profile

Screening identified Report Type and User Location as significant factors; however, no factor interactions were significant. Because this test showed that the levels of both User Location ("A"/"B") and Report Type ("1"/"2") can significantly influence response time, it is vital that levels of these two factors, in the set of requirement verification test runs, accurately reflect their current or estimated levels in the operational environment to ensure the test results mirror real world expectations.

Next, an analysis using historical operational transactions revealed that at peak load 80% of report requests come from User Location "A" and 20% from Location "B" with both user locations requesting an equal percentage of each Report Type, shown in Table 4. The user operational profile created did not take into account the values of the non-significant factors.

Table 4. User Operational Profile (Requests per Hour at Peak Load)

	Report Type "1"	Report Type "2"	Total
User Location "A"	40	40	**80**
User Location "B"	10	10	**20**
Total	**50**	**50**	**100**

The output of Objective 3 testing is an equation (not shown) predicting response time as a function of the significant factor levels. This equation reveals that there is a significant increase in response time when the User Location is "B" and that the predicted response time from the combination of User Location "B" and Report Type "2" exceeds the 12-second response time requirement (see Table 5). From this testing we gain insight into what parts of the factor space are likely to cause problems.

Table 5. Predicted Report Retrieval Response Time Confidence Intervals (seconds)

	Report Type "1"	Report Type "2"
User Location "A"	5.1 - 6.1	9.0 - 10.2
User Location "B"	12.9 - 14.1	19.5 - 20.9

Table 6 shows the requirements verification test design. Note that the relative fraction of both "A"/"B" user locations (80/20) and "1"/"2" report types (50/50) mirrors the user operational profile from Table 4. Not shown is the computation required to determine the minimum sample size (number of replications of the 10 test runs) required to achieve the desired level of confidence (usually 95%).

Table 6. Requirement Verification Test Design

Test Run #	User Location	Report Type
1	A	1
2	A	2
3	B	1
4	A	1
5	A	2
6	A	1
7	A	2
8	B	2
9	A	1
10	A	2

Table 7 depicts the response times of the 10 requirement verification test runs. As expected, the response times for test runs associated with User Location "B" are significantly higher compared to those associated with User Location "A."

Table 7. Requirement Verification Test Results

Test Run #	User Location	Report Type	User Location "A"	Use Location "B"
1	A	1	4.81	
2	A	2	9.84	
3	B	1		12.80
4	A	1	4.18	
5	A	2	10.31	
6	A	1	5.65	
7	A	2	7.66	
8	B	2		19.90
9	A	1	6.21	
10	A	2	8.11	

Figure 4 shows the analysis of the results in Table 7. The JMP tool is used to compute the tolerance interval statistic associated with the hypothesis that 80% of the response times are less than 12 seconds for a confidence level of 95%. Because the statistic (13.51366 seconds) exceeds the requirement of 12 seconds, the hypothesis is rejected, meaning that there is insufficient evidence to verify that the requirement has been satisfied. Obviously, more work is required to reduce the report retrieval response time; however, this sequence of tests has provided insight into where improvement is needed.

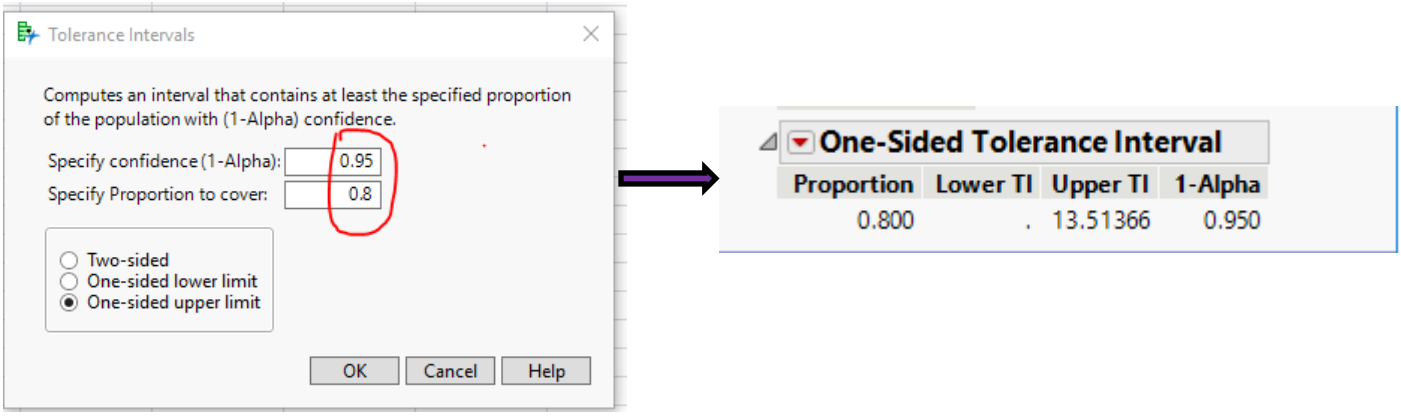

Figure 4. Analytical Results of Requirement Verification Testing

Scalability

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential adaptability to accommodate growth. Sometimes scalability is confused with capacity. Capacity planning is a strategic activity, whereas scalability planning is a more responsive activity. Usually, scalability arises during peak usage activities such as end of fiscal year activities in financial and procurement systems.

Typically, scalability NFRs are stated in terms of the number of concurrent users the system can handle without degrading System Response Time (SRT). Once the load, created by the number of concurrent users, causes SRT to degrade, then the system has reached its scaling limit.

A scalability verification test usually involves the following steps:

1. Incrementally generate users in the same proportion that the user types appear in the set of concurrent users.
2. Wait for system to stabilize after each increment of users is added.
3. Determine when SRT begins to degrade. A clear increase in average transaction response time (latency) or, alternatively, a clear decrease in total transactions per second (throughput) is a good indicator of SRT degradation.
4. Record the number of concurrent users when SRT degradation begins.

As illustrated in the Performance and Reliability sections, rigorously verifying scalability requirements requires:

- Formulating appropriate null and alternative hypotheses (e.g., comparing the number of concurrent users when System Response Time degrades to the required number).
- Selecting an appropriate statistic and level of confidence.
- Computing minimum number of the test runs (minimum sample size).
- Executing load test runs.
- Computing chosen statistic.
- Comparing computed statistic to minimum/maximum value allowed.

For a complex system, a large expenditure of resources is typically required to rigorously set up and run a single concurrent users load test. In these situations, a sensible sequential test strategy is to first perform a single load test and evaluate the margin of safety between the actual and required concurrent users at System Response Time degradation. If the concurrent user safety margin is sufficiently high and System Response Time degradation is not precipitous, it may make sense to stop testing after only one run.

Reliability

Like performance and scalability, reliability is an important system attribute that may be treated as an explicit NFR and can vary randomly based on the state of the system. For example, the success rate of a Single Sign-on capability can depend on the load being placed on the system by users attempting to sign on. Under high loads, when a sign-on server goes down and a large number of users simultaneously try to sign back in when it comes up, the Single Sign-on capability may be overwhelmed and behave unreliably. As an example, a reliability NFR may be written as:

98% of the time users successfully execute the Single Sign-on command

Like the response-time performance NFR previously discussed, this Single Sign-on NFR also requires a statistical test to rigorously evaluate and assess (or verify). The following is a simple example that illustrates how to apply the STAT statistical method, the One Proportion Z-test, to rigorously verify this requirement.

Example—Reliability NFR: 98% of the time users successfully execute the Single Sign-on command

Assuming a Single Sign-on success rate of 98%, minimum sample size computations show approximately 500 sign-ons would be adequate to verify, with 95% confidence, the hypothesis that the success rate for the Single Sign-on population is 98% or higher. The underlying statistical information to apply a z-test (assuming an underlying normally distributed population) is:

- H_0: Single Sign-on success rate is 0.98 or higher (null hypothesis)
- H_A: Single Sign-on rate is less than 0.98 (alternate hypothesis)
- $\hat{p}$ = 0.97 (observed proportion in sample)
- p_0 = 0.98 (null hypothesis value of 98%)
- n = 500 (sample size)
- $z_{.05} = -1.645$

Note that the assumption is the sign-on process is functioning as designed (98% success rate) and this test attempts to find evidence to reject this assumption, that is, the sign-on process has a success rate less than 98%.

Compute z-statistic using the formula below and compare to $z_{.05}$.

$$z = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}$$

We see that z = -.714 > -1.645, and therefore, we fail to reject the null hypothesis that success rate is equal to or greater than 98%.

For more detailed guidance on software reliability, see the STAT COE best practice "Software Reliability Fundamentals Best Practice." (Rowell & Kolsti, 2018)

Risk-based Approach to Software Test Coverage

Background

The SAFe Requirements Model mandates each user story include "acceptance criteria" that are validated via story acceptance and unit tests (SAFe Requirements Model, 2021). Government testers are becoming increasingly involved in the Agile process and are expected to make judgements whether to accept testing conducted at the sprint level as adequate. Thus, any implementation of the DevSecOps

methodology must ensure testing is not only "adequate", but more importantly testing which can be demonstrated to be adequate using meaningful quantitative measures, such as test coverage measures.

Below are some examples of potential "test coverage" metrics:

- Percentage of tests that are automated
- Number of automated tests per capability, feature, epic, and/or story
- Percentage test coverage of capabilities, features, epics, and/or stories
- **Percentage test coverage of all possible test cases in a capability, feature, epic, and/or story**
- Percentage test coverage of code (by module or line)
- Number of cybersecurity audit/penetration tests
- Percentage of cybersecurity audit/penetration tests that are automated

All of the above "test coverage" metrics can be useful in understanding the state of Agile testing. However, only the **bolded** metric provides meaningful quantitative information about the amount of risk associated with the state of current Agile testing compared to the risk of executing all of the possible test cases at a specified level (capability, feature, epic, story). This metric assumes that we have identified all possible Agile test cases at the specified level and know exactly what test cases have been executed and what test cases have not been executed. The untested cases represent risk that we have implicitly accepted because the potential problems associated with the untested test cases have not been identified yet. Given inadequate resources to test all possible Agile test cases at a specified level, the challenge is to find a set of test runs that minimizes the risk associated with untested test cases. The next section presents an example of how to optimize software test coverage while explicitly quantifying the risk associated with untested test cases.

Optimizing Software Test Coverage in a User Story

Assume that an Agile team needs to develop a test design for a user story involving a parts ordering form with five input fields: Availability, Delivery Mode, Urgency, Delivery Location, Funding Source (Table 8).

Table 8. Parts Ordering Form for User Story

Availability	Delivery Mode	Urgency	Delivery Location	Funding Source
Available	Military Plane	Overnight	CONUS	Working Capital Fund
Back-Ordered	Re-supply Ship	2-5 Days	OCONUS-E	General Fund
Discontinued	USPS	6-10 Days	OCONUS-W	Transaction Fund
Replaced	UPS	> 10 Days		

With each of the first three fields having four different choices in the dropdown and the remaining two each having three different choices, there are a maximum of 576 (4x4x4x3x3) possible test cases to be

executed, assuming all combinations of choices are feasible. Table 9 provides a complete analysis of the interactions in the user story form.

Table 9. Interaction Analysis of Parts Ordering Form

Item	Number
Factors	5
Levels per Factor	3-5
Total Levels Across All Factors	18
Total Possible Combinations (w/o constraints)	576
1-Way Interactions	18
2-Way Interactions	129
3-Way Interactions	460
4-Way Interactions	816
5-Way Interactions	576

Because of resource and other limitations, it is frequently impossible to execute all of the possible test cases (exhaustive testing) in this kind of situation. The questions to be answered are how many test cases to execute and which test cases to execute. Fortunately, empirical studies have shown that regardless of the actual number of factors in a form, a very high percentage of software faults arise from the interaction of a small number of factors (6 or less) (Kuhn et al, 2004).

Combinatorial Optimization (CO) is an advanced mathematical technique that can identify a significantly smaller (less than exhaustive) number of test cases based on identifying 6-way or less factor interactions that are highly likely to cover a high percentage of software faults. The National Institute of Science and Technology (NIST)-has developed a CO tool, Automated Combinatorial Testing for Software (ACTS), that can quickly identify effective and efficient sets of test cases over a range of different t-way factor interactions (Automated Combinatorial Testing for Software, 2021).

Figure 5 shows the 16-test case set and associated interaction coverage metrics for application of the ACTS 2-way factor solution while Figure 6 shows the ACTS 3-way factor solution of 64 test cases—only 42 test cases are shown in Figure 6.

The interaction coverage metrics at the bottom of each figure explicitly quantify the risk of not finding defects associated with each test design in terms of the number/percentage of covered/uncovered interactions. Thus, ACTS provides the decision maker with valuable tradeoff information to help allocate testing resources. See the STAT COE best practice on combinatorial design for more detailed guidance on combinatorial testing. (Bush & Ortiz, 2014)

Test Result | Statistics

	AVAILABILITY	DELIVERYMODE	URGENCY	DELIVERYLOCATION	FUNDINGSOURCE
1	Available	Military plane	2-5 days	OCONUS-East	General Fund
2	Available	Re-supply ship	6-10 days	OCONUS-West	Transaction Fund
3	Available	USPS	>10 days	CONUS	Working Capital Fund
4	Available	UPS	Overnight	OCONUS-East	Transaction Fund
5	Back-ordered	Military plane	6-10 days	CONUS	Working Capital Fund
6	Back-ordered	Re-supply ship	>10 days	OCONUS-East	General Fund
7	Back-ordered	USPS	Overnight	OCONUS-West	General Fund
8	Back-ordered	UPS	2-5 days	CONUS	Transaction Fund
9	Discontinued	Military plane	>10 days	OCONUS-West	Transaction Fund
10	Discontinued	Re-supply ship	Overnight	CONUS	Working Capital Fund
11	Discontinued	USPS	2-5 days	OCONUS-East	Working Capital Fund
12	Discontinued	UPS	6-10 days	OCONUS-West	General Fund
13	Replaced	Military plane	Overnight	CONUS	General Fund
14	Replaced	Re-supply ship	2-5 days	OCONUS-West	Working Capital Fund
15	Replaced	USPS	6-10 days	OCONUS-East	Transaction Fund
16	Replaced	UPS	>10 days	CONUS	Working Capital Fund

Factor Interaction Coverage:

1-way factor interactions 18/18 100%
2-way factor interactions 129/129 100%
3-way factor interactions159/460 35%
4-way factor interactions 80/816 10%
5-way factor interactions 16/576 3%

Figure 5. Analysis of 2-way Factor Interaction Solution (16 Test Cases)

Test Result | Statistics

	AVAILABILITY	DELIVERYMODE	URGENCY	DELIVERYLOCATION	FUNDINGSOURCE
1	Available	Military plane	Overnight	CONUS	Working Capital Fund
2	Available	Military plane	2-5 days	OCONUS-East	General Fund
3	Available	Military plane	6-10 days	OCONUS-West	Transaction Fund
4	Available	Military plane	>10 days	CONUS	General Fund
5	Available	Re-supply ship	Overnight	OCONUS-East	Transaction Fund
6	Available	Re-supply ship	2-5 days	OCONUS-West	Working Capital Fund
7	Available	Re-supply ship	6-10 days	CONUS	General Fund
8	Available	Re-supply ship	>10 days	OCONUS-East	Working Capital Fund
9	Available	USPS	Overnight	OCONUS-West	General Fund
10	Available	USPS	2-5 days	CONUS	Transaction Fund
11	Available	USPS	6-10 days	OCONUS-East	Working Capital Fund
12	Available	USPS	>10 days	OCONUS-West	Transaction Fund
13	Available	UPS	Overnight	CONUS	General Fund
14	Available	UPS	2-5 days	OCONUS-East	Working Capital Fund
15	Available	UPS	6-10 days	OCONUS-West	Transaction Fund
16	Available	UPS	>10 days	CONUS	Working Capital Fund
17	Back-ordered	Military plane	Overnight	OCONUS-East	General Fund
18	Back-ordered	Military plane	2-5 days	OCONUS-West	Transaction Fund
19	Back-ordered	Military plane	6-10 days	CONUS	Working Capital Fund
20	Back-ordered	Military plane	>10 days	OCONUS-East	Transaction Fund
21	Back-ordered	Re-supply ship	Overnight	OCONUS-West	Working Capital Fund
22	Back-ordered	Re-supply ship	2-5 days	CONUS	General Fund
23	Back-ordered	Re-supply ship	6-10 days	OCONUS-East	Transaction Fund
24	Back-ordered	Re-supply ship	>10 days	OCONUS-West	General Fund
25	Back-ordered	USPS	Overnight	CONUS	Transaction Fund
26	Back-ordered	USPS	2-5 days	OCONUS-East	Working Capital Fund
27	Back-ordered	USPS	6-10 days	OCONUS-West	General Fund
28	Back-ordered	USPS	>10 days	CONUS	Working Capital Fund
29	Back-ordered	UPS	Overnight	OCONUS-East	Transaction Fund
30	Back-ordered	UPS	2-5 days	OCONUS-West	General Fund
31	Back-ordered	UPS	6-10 days	CONUS	Working Capital Fund
32	Back-ordered	UPS	>10 days	OCONUS-East	General Fund
33	Discontinued	Military plane	Overnight	OCONUS-West	Transaction Fund
34	Discontinued	Military plane	2-5 days	CONUS	Working Capital Fund
35	Discontinued	Military plane	6-10 days	OCONUS-East	General Fund
36	Discontinued	Military plane	>10 days	OCONUS-West	Working Capital Fund
37	Discontinued	Re-supply ship	Overnight	CONUS	General Fund
38	Discontinued	Re-supply ship	2-5 days	OCONUS-East	Transaction Fund
39	Discontinued	Re-supply ship	6-10 days	OCONUS-West	Working Capital Fund
40	Discontinued	Re-supply ship	>10 days	CONUS	Transaction Fund
41	Discontinued	USPS	Overnight	OCONUS-East	Working Capital Fund
42	Discontinued	USPS	2-5 days	OCONUS-West	General Fund

Factor Interaction Coverage:

1-way factor interactions 8/18 100%
2-way factor interactions 129/129 100%
3-way factor interactions 460/460 100%
4-way factor interactions 316/816 39%
5-way factor interactions 65/576 3%

Figure 6. Analysis of 3-way Factor Interaction Solution (64 Test Cases)

Recommendations

To make the most of the test coverage approach described in this section requires at least one Agile team members trained in both recognizing situations (user stories, case studies, lessons learned) with large numbers of possible test cases and evaluating the percentage of test cases actually covered by a proposed test design. This capability enables the team to determine the amount of risk associated with their test design. Ideally, someone on the team will be trained to use a tool like ACTS to optimize test coverage within the constraints of available test resources rather than having to manually guess at a set of test runs with good coverage. At a minimum, the self-organizing Agile team members must reach a consensus that test coverage is an important attribute of a test design that must be formally evaluated and optimized for each user story.

Software Development Process Metrics

Metrics that are unambiguous, quantifiable, and measurable are helpful to the decision maker during the software development process. However, the most important property of software development metrics or any metrics is that driving the value of these metrics in the right direction improves the desired outcomes of the software development process. Rigorously applying the STAT process to the definition, refinement, measurement of software development metrics, and assessment of their predictive power of can produce invaluable metrics for the decision maker.

Below is a list of software development process metrics that are promising candidates for rigorous application of the STAT process:

- **Time to production:** total time for new software features to be running in production environment
- **Lead-time:** total time to deliver and deploy a new capability to the production environment
- **Deployment speed:** total time to deploy a new software feature into production environment
- **Deployment frequency:** how often new releases are deployed into production environment
- **Production failure rate:** frequency application fails in production environment
- **Time to recovery:** total time from failure to recovery of application in production environment

Automated Software Testing

This section summarizes STAT COE guidance for DoD managers and practitioners considering or already applying test automation to software intensive systems. It emphasizes the need to first perform a comprehensive return on investment (ROI) analysis in order to make a solid business case for automation followed by the application of a systems engineering approach based on the scientific method to implement the chosen automation capability. The latest STAT COE best practices on automated software testing include the following titles:

- “Automated Software Testing Implementation Guide for Practitioners and Managers” (Simpson et al, 2018)

- "Automated Software Testing Practices and Pitfalls" (Pestak & Rowell, 2018)

Guidance Objectives

This guidance addresses the following learning and implementation objectives:

- Ensuring automation improves test effectiveness and efficiency at an affordable cost
- Understanding how to best implement test automation
- Understanding the technical components and required staffing for automation
- Building an effective Test Automation Solution
- Increasing the likelihood that automation is applied whenever and wherever it makes sense
- Ensuring automation is resilient and adaptable to change
- Creating modular components to replicate automation within and across programs
- Learning what resources are available to support automation

Lifecycle

This guidance is organized around the implementation phases depicted in Figure 7, which are intended to encompass the lifecycle of automated software testing as it applies to the roles of managers and practitioners.

Although the phases can be visualized and enacted in a chronological or linear fashion, we realize and stress that there is significant connectivity between them making moving in a less structured or iterative direction frequently advisable. The suggested approach also involves maturing several important automation tasks across multiple phases. For example, automation tool selection is often considered a primary and critical decision. Tool selection and tool acquisition should be part of each of the plan, design, and execute phases. This practice enables increased knowledge and topic maturity in subsequent phases. Iteration and looping of the phases is a key to success.

Assess	Plan	Design	Test	Analyze & Report	Improve & Maintain
Is automation feasible?	What should we automate?	What tools, framework, personnel, data collection system?	Where are the errors in SUT and automation?	How good is automation performance?	What needs to be done to grow?

Figure 7. Automation Lifecycle Phases for Managers and Practitioners

Minimum Set of Automation Tasks

If time is restricted and an automation decision must be made quickly, along with short turn preparation for automation be sure to at least consider the following tasks:

- Research automation opportunities and learn which automation tools are best. Know the costs.

- Obtain leadership support by presenting the ROI and quickly identify the barriers to success.
- Learn which parts of your testing are best for automation, design the automation framework, and determine the suite of tools needed for your automation program.
- Find, hire, or grow the automation expertise. Growing can be easier than you think.
- Start with simple automation tasks and increase complexity as automation capability matures.
- Use STAT methods to optimize coverage of the input test space and get the most out of the output from automated tests.
- Select the automation frequency based on the software development cycle and testing needs.
- Understand that maintenance can be the most costly phase and require the most resources.

Assessment of Automated Software Testing Implementation

A best practice for automated software testing (AST) programs is to conduct periodic independent, external assessments to identify improvement opportunities. The STAT COE has developed and implemented the Test Automation Process Insights (TAPI) assessment service to support this best practice for DoD organizations. The current implementation of this service contains the following major tasks:

- Pre-visit webinar to organization on opportunities to optimize automated testing and tools
- Pre-visit request to organization for information on current AST program
- Planning & Design
- Setup & Execution
- Validation and Reporting
- Site visit to gather additional information from organization on current AST efforts
- Analysis & Recommendations report

Conclusion

The STAT process along with its methods and techniques are powerful enablers to ensure that a program's DevSecOps testing process is effective, efficient, defensible, and resilient. Analytical approaches using DOE and other statistical methods are essential for rigorous testing of performance, reliability, and scalability NFRs. For complex user story test designs, combinatorial optimization techniques enable both explicit identification of testing risk and optimal test coverage. Rigorously derived software development process metrics consistent with process goals provide the decision maker with invaluable information for making sound, defensible decisions. Implementation of automated software testing following the STAT best practices within the DevSecOps lifecycle significantly increases the likelihood of achieving the desired ROI. In conclusion, the STAT best practices presented in this paper and referenced on the STAT COE website provide a solid, well-documented foundation for initiating and improving the T&E of software intensive systems using the DevSecOps lifecycle.

References

Automated Combinatorial Testing for Software. (2021, March 18). Retrieved from National Institute of Science and Technology Computer Security Resource Center: https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software

Burke, S. et al (2019, December 31). *Guide to Developing an Effective Test Strategy.* Retrieved from Scientific Test and Analysis Techniques Center of Excellence Best Practices and Test Planning Guides: https://www.afit.edu/STAT/statdocs.cfm?page=1126

Bush, B., & Ortiz, F. (2014, March 25). *Combinatorial Test Designs.* Retrieved from Scientific Test and Analysis Techniques Center of Excellence Best Practices and Test Planning Guides: https://www.afit.edu/STAT/statdocs.cfm?page=1126

DoD Enterprise DevSecOps Reference Design Version 1.0. (2019, August 12). Retrieved from Chief Information Officer, U.S. Department of Defense: https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf

DOD Instruction 5000.02 Operation of the Adaptive Acquisition Framework. (2020, January 23). Retrieved from https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500002p.pdf?ver=2020-01-23-144114-093

DOD Instruction 5000.87 Operation of the Software Acquisition Pathway. (2020, October 2). Retrieved from https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodi/500087p.PDF?ver=virAfQj4v_LgN1JxpB_dpA%3D%3D

Kuhn, R. et al. (2004). Software Fault Interactions and Implications for Software Testing. *IEEE Transactions on Software Enginering, 30*(6), 418-421.

Nonfunctional Requirements. (2021, March 18). Retrieved from Scaled Agile Framework: https://www.scaledagileframework.com/nonfunctional-requirements/

Pestak, T., & Rowell, B. (2018, September 30). *Automated Software Testing Practices and Pitfalls.* Retrieved from aScientific Test and Analysis Techniques Center of Excellence Best Practices and Test Planning Guides: https://www.afit.edu/STAT/statdocs.cfm?page=1126

Rowell, B., & Kolsti, K. (2018, November 9). *Software Reliability Fundamentals Best Practice.* Retrieved from Scientific Test and Analysis Techniques Center of Excellence Best Practices and Test Planning Guides: https://www.afit.edu/STAT/statdocs.cfm?page=1126

SAFe Requirements Model. (2021, March 18). Retrieved from Scaled Agile Framework: https://www.scaledagileframework.com/safe-requirements-model/

Simpson, J. et al. (2018, October). *Automated Software Testing Implementation Guide for Managers and Practitioners.* Retrieved from Scientific Test and Analysis Techniques Center of Excellence Best Practices and Test Planning Guides: https://www.afit.edu/STAT/statdocs.cfm?page=1126