Automated Software Test Implementation Guide for Managers and Practitioners

Authored by: Jim Simpson, Jim Wisnowski, and Andrew Pollner

October 2018

The goal of the STAT T&E 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 https://www.afit.edu/STAT/.

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

1. Executive Overview

This guide is intended to serve those in the Department of Defense (DoD) 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.

Who is the intended reader? Some organizations are considering automation for the first time and are interested in knowing more about the automation concept and the steps needed to make automation a reality for their program. Others have implemented some aspects of automation and may be interested in either applying automation more broadly or are looking for additional opportunities to automate across the acquisition lifecycle and/or within the test process phases. In both cases, the key personnel that must understand how to implement automation are the responsible managers and practitioners. This guide addresses each of these roles separately because each has different focus areas, tasks and responsibilities associated with bringing life to a purpose-built, maintainable and extensible test automation solution.

This implementation guide addresses the following 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

Common objectives when considering or applying test automation to software intensive systems are to select additional system functions or capabilities to automate, to secure more automators, or to change or grow the capabilities of the tools used. Additionally, there are a wealth of resources from groups with automation experience and guidance is readily available in texts and online. Some software acquisition programs have had some exposure to, or experience in automation and are interested in improving one or more aspects of their automation process.

The guide is organized around the implementation phases listed/depicted in Figure 1.1, which are intended to encompass the lifecycle of automated software testing, as it applies to the roles of managers and practitioners within your test program.

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 1.1. Test Automation Lifecycle Phases for 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. This guide suggests tool selection and tool acquisition 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.

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.

Table of Contents

List of Figures

List of Tables

2. Use of the Guide

This guide is divided into two stand-alone sections corresponding to the separate, but connected roles of manager and practitioner. Readers with one of these specific roles can focus on the appropriate section, along with the Introduction section which depicts the relationship between the tasks of each role.

At the start of each phase, checklists for both automation roles provide the reader a concise, sequential yet comprehensive summary of the tasks/sub-tasks required in the phase. Readers not having one of these specific roles can scan the checklists to gain an appreciation for and high-level understanding of the automation roadmaps and steps required to successfully automate testing.

Another handy feature of this document is the frequent placement of summaries, or ***Bottom Lines***, throughout to highlight various activities and concisely capture the essence of a task within a phase. Additionally, inputs and deliverables are stated for each task and can be used as a checkpoint to ensure the starting and ending conditions for tasks are met. The Microsoft Word "Find" function (e.g., CTRL-f) will allow you to quickly locate all the ***bottom lines***, inputs, and deliverables, which can serve as a brief synopsis of the vital tasks to undertake when planning for or conducting software test automation. This search feature can also screen to find relevant content based on user roles, automation maturity, automation phase, and other attributes.

The desire and intent is to distribute this guide widely and solicit feedback so that the future versions can be continually improved and redistributed via the Office of the Secretary of Defense's (OSD) Scientific Test & Analysis Techniques Center of Excellence (STAT COE) website[1]. It is also hoped that this guide and others like it (e.g., Automated Software Testing State of DoD and Industry[2]) may be of value to the AST community as a useful reference. Ultimately, we desire to see improved communication and better collaboration among AST professionals thereby connecting like-minded people, projects, and interests.

The material contained herein is based on industry best practices and sourced from test automation subject matter experts (SMEs), published material including peer-reviewed journal articles, conference materials, textbooks, and increasingly from direct conversation with those of you in the DoD taking advantage of automation for testing software intensive systems. The STAT COE is available to assist you as needed and can put you in touch with groups or experts willing to assist as you move towards automated software testing.

[1] Please visit https://www.afit.edu/STAT for additional information.
[2] Please contact the STAT COE for distribution of this document via coe@afit.edu.

3. Introduction

One of the most widespread advances in industry over the last decade of software testing is the rapid growth of automated test solutions. Automated Software Testing (AST) has had significant impact across the Department of Defense (DoD) and industry. The DoD has not taken full advantage of the efficiencies and improved performance possible using automated approaches across the software development lifecycle. The failure to take full advantage of AST methods is connected to the DoD's lack of understanding of the AST process. As the DoD moves toward improving automation capability, there is a need for a comprehensive, yet succinct guide to the AST process—much like tactics, techniques, and procedures (TTPs) used for military operations. The purpose of this Implementation Guide is to provide managers and practitioners a handbook that outlines a reusable framework to employ AST methods across a variety of DoD systems.

Successful test automation programs require an organization-wide commitment. Various roles within the organization can make the test automation practice a successful and sustainable endeavor. The broadly defined roles of Managers and Practitioners in the overall test automation process are described with specific tasks, inputs, deliverables, and key takeaways throughout each phase of the automation cycle.

This effort is part of the Office of the Secretary of Defense's Scientific Test and Analysis Techniques Center of Excellence (STAT COE) initiative to better educate programs on the benefits of automated test. The purpose of this manual is to describe the general flow of an AST program from end-to-end and provide insights to activities that will lead to a successful automation effort. The guide describes detailed tasks within the six primary AST phases: *assess, plan, design, test, analyze and report,* and *maintain and improve*. These phases allow programs to comply with the DoDI 5000.02 (Enclosure 4, paragraph 5.a.(12))[3] requirements for a test automation strategy. These phases are not necessarily sequential, as Figure 1.1 shows; rather, the activities for successful AST require an iterative approach.

The intended audience is leadership (both program and test), system engineers, software engineers, software developers, software testers, and test automators. The tasks are described at a general level and technical details are explained from the vantage point of someone with little knowledge of software test and automation. The AST process flow was developed primarily from interviews with experts across DoD and industry who have had success and failure automating test cases. Additional sources include previous DoD studies, textbooks, technical journals, websites, blogs, and conference briefings. Though every program has unique experiences in the automation journey, there are many common elements that have formed the basis for this recommended methodology for DoD systems.

[3] Available at https://www.acq.osd.mil/fo/docs/500002p.pdf.

4. Implementation Guide Task Crosswalk for Manager and Practitioner

With the understanding that the detailed guides for the manager and the practitioner immediately follow this section – it was assumed that the reader may be asking, "how does a certain task that one automation role (e.g. practitioner) has relate to what one or more tasks of the other role (e.g. manager) might be doing?" Neither one of these roles will be executed in isolation; nor will many of the tasks that must be accomplished must be executed in order to serve the other automation role. Thus, an effort was made to generate a "crosswalk" that shows the relationships between and among tasks for the two roles. To better understand the crosswalk concept between practitioners and managers, consider the assess phase tasks for each position. The manager's first task is to evaluate the system under test (SUT) from a program perspective, as it relates to the potential to use automation, left side of Figure 4.1. The practitioner (right side of Figure 4.1) meanwhile is relying on technical expertise and automation experience to determine how automation might best be used for the SUT. Communication between the two sharing their different perspectives and information, can assist the other in completing their tasks. Likewise, the practitioner's detailed ROI study for automation will consider and compare alternative automation tools. This information is expected to be provided to the manager in order to evaluate and recommend the appropriate automation tools.

Figure 4.2 provides an overview of not only the tasks associated with each automation phase for both the manager and the practitioner, but also shows the connectivity within and between phases. At a very high level, the manager is busy early on strategically planning and leading the design of the Test Automation Solution (TAS), while the practitioner is studying the system(s) under test (SUT), identifying automation opportunities while considering potential automation resources, architectures, and software tool requirements. This figure also serves to show all the tasks in one picture so that the reader can gain a better appreciation for everything involved at one time, which jump starts the automation procedure's mental processing and assimilation.

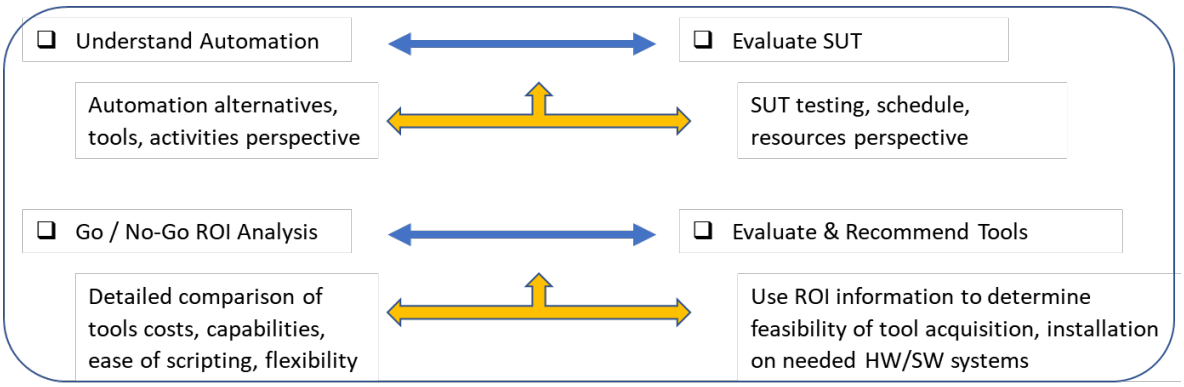

Figure 4.1. Example Crosswalk for Managers and Practitioners (Access Phase)

	Manager	Practitioner
Assess	❑ Understand Acquisition Landscape ❑ Understand Automation ❑ Identify Organizational Fit ❑ Go / No Go Return on Investment Analysis	❑ Evaluate System Under Test (SUT) ❑ Understand the Test Environment ❑ Evaluate & Recommend Test Automation Tools
Plan	❑ Request Project Support ❑ Determine Requirements to Automate ❑ Identify and Acquire Resources and Tools ❑ Define Roles ❑ Chart a Path to Success	❑ Determine Test Automation Strategy ❑ Define Automation Lifecycle ❑ Identify Initial Automation Candidates ❑ Identify Automation Resources Needed
Design	❑ Develop the Test Plan ❑ Procure Resources ❑ Manage Planning and Execution of Initial Pilot ❑ Conduct Design Review	❑ Construct Test Automation Architecture ❑ Specify Test Automation Technical Approach ❑ Capture Test Steps of Operator's SUT ❑ Develop Automation Scripts ❑ Construct Test Automation Framework ❑ Conduct Pilot Project ❑ Establish Test Automation Solution
Test	❑ Approve and Schedule Project ❑ Manage to Plan ❑ Identify and Manage Variances	❑ Verify Test Automation Solution is Working ❑ Verify Automated Tests Against SUT ❑ Consider and Decide on Test Oracle ❑ Execute Automation ❑ Clean up Test Automation
Analyze	❑ Determine Automation Effectiveness ❑ Lead Analysis Plan ❑ Manage Defect Reporting Process ❑ Review Costs ❑ Communicate Results	❑ Analyze Output Data Artifacts ❑ Develop Test Automation Reports ❑ Determine Failure Causes ❑ Develop and Quantify Metrics and Measures
Improve	❑ Standardize and Document Approach ❑ Replicate Approach ❑ Maintain Automation Capability ❑ Manage Continuous Improvement	❑ Transition Automation ❑ Manage the Test Automation Solution (TAS) ❑ Update Automation Code ❑ Manage and Optimize Scripts ❑ Identify Alternative Execution Technologies

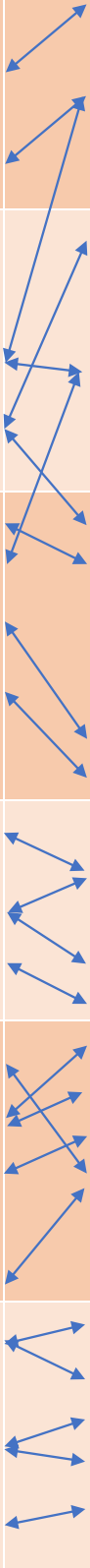

Figure 4.2. Tasks and Crosswalk for Managers and Practitioners

5. Implementation Considerations for Management

Automation managers must enthusiastically lead the automation program with active participation. They have a distinctly different role from the practitioner across each of the phases. They are responsible for the usual program management functions of planning, tracking, and managing cost, schedule, and performance. They must also be more engaged technically than other program managers to understand the impacts of less or more automation. Finally, because automation often requires a culture change, automation managers must be leaders continuously championing the automation cause.

Figure 5.1 below shows some of the major tasks across each of the phases for management to consider. It should be emphasized, that these phases may appear to be sequential much like a waterfall software development model, but managers often will execute tasks simultaneously across multiple phases.

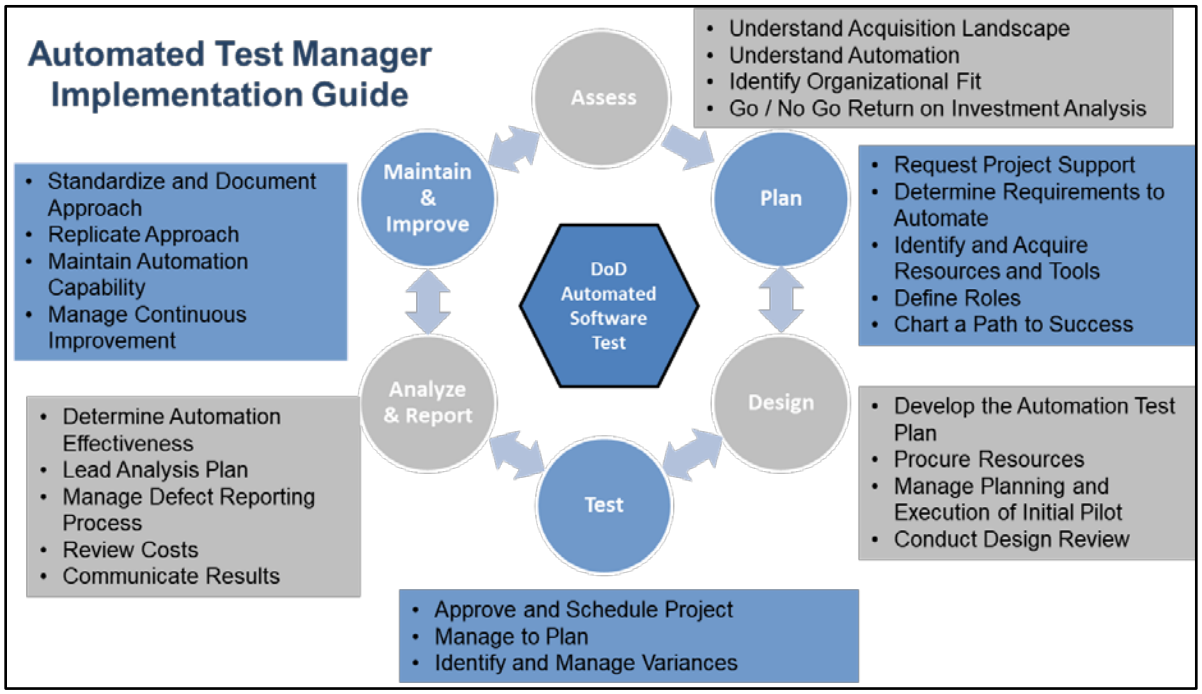

Figure 5.1. Phases and Tasks of Automated Software Test for Managers

5.1. Assess Phase

Manager Checklist for:
Assess Phase

- ❑ **Understand Acquisition Landscape**
  - ❑ Mine the goals, objectives and requirements of the system under test for automation opportunities
  - ❑ Determine the automated test types and test objectives (e.g. unit, functional, integration, or performance) based on the acquisition lifecycle (DT, OT, etc.)
  - ❑ Know limitations based on software development model (contractor vs government, black box vs white box)
- ❑ **Understand Automation**
  - ❑ Meet with internal and/or external experienced automators to better understand appropriate automation practices and learn how automation can best serve your program
  - ❑ Research automation via internet searches, ASTQB, ISTQB, ASQ[4] and other professional societies, and commercial tool vendor sites.
  - ❑ Research automation performed in like-organizations or on like-systems under test (e.g. JMPS, DCGS), and elsewhere
  - ❑ Learn what test automation is being performed by the contractor on the system under test
- ❑ **Identify Organizational Fit**
  - ❑ Gauge and document the keys to organization culture change and adoption of test automation efforts via personal interview within and outside your organization
  - ❑ Obtain the rough order of magnitude state of automation capability and potential (via training) in the organization
  - ❑ Determine the options for increasing the automation capability
- ❑ **Go / No Go Return on Investment Analysis**
  - ❑ Determine the estimated costs of automation. Any schedule delays should also be converted to estimated costs.
  - ❑ Quantify and estimate the benefit that will be realized in test automation
  - ❑ Perform a return on investment analysis
  - ❑ Form a small team to review the costs and benefits to recommend a Go / No Go decision

Figure 5.2. Manager Checklist for Assess Phase

Assessment allows us to evaluate if automation makes sense for the project and the program. Automation is not just about the tools, but also needs to consider the skills of those using the tools, the ability to support the tool infrastructure, and the time required to generate the automation solution. The assess phase, Figure 5.2, can be thought of as a pre-planning phase whose activities cover all aspects of automation immediately preceding the task of starting the automation project. The assessment phase is important in that it helps the project or program understand what challenges may lie ahead with automation. It provides a gap analysis that enables each individual project or program to

[4] See Appendix B: Certification, Education, and Resources for AST

determine a go-forward plan. In some cases, when the organization is not setup to continue (due to budget, staffing, or other constraints), assessment can result in a decision *not* to continue with automation at this point in time. As with any new technology approach, there is often a complete change to how testing is done under automation control vs. how it is traditionally performed in a manual test environment. Organizations need to be prepared for this paradigm shift and practitioners need to realize that the automation is a part of a larger testing strategy and everything needs to continue to align. This phase includes everything from the development of automated tests to deployment of the technology and maintenance.

Automation presents a significant effort to the organization, but also provides a high reward in efficiency and effectiveness. Once a test team begins to use automation in their testing, the drudgery of manual testing diminishes while the analytical problem-solving test scenarios increase. Test automation has the potential to make testers more productive and more satisfied with the work that they do by diminishing the amount of repetitive key pressing traditionally done during the manual test execution phase. The system(s) under test (SUT) is the target for the automated testing. Understanding the SUT construction and components is key and early requirement in the assessment process. Similarly, what test tools are available and compatible with the SUT will be key to realizing an automation solution. Selecting the correct tool for the SUT sets a project or program off to a good start with the first milestone in measuring a project's automation potential.

The manager ultimately must weigh the benefits and costs of automating. Every organization and test program is different, but all should consider the common approach to making the go/no go decision covered in the Assess Phase tasks. Automation should be approached from a long-term perspective as the manager ultimately computes an ROI analysis to inform the direction.

5.1.1. Task: Understand the Acquisition Landscape

In most DoD software-intensive acquisition programs and systems, the contractor delivers the software to the government at various levels of maturity. In most cases, testers will not have access to source code, automation testing accomplished at contractor facilities, or other artifacts typical of software automation early in development. Testing will be more performance-driven and vary across the acquisition lifecycle. One key responsibility for the manager is to try to obtain as much applicable automation work from the contractor as possible. There may be significant coordination with program management needed to have proprietary automation made available through negotiations and contract actions.

Figure 5.3 shows the usual DoD acquisition phases and milestones from 5000.02. In the Materiel Solution Analysis phase, there is little actual software development activities occurring, other than planning for automation opportunities. Managers should have awareness of the contractual negotiations and try to influence automation in contractor test activities. Most of the software development activities will occur between Milestone (MS) A and C. Automated test is generally applicable in Developmental Test (DT) and Operational Test (OT). In production and deployment, operational test continues as well as system improvement and sustainment activities, so the focus tends to be more on regression testing associated with software updates.

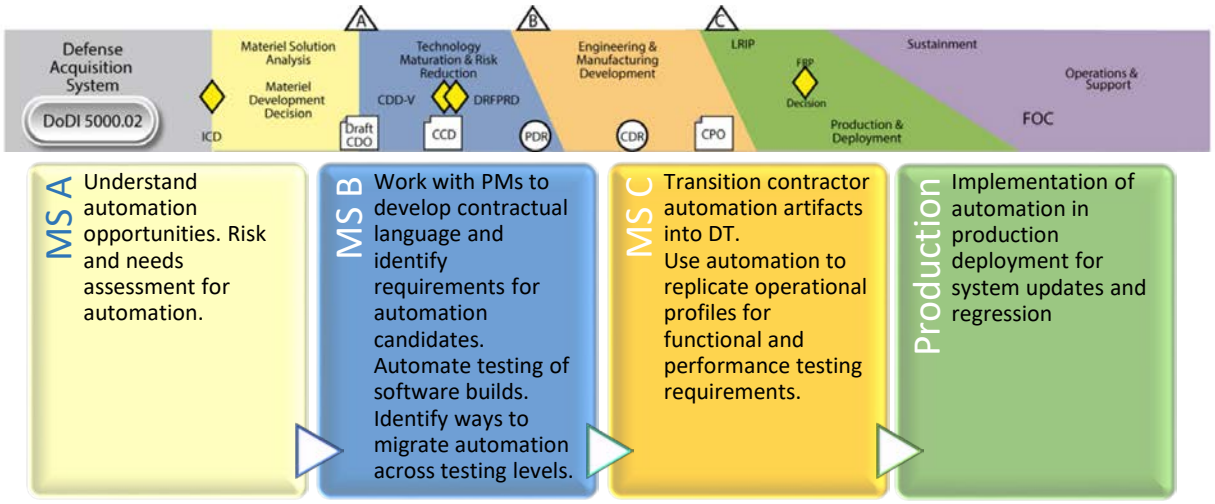

Figure 5.3. Notional Manger Automation Activities in the Access Phase across the Acquisition Lifecycle[3]

Automated software testing activities are equally important to the software intensive defense business systems whose requirements and acquisition policy is detailed in DoDI 5000.75[5]. Figure 5.4 shows the iterative Business Capability Acquisition Cycle for these business systems where there are Authorizations to Proceed between the phases that are the milestones. The manager will have similar duties for AST as in the 5000.02 analog shown in Figure 5.3.

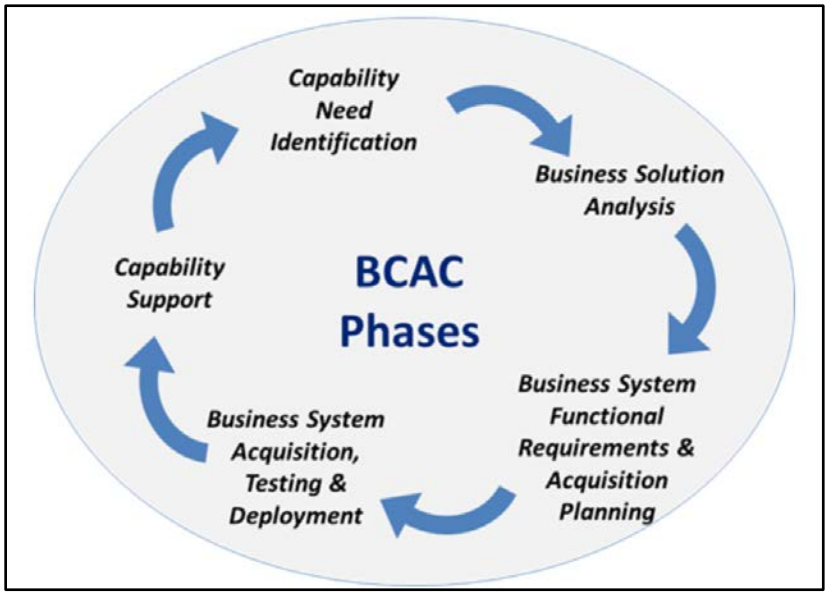

Figure 5.4. Business Capability Acquisition Cycle (BCAC) from DoDI 5000.75[5]

The first step to determining when and where in the program test process that test automation is beneficial is to take the goals, objectives, and requirements for the system under test and look for logical opportunities to apply automation. Michael Cohn's test automation pyramid, Figure 5.5 is a

[5] https://www.dau.mil/guidebooks/Shared%20Documents/DoDI%205000.75.pdf. Accessed 22-Oct-2018.

standard in the industry where the best success and ROI in automation is achieved at the lower levels of software development maturity such as unit testing. As you automate through the service levels (component, integration, Application Programmer Interface or API) and GUI there will be increasing costs, complexity, and likelihood of brittleness and failure.

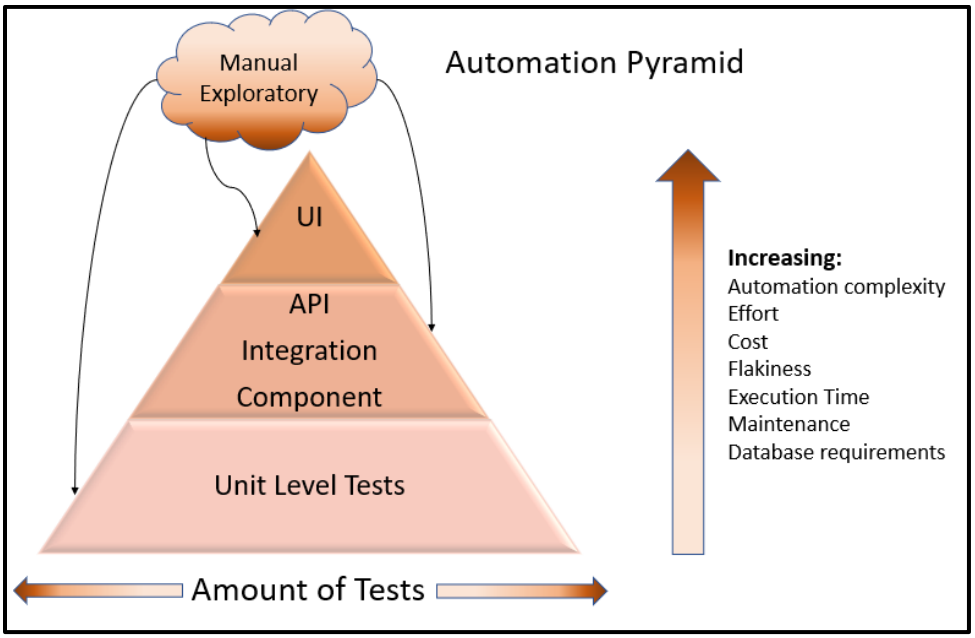

Figure 5.5. Automation Pyramid[6]

Managers should try to have the bulk of the automation efforts at the unit level, though in practice this may be difficult, particularly for the vast majority of DoD systems where the government is not directly developing the software. The hope is to execute these automated tests early and often. Care must be taken to not automate if it is only run a few times and is already efficient. Conversely, if the system undergoing software development has significant repetitive testing (e.g. regression), automation will help better ensure core functionality has not been impacted with the new updates.

Many systems will not have access to the software code so only black-box testing will be possible. Whether capable of white, gray, or black-box testing, not all software requirements are testable and of those that can be tested, not all should be automated. Realize that the program documentation (e.g., Capabilities Development Document or System Requirements Document) of system and design requirements/specifications are key inputs, but many other requirements exist and need to be tested based on the expected operational use and operational environment.

Management should recognize that feedback from fielded systems and the number of issues that come through help desk can be used to identify incomplete testing and serve as further justification for automation assistance.

[6] "There is not a good reference for the original idea." https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid. Accessed 3-Oct-2018

5.1.1.1. *Inputs:*

- Program documentation, SME input, and team advice

5.1.1.2. *Deliverables:*

- High-level screening of major lifecycle components as candidates for AST

Bottom Line: Before launching into a detailed discovery of the mechanics, resources required, and steps involved in test automation, consider your system maturity, current test environment, your plan and the schedule for testing. Also, be sure to identify opportunities in the acquisition lifecycle for automation, so that your automation research can be better aligned with your needs.

5.1.2. Task: Understand Automation

When considering opportunities for automation across the acquisition lifecycle, you should have test team members with some experience in AST or at least have reasonable access to individuals with these skills. Some useful resources for a background in AST include the STAT COE, organizations with AST expertise such as the SPAWAR Rapid Integration and Test Environment (RITE), commercial vendors, LinkedIn forums, industry experts, and textbooks such as *Implementing Automated Software Testing* (Dustin et al.), *Experiences of Test Automation* (Graham and Fewster), *Introduction to Software Testing* (Ammann & Offutt), *Foundations of Software Testing* (Mathur), and *Software Testing* (Hambling). Know there is a difference between automation and testing, and that automation supports testing (Figure 5.6).

The goal as a manager is to understand AST at a high level, determine what types of testing can be successfully automated, and generally recognize the value of applying automation.

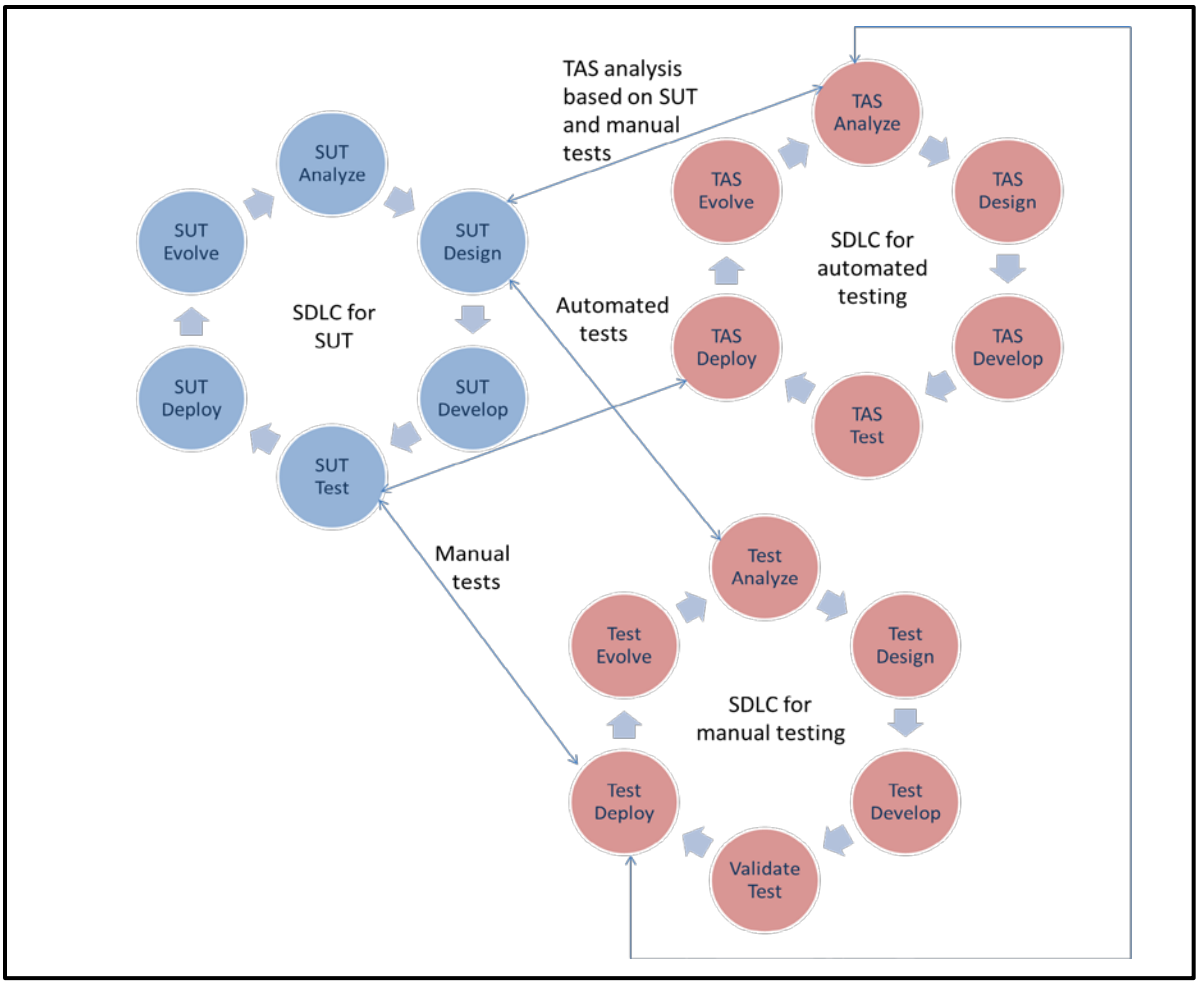

Figure 5.6. Interrelationship between Lifecycles of Software, manual, And Automated Testing[7]

An important aspect of the research is investigating what automation efforts have occurred (or purposefully have not occurred) on relevant systems, whether applied to previous versions of the system, or subsystems, or to similar systems. It is essential to investigate broadly and aggressively across organizations and across services. It is not uncommon for software development or acquisition teams (especially in larger programs and in the joint environment) to not have visibility into automation efforts in closely related programs. Where possible, try to leverage the previous and current automation work (tools and scripts) to quickly gain as much understanding as possible about the business decision to automate.

Managers should be informed on the adequacy and constraints of current manual test processes. Understand that automation is not just about converting manual scripts, but also about expanding

[7]Baker, Bryan, et al, "Certified Tester Advanced Level Syllabus, Test Automation Engineer," Version 2016, Figure 4, pg 42, International Software Qualifications Testing Board (ISQTB) <https://www.istqb.org/downloads/send/48-advanced-level-test-automation-engineer-documents/201-advanced-test-automation-engineer-syllabus-ga-2016.html> Accessed 3-Oct-2018

current test activities for increasing coverage and reducing deployment risk. Automation gains us efficiencies in execution, but we also want the testing process to be more effective.

Figure 5.7 provides the 30,000 foot view of the test automation process flow. The team develops test cases with the associated steps and data for the operational requirements that are judged feasible for automation. The engineers develop the automation scripts using a suite of tools that is itself a software development project that may have errors. These scripts will be executed many times and evolve with the SUT as well as automation maturity. Analysts evaluate output from the scripts to assemble reports on metrics such as defects discovered, execution time, and code or function coverage. A disciplined Configuration Management system is necessary to track changes over time as well as software defects.

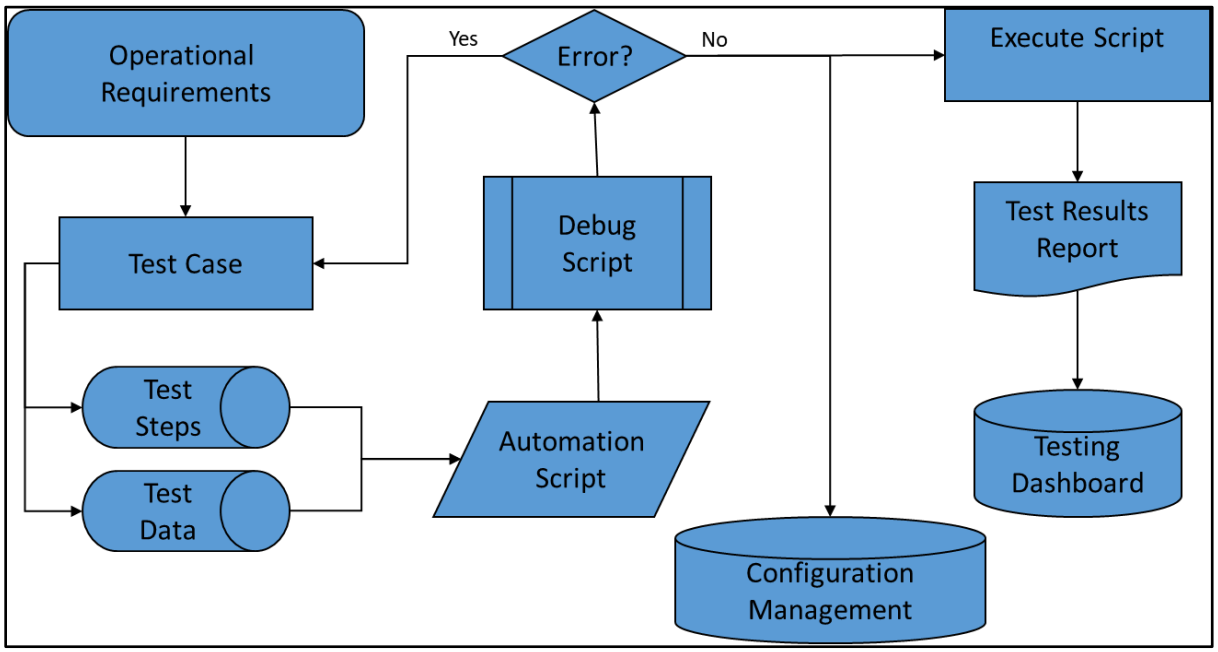

Figure 5.7. Components and Flow of an Automated Test Process

The system software development contractor may be conducting AST internally either as a standard practice or contractual requirement. Try to access and learn what they have done or are doing. It may require repeated inquiry, contractual language (i.e., Contract Data Requirement Lists or CDRLs), or a site visit involving demonstrations and documentation. Also, if early enough in the lifecycle, it may enable the government to require the contractor to automate software test and to deliver an automation capability.

5.1.2.1. *Inputs:*

- AST sites in DoD, textbooks, vendor resources, training/certification program, SMEs, documentation/dialog with other systems going through AST

5.1.2.2. *Deliverables:*

- Baseline understanding of the purpose and success factors in AST as described in ISQTB's Test Automation Engineer syllabus, Chapter 1[7]; and a whitepaper summarizing AST concepts relevant to particular system/environment.

Bottom line: Conduct research into automation opportunities for your program. Understand that AST can remove many existing constraints allowing more capability than otherwise envisioned. Find the right people that can perform a technical assessment, learn about ongoing automation for similar programs, and find out what the contractor is doing with automation.

5.1.3. Task: Identify Organizational Fit

Automation efforts have a better chance of success when there is not only sufficient capability, but also support across the organization. Automation in testing is not the typical mindset of most DoD organizations. One essential role across all organizations: government; contractor; or commercial, is that of leadership championing AST and actively managing the process. Organizational culture may be an important and seemingly insurmountable obstacle. If manual testing is the "way we have always done it," then a combination of leadership, policy, and technical skill insertion is needed to move forward. AST is the cornerstone as programs transition to test-driven development (TDD) initiatives such as Agile, DevOps, and Continuous Integration. These programs also require culture change as software developers may not be interested in overhauling test methodologies, operators may desire stability, and testers are not necessarily focusing on risk reduction.

There should be a general understanding of the AST resources required within the current test environment. Examples include personnel (testers and automators), software/tools, host computers, network environment, cloud support, information assurance/cyber protection, software approval processes, enterprise licensing, and so forth. Critical to success is to plan for and make feasible the overall schedule, and provide for flexibility because it may not be possible to automate within the existing timelines.

There often is a distinct difference in skillsets and experience between software testers and automators. With today's automation tools, testers with little software development experience can effectively learn to automate some aspects of testing that is otherwise done manually. The "shiny object" of record/playback automated testing is often the result of vendor marketing aimed at impressing busy executives rather than delivering real automation capability. A robust, streamlined, maintainable, and reusable automation solution often requires significant investment in software coding and development to grow fully independent and reusable automation test scripts that take full advantage of automation capabilities. Rarely will a single automation tool with a friendly graphical user interface (GUI) be the sole solution for all the automation needs. A suite of tools, each for a different purpose (e.g., browser apps, mobile apps, tracking, unit testing, and continuous testing) with different capabilities integrated into an automation framework, is often the recommended solution. Many tool vendors market their products as not requiring any software development experience, but they often can automate only specific types of tests and have limitations on maintainability and reusability.

The manager must assess if the team has the right skills needed for automation. The current skill set could drive a need for training and/or specialized certifications in addition to increased experience levels. New automation projects should use individuals with experience in automation and not just hope that manual testers will figure out a way to make automated tools work.

The manager has a large role to play early in the AST journey. It may be helpful for the manager to canvass the organization with personal interviews and/or surveys to collect necessary information to understand the current and potential future AST landscape. The manager will need to establish

credibility, develop a specific action plan and continue to lead the initially ill-defined automation effort. Recognize that automating solely to satisfy a mandate or to be part of a burgeoning community without sufficient infrastructure will not lead to success.

5.1.3.1. *Inputs:*

- Identify current resources and capabilities in personnel and tools through walk-arounds, interviews, and surveys

5.1.3.2. *Deliverables:*

- High-level summary of needs, resources and gaps

Bottom line: Pave the way to automation success by securing leadership support, identifying the major pieces that must be in place to automate and then comparing the needed resources to your program's current state. Let manual testers know that they too will contribute to the overall success of automation as it is a team effort. Gain a sense for the hurdles to overcome in order to build an automation capability.

5.1.4. Task: Go/No Go Return on Investment Analysis

The decision to automate must include several factors that may carry different emphases or weights. A frequently used approach to supporting the automation decision is to quantify the costs and benefits in the form of an ROI analysis.

Benefits: There are many benefits to AST to be considered and estimated up front. Some of these include:

- improved software quality through deeper, more thorough test coverage
- fewer defects sent to next software test phase
- automation of tedious processes allows test staff to focus on challenging exploration
- testing can be done unattended (overnight)
- better coverage of the operational use space or software capability conditions
- better repeatability and reproducibility of test results; lower error rate from tedious tasks
- quicker software development cycle/sprint times

Costs: There are both direct and indirect costs associated with an automation project. Representative direct costs include:

- software licensing and training
- hardware and middleware components for the automated test framework
- cloud and network services
- labor investment for learning new tools, integrating components, executing/analyzing tests
- contractor consulting costs

Indirect costs can be thought of as the hidden or unexpected time required to automate. Some examples are:

- time taken away from the manual testing function
- time to collaborate across many test functions

- extra time to keep management informed
- time to achieve and maintain competence in using the suite of tools
- delayed capability delivery costs from schedule slips
- time to maintain the automated solution
- maintenance costs required due to frequently changing system configuration and images

Costs and benefits should also be viewed through a long-term lens. The direct sunk cost of licensing, tools, training, etc., should be amortized over the expected duration of the automated software test program. For example, Figure 5.8 shows a notional project where the manual cost of test execution averaged $150k in the base 6-month period. In the following 6 months manual test execution continued while an automated test solution was implemented. Combined, the total cost was close to $250k. However, within that 6 month period, some of the manual tests were automated as indicated by the reduced cost of manual testing. At the 12 month mark no additional implementation costs were necessary and the process of converting manual to automated tests continued. Because of automation the combined manual and automated test execution costs were reduced (<$100k) as compared to the full manual costs ($150k) from the base period. At 18 months, a significant portion of the manual tests were automated, further reducing the total test execution cost (<25k). At 24 months only a few remaining tests which cannot be automated remain with a consistently low test execution cost. Additional automated tests at 24 months were added for deeper testing.

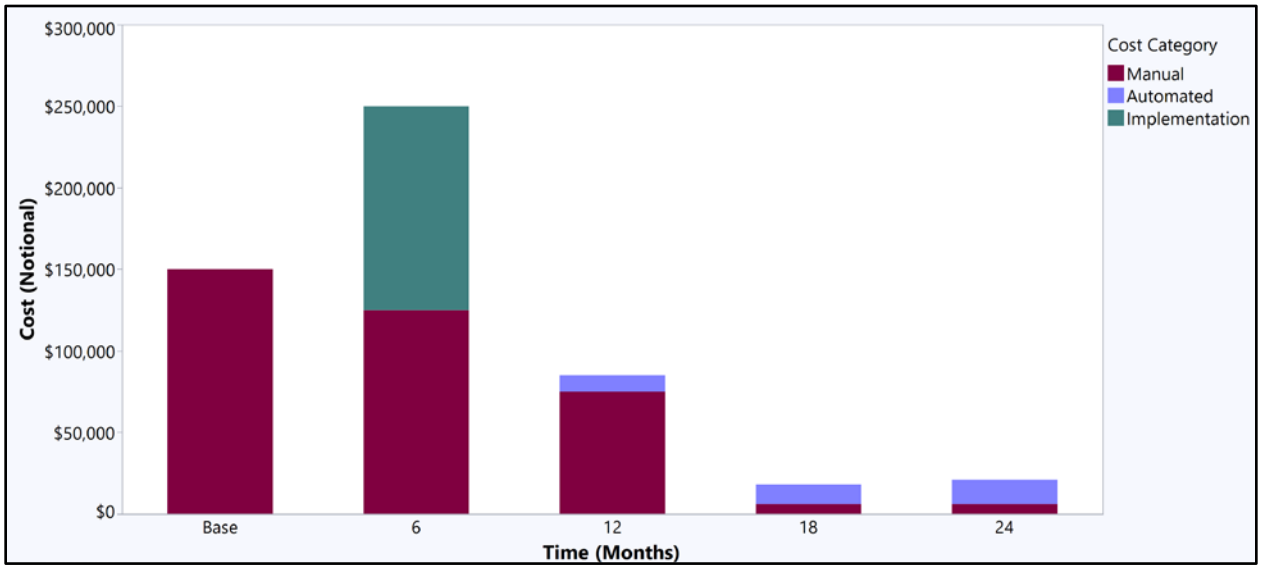

Figure 5.8. Automation and Manual Testing Execution Costs over Time

It should be recognized that all requirements should not necessarily be automated. If there are some tests that only occur one time or are on stable software builds and environments, then typically the marginal benefits would not outweigh the costs. Take time to thoughtfully consider both the quantitative and the qualitative costs and benefits of the automation effort.

Benefits should also be viewed for their long-term impact. Managers should recognize that the initial backlog of manual tests will be converted to automation over time. The effort can begin the development of additional tests for enhanced coverage and deployment risk reduction. The test

execution costs, previously depicted in Figure 5.8, can also be viewed in the number of scripts and test cases. Figure 5.9 shows the transition from manual to automated testing resulting in an increase in the number of test scripts over time. All candidate manual tests have been converted to automation, except for a few remaining manual tests. This resulted in additional test coverage that automation makes possible.

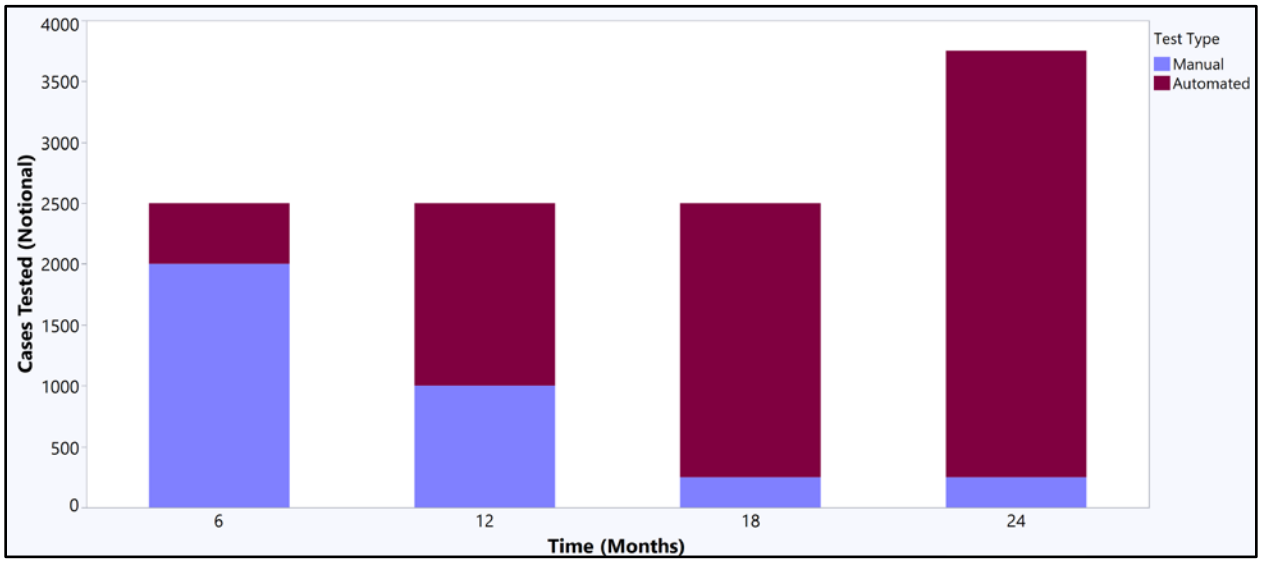

Figure 5.9. Manual Tests Migrated to Automation Eventually Increasing in Quantity and Coverage

It is helpful to estimate the ratio of the expected additional time to develop an automated routine to the time it takes to run a manual test. This ratio provides an indication of how many tests it will take for the automation investment to pay for itself in terms of time, not necessarily dollars. The manager can then use this ratio to see if the automation investment will be worthwhile based on the expected additional testing conducted over the lifecycle. There is no set guidance on the actual value of this ratio, rather it is an indicator of the expected duration of time for to realize a savings. As mentioned in the benefits and costs sections, many of the actual benefits and costs are not considered up front. It would be prudent to compute this ratio for a range of expected costs and benefits (i.e., optimistic, realistic, pessimistic) to see the sensitivity of the results to the estimated benefits and costs.

To combat the usual array of untested assumptions, unrealistic expectations, and program pressures, leadership needs to be provided with objective analyses to assess risk. Ideally, leadership wants to make a 'Go/No Go' decision as early as possible; however, not before they have been provided a solid business case from the staff. Typical leadership roles participating in these decisions include the Integrated Product Team Test Lead, Director of Engineering, Program Manager, and Software Development leadership.

If an existing AST program fails to provide sufficient ROI, carefully consider whether it makes sense to execute the project as a foundation for a follow-on automation effort that would provide much greater capability. Because many costs have already been incurred and a myriad of lessons learned, it is likely that the next AST program will provide a substantial ROI.

Now is the time for leadership to make the decision to automate or not. Be sure to convert the metrics from automation into quantifiable gains (time and money) to the program, and then have your operations analyst compute the automation ROI for both the short and the long range as appropriate for the program. Use the ROI to help leadership make an informed decision and be sure to document all analyses along with minutes of decision briefings.

5.1.4.1. *Inputs:*

- Cost and benefit estimates along with an understanding of the organization culture

5.1.4.2. *Deliverables:*

- 'Go/No Go' decision briefing focused on ROI

Bottom line: Estimate the rough order of magnitude for direct and indirect costs (financial and time) and benefits for the near, mid, and long-term. Refine estimates with SME input and use this knowledge as your team creates the 'Go/No Go' decision briefing.

5.2. Plan Phase

Manager Checklist for:

Plan Phase

- ❑ **Request Project Support**
  - ❑ Review the ROI analysis performed in the Assess phase and update as necessary
  - ❑ Check on the status of the funding request, and, if necessary formally request funding
  - ❑ Develop a strategic automation plan and socialize with senior leadership
- ❑ **Determine Requirements to Automate**
  - ❑ Research and decide on the expected functionality of the automation
  - ❑ Identify requirements that are less challenging with high probability of automating success
  - ❑ Determine and document test automation objectives
  - ❑ Ensure requirements trace to system capability and function
  - ❑ Develop process flow charts to decompose the process and identify automation needs (i.e. input data generation, test execution, output data collection and collation, etc.)
  - ❑ Consider the Developmental Evaluation Framework to help prioritize and ensure automation is appropriate
- ❑ **Identify and Acquire Resources**
  - ❑ Determine personnel options and recommend best option
  - ❑ Initiate and participate in the hiring process to include procuring technical advisory contractors
  - ❑ Decide on equipment (computers, cloud, network…) and facilities (lab, work areas, meeting rooms…)
  - ❑ Decide on automation software tools (collaboration, automation, analysis, requirements, open source, commercial…)
  - ❑ Determine best tool acquisition strategy (individual, network, or enterprise licenses) and procure tool licenses
- ❑ **Define Roles**
  - ❑ Determine the relevant automation roles for the initial project and possibly the next project. At least consider test automation architect and engineer, database expert, analyst, tools SME, tools system administrator, configuration manager
  - ❑ Decide on automation roles and loads for each team automation specialist
  - ❑ Develop an automation specialist management plan and socialize for improvement
- ❑ **Chart a Path to Success**
  - ❑ Lead the culture change and empower team to learn automation principles and explore tools
  - ❑ Create a well-defined and documented test automation Plan of Action and Milestones
  - ❑ Determine qualitative measures of success in accomplishing the culture change
  - ❑ Determine metrics to track and assess for project automation success
  - ❑ Develop and decide on metrics for assessing how automation increases test coverage and testing efficiencies
  - ❑ Refine metrics based on short and long-term impacts

Figure 5.10. Manager Checklist for Plan Phase

The Plan phase, Figure 5.10, flows from the 'Go/No Go' decision and sets forth the commitment of resources to undertake automation. Management has to support the team by planning for and delivering resources such as staff, tools, and hardware to get the project moving. Managers have to decide what requirements should be automated sensitive to the need to demonstrate success early in order to build momentum. The core capability continues to grow with incremental improvements over time. All the planning should be documented in an automation test plan.

5.2.1. Task: Request Project Support

Given a 'Go' decision, review the details of the ROI analysis as some things may have changed or leadership may have given additional direction. During the 'Go/No Go' decision process, the requested funding should have been part of the decision brief during the senior leadership meeting. If not, the manager must now make the case for automation up the chain seeking appropriate resourcing. Fortunately, the ROI analysis in the previous task translates well to convince budget authorities of the value of an automation solution. It is helpful to have other programs' success stories and ROIs available to further the case. Ideally, there is strategic direction in the acquisition community highly encouraging the use of automation where practical. A good, detailed plan will go a long way toward convincing decision-makers to invest in the project.

Realize there will likely be significant lead time involved before funding arrives in order to execute many of the tasks. The current team can begin researching automation principals and tools while awaiting additional resources. There are many open source tools worth trying, investigating and running examples. This is further complicated with new hiring; or, if outside government support contractors are anticipated.

5.2.1.1. *Inputs:*

- Updated resource estimates from the Assess Phase

5.2.1.2. *Deliverables:*

- Decision brief and leadership commitment to fund the project

Bottom Line: You will need to champion your Automated Software Test program cause to program budget authorities with updated estimates from the 'Go/No Go' to ensure you have sufficient funding to proceed.

5.2.2. Task: Determine Requirements to Automate

Requirements are statements of expected functionality. Software requirements have already been addressed with the team at a high level focusing on those offering the best opportunity for successful automation. Plan to start small and continue to grow in automation capability. Do not underestimate the power of success—go for those requirements first that are less challenging but nevertheless impactful. The goal is to think beyond traditional requirements and the traditional depth/breadth of testing as automation can enable more rigorous and more complete testing of the system under test.

Careful test planning always starts by forming concise yet comprehensive objectives, and for most phases of developmental testing, objectives come from the requirements. Software testing

requirements will have to be distilled from various sources. Rarely will there be a comprehensive and well thought-out list of requirements with testable criteria. These requirements must evolve through an iterative and collaborative approach where stakeholders can agree to a common set of requirements as well as a prioritization scheme.

First, those requirements that are traceable to actual system capability and functions should be identified. Source documents for these kind of requirements include Capabilities Development Documents, Capabilities Production Documents, Concept of Operations, Operational Mode Summary/Mission Profiles, system specifications, system software specification, and component specifications. Other source documents include System Engineering Plans, Test and Evaluation Master Plans (TEMPs), test plans/detailed test procedures of similar systems, and contractor design documents and test plans. A meticulous process decomposition using flow charts and activity diagrams will identify many sub-requirements that trace to a system function. MITRE's Development Evaluation Framework, discussed in Appendix D: Considerations for Automating Test Requirements and Test Cases, can be helpful in breaking down the system objectives. Depending on the phase of test (e.g., unit, integration, functional, performance in DT, integrated DT/OT, OT), the testing scope (level of detail and number of requirements) will vary. Requirements not directly associated with capabilities and functions also need to be addressed in the Planning phase. Next, non-functional requirements such as those associated with the expected operating system, information assurance features, hardware systems, and other environmental factors need to be identified. The requirements for the AST framework should start taking shape and will be addressed in subsequent sections. It is helpful to have a requirements management (RM) system in place enabled by tools such as Atlassian JIRA, IBM Rational DOORS, qTest, and XQual to help plan and track automation requirements. The process described in this paragraph provides the information for creating a Requirements Traceability Matrix (RTM).

After completing the requirements definition, the development evaluation framework, tool research, and consultation with test automation experts, it should be clear which tests are the most promising to automate. Decisions regarding what can be automated and should be automated will be influenced by the software development life cycle (SDLC) to include (unit test/development, integration, functional, and performance) and test phase (DT, integrated DT/OT, and OT). Factor in ease of automation, success of automating similar requirements across other programs, the team's expected capability level, the potential improvement in test coverage, and how often a test will need to be run manually. Balance these considerations against the risks associated with upfront fixed automation investment costs, penalties for scheduling delays due specifically to automation challenges, additional workforce development, and initial time investment taken away from manual testing. Recognize that correctly choosing not to automate some of the requirements may be some of the best decisions you make. Finally, do not assume *a priori* that a direct translation of a manual test is the best way to automate.

Figure 5.11 shows that out of the "Sustainable Manual" tests a large percentage is "Presumed Automatable" except for a small percentage of manual-only tests. However, most organizations define what needs to be tested based on schedules rather than on the totality of testing for proper coverage. The "Test Universe" represents what is possible with a well executed test strategy with multiple conditions. Of this, a large "Potential to Automate" exists. Most organizations are not thinking about the "Test Universe" because without automation it is unachievable.

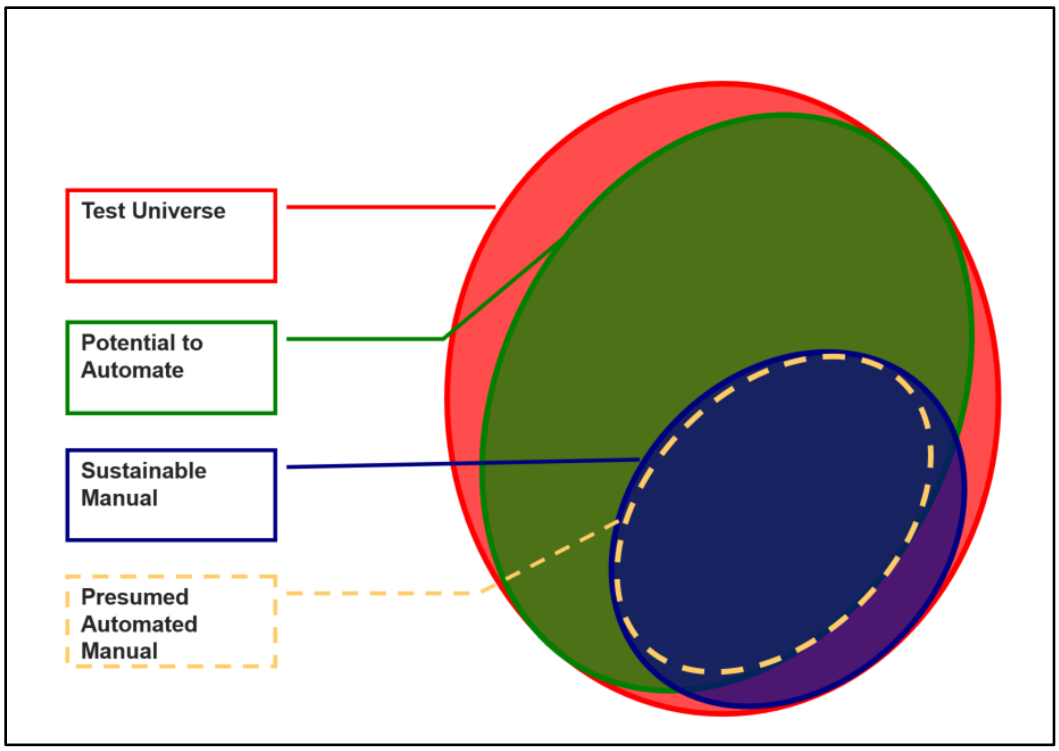

Figure 5.11. Notional Test Coverage with Manual and Automated Testing

It is helpful to bin automation opportunities into initial categories of easy, moderate, difficult, and very difficult, and to sequentially automate in this fashion. This ordering will allow the team to gradually sharpen their skills while achieving success along the way. Multiple automators will benefit from each other's learning, which accelerates automation capabilities.

The requirements that benefit most from automating are those associated with:

- Repetitive tests running on multiple builds
- Tests that tend to be "becaused" or influenced by human error
- Tests that require multiple datasets and platforms
- Frequently used functionality that introduces high risk operationally
- Time-intensive and tedious manual tests

Automating regression tests and smoke tests that are tedious, error-prone, frequent, and provide significant benefits.

Tests that are less likely to benefit from automating are:

- Newly designed tests that have not been manually executed
- Tests with frequently changing requirements
- Tests that are highly unlikely to be executed in the operational environment

5.2.2.1. *Inputs:*

- Test requirements derived from source documents

5.2.2.2. *Deliverables:*

- Requirements test matrix ranked by automation priority

Bottom line: The places to start in detailed automation planning are the system requirements and intended operational capabilities. Collect all the available system requirements documents and develop a comprehensive and automation-prioritized requirements matrix.

5.2.3. Task: Identify and Acquire Resources

Managers are the cornerstone for sourcing the resources. The primary resources are people, equipment, and tools. They must understand that resources are limited and often shared (at a much lower percentage than typically "agreed" upon). Furthermore, the resources are not necessarily required full time for a lengthy duration—you may only need a capability for a week.

For people, consider all your options for manpower needs. Hiring needed talent can be a solution, although in the DoD that process can take more time than you can afford. What many organizations have found to be successful has been to either train and mentor technical staff within the organization, or find resources locally based on your needs and the nearby talent availability. The long-term benefit to the software acquisition program depends on how much automation is value added, given the constraints of the system requirements that are testable, the timelines, and expected future automation efforts. An alternative may be to contract out the work to an experienced AST group, whether government, contractor, or commercial. A detailed discussion of specific knowledge, skills, and abilities will be required and are further explored in the next section on team roles.

Many organizations will need to hire contractors to help in automating tests. They will be embedded with the team on-site. As the manager, you likely will be expected to be the technical representative to the contracting officer responsible for writing the performance work statement, evaluating proposals, and advising on source selection. This set of tasks alone can be very time consuming.

The number and types of tools available for AST can appear overwhelming initially. Each phase of an automated test program potentially has a requirement for a different tool or set of tools that may need to be integrated with a disparate collection of tools for other purposes. The goal in the Planning Phase is to understand the general capabilities of the relevant tools and then down-select to a few promising candidates that will fit your workforce, timing, and desired level of automation rigor. A thorough assessment of needs will typically result in a multiple-tool solution.

Begin with the tools your team uses or feels comfortable with and can efficiently use for the upcoming AST. These tools will have higher priority due to the associated shorter learning curve and better overall automation probability of success. Write up a brief gap analysis of these tools to find the capability holes in order to identify alternative tool solutions. Perform a broad search for possible tools such as those used historically, open source alternatives to include tools built specifically for the DoD use, freeware, and commercial packages. Take time to interview experienced automators to find out what they are using and why, what they have tried but no longer use, and what they would like to use if their program was unconstrained.

Some measures to keep in mind as you consider possible tool solutions are:

- Does the tool support the types of tests you'll be conducting (unit, regression, functional, test management, mobile, agile/dev ops, etc.)?
- Are the types of tests modular and capable of being shared across application domains?
- Does the tool use a common scripting language like VB, JavaScript, Python, etc.?
- Where in the testing or acquisition lifecycle can it automate?
- What operating systems does it support? Is it for web-based systems only?
- What is the ease of use and how big is the learning curve to effectively automate?
- Can the team be reasonably expected to efficiently use the tool? Is it easy to debug scripts?
- Does it have a GUI capture-replay capability only or does it support application programmer interface (API) calls to the GUI?
- What are the total costs to include licensing, training, maintenance, and support? Are enterprise licenses available elsewhere that can be used?
- How responsive is the vendor for support questions and troubleshooting? What training or user community is available free on-line?
- What are the information assurance hurdles? Can it be integrated into and operated on government computers (e.g., Navy Marine Corps Internet (NMCI)) and if so, how long is the approval process?
- What are the output products and can they be easily accessed and customized? How well does it provide insight for debugging (or fault identification) versus just showing that a test has failed?
- Is there a history of Verification, Validation, and Accreditation (VV&A) within the DoD?

A tools assessment is one of the most time-intensive yet rewarding parts of the planning process. Often, test automation projects have selected a tool based on very limited information (e.g., only one they know of) and later regret their decision (pay now or pay later syndrome). Better automation tools typically require automators to have the time and motivation to learn the tool, resulting in a more stable (less maintenance), and more extensive automation capability. Consider all reasonable options and understand the most popular tool may not be the best choice for your team.

Automating software testing is itself a software development project. There are many costs beyond the tools to consider as well. These additional expenses may include:

- Facility costs for a software integration lab
- Computer hardware and peripheral costs
- Network and storage costs
- Overhead costs for the AST team

5.2.3.1. *Inputs:*

- Expected resource requirements: staff, tools, and environment

5.2.3.2. *Deliverables:*

- Budget of resources required for automation

Bottom line: Automating software tests is resource intensive. Finding or training automators and keeping them engaged and motivated is the single most important investment by any group interested in successful automation. This aspect of the automation process also tends to take the most time and

can be expensive depending on the route chosen. Do not forget the costs (to include labor) of tools and computer resources.

5.2.4. Task: Define Roles

There often is a distinct difference in skillsets and experience between software testers and automators. With today's tools, testers with little software development experience can effectively learn to automate some aspects of testing otherwise done manually. However, a robust, streamlined, maintainable, and reusable automation solution often requires significant investment in software coding and development to grow a fully independent and reusable test automation framework that takes full advantage of automation capabilities. Rarely will a single automation tool with a friendly graphical user interface (GUI) be the sole solution for all the automation needs. A suite of tools, each for a different purpose (e.g., browser apps, mobile apps, tracking, unit testing, and continuous testing) with different capabilities integrated into a robust automation framework, is often the recommended solution. Many tool vendors market their products as not requiring any software development experience, but they succeed in many ways at automating only specific types of tests and have limitations on maintainability and reusability.

There are several key possible roles managers must consider and include. It is likely one individual will fulfill more than one role and, depending on the scale of the project, there may be several members of your team in the same role.

System/domain subject matter expert: This skill is essential to understanding the SUT and how to effectively integrate automation in testing. This type of staff person is likely organically available and already participating at some level in system test. The key is to find or educate someone that understands the capability of an automation solution.

Test automation architect: The overall automator in-charge who has the technical understanding of automation tools and who will create the vision for designing a purpose-built architectural solution to support automation of requirements-derived tests.

Test automation engineer: These team members are the ones building and maintaining components of the Test Automation Solution (TAS). This is more like a software development process activity than manually testing.

Database administrator: Many tests require access to data products from disparate sources. A knowledgeable team member who can effectively connect the sources is often required. Database skills will also be necessary to instantiate databases in a test environment (possibly requiring obfuscation of personally identifiable information (PII)) and to manage the output data generated from the TAS.

Operations analyst: With the automated solution comes a considerable quantity of output data that characterizes system performance. Many software engineers and architects do not have advanced skills in analytical methods or the time to efficiently discover software defects, trends, root causes and other metrics associated with software performance. An analyst may be required to keep up with the large quantities of data routinely generated as artifacts of the process. The analyst will also have insight into scientific test and analysis techniques (STAT) that may need to be incorporated into the solutions such as combinatorial factor array designs to maximize defect discovery opportunities.

Tools SME: Many AST tools with excellent capability also have a steep learning curve. The tools SME will be the expert to effectively integrate all of the tools into the framework. This task may require building customized programming interfaces that transfer data from one system to another and aggregating data for reporting purposes.

Tools system administrator: Like all software applications, tools will need licensing and administration rights. DoD information assurance standards can be especially challenging to get software approved and operational in a test program.

Configuration manager: Automating software test is itself a software development project that requires careful managing of the configuration of SUT versions, the scripts, and the output generated. Many scripts may be rendered useless as the SUT or components within the framework change necessitating an internal configuration management process.

Though every member on the team will be involved at some level across the automation lifecycle, Table 5.1 highlights the phases where the skillsets of each role will be in highest demand.

Table 5.1. Summary of Role Support across the Automation Lifecycle

Role	Assess	Plan	Design	Test	Analyze & Report	Maintain & Improve
System SME	**X**	**X**				
Automation Architect			**X**			**X**
Automation Engineer			**X**	**X**	**X**	**X**
Database Admin				**X**	**X**	**X**
Ops Analyst		**X**	**X**	**X**	**X**	**X**
Tools SME		**X**	**X**	**X**		
Tools Admin			**X**	**X**		
Configuration Mgr				**X**	**X**	**X**

There will be significant lead time required to hire and train personnel to achieve initial automation capability. The time and resources required will be a function of automation goals. Management must decide whether to grow the current workforce internally or hire out the positions. They must identify where the talent resides, the best source (military, civil service, or contractor) and how to attract the right people taking into account the delays associated with the hiring process. If the decision is to build an AST capability by training the current workforce, it may be difficult to find the individuals with the right potential and motivation who can be freed up from their current duties enough to train and be successful. If this approach is taken, a deliberate skill growth plan that identifies training needs, appropriate peers, mentors, coaches, and training timelines is crucial to success. The long-term benefit of growing the automation capability internally could be substantial. Consider too that this option may not be the best investment depending on how much automation is value added, given the constraints of the system requirements that are testable, the timelines, and expected future efforts. An alternative may be to contract out the work to an experienced AST group, whether government, contractor, or commercial.

5.2.4.1. *Inputs:*

- Team skillsets and role definitions

5.2.4.2. *Deliverables:*

- Staffing and training plan

Bottom line: Build your team for success! Realize that simplistic strategies, such as forcing current manual testers to be automators, analysts, or developers, are not likely to lead to long-term success.

5.2.5. Task: Chart a Path to Success

The manager's primary function is to lead the project team to attain its customization goals. The critical component is a well-designed plan that realistically balances resources with automation goals. A Plan of Action and Milestones (POA&M) with high level tasks and schedules is essential to successfully execute the proposed automation program.

The manager will also want to track success factors that are not technical per se, but more along the lines of the managing a potentially substantial paradigm shift in the test program. Factors to manage may include the:

- Team's acceptance/buy in of the expected automation
- Team's transitioning to automation with new responsibilities, tools, and methods
- Team's productivity fluctuations as automation progressively stabilizes
- Team's changing requirements for reporting methods and accountability
- Collaboration by all team members to contribute towards successful automation

The manager must also focus on the quantifiable metrics defining success. Some metrics the manager may consider for achieving automation success are the:

- Percentage of members on board with automation
- Number of tools evaluated
- Initial automation capability achieved
- Sustained automation capability
- Enhanced automation capability
- Training accomplished in automation
- Percentage of test requirements deemed viable for automation
- Ability of test force to focus energies on high priority/high risk areas

There are also metrics for the actual automation solution that need to be tracked. The primary goal of automated software testing is to deliver better software quicker. We need to discover defects and opportunities for improved performance more quickly and more thoroughly than otherwise would have been achieved using a manual approach. Some metrics to consider when comparing fully manual versus some degree of automation are:

- Increased coverage for lines of code tested
- Increased coverage of expected operational paths and use cases
- Percentage of automatable requirements currently automated
- Manpower savings over manual testing, especially for repetitive testing (e.g. regression)
- Ability to scale with multiple users and environments; perform high-load and boundary testing
- Better output data for analysis and reporting

- Higher defect discovery rate through better coverage and/or freeing manual testing resources ,for deeper exploratory testing
- Higher quality of delivered software
- Shorter time to field system
- Number of tests executed during continuous testing to include overnight and weekends
- Reusability of automated scripts

Consider the short term and long-term impacts of the metrics and how to effectively communicate these values to both the test team and leadership. Regularly scheduled automation reviews with metric reports are helpful to show progress and identify improvement opportunities. There will likely be several measures needed to fully capture the overall automation quality. Consider tracking these metrics with dashboard type designs visible to the team.

5.2.5.1. *Inputs:*

- Evaluation of current automation capability from multiple input sources.

5.2.5.2. *Deliverables:*

- POA&M with a scorecard tracking relevant metrics quantifying team's acceptance, automation progress, along with software testing efficiency and effectiveness.

Bottom Line: Automating software test will not happen without strong leadership continuously monitoring appropriate measures describing the team's acceptance, capability, and actual progress. The manager must effectively determine appropriate steps to keep progress on track.

5.3. Design Phase

Manager Checklist for:

Design Phase

- ❑ **Develop the Automation Test Plan**
  - ❑ Outline major sections
  - ❑ Assign team members responsibility for appropriate sections
  - ❑ Conduct periodic reviews with team and stakeholders (as required)
  - ❑ Brief plan to senior leadership emphasizing required areas needed for support
- ❑ **Procure Resources**
  - ❑ Ensure qualified personnel are in place on team
  - ❑ Select and acquire appropriate tools for automation vision
  - ❑ Procure appropriate resources for test automation architecture and framework
  - ❑ Establish a comprehensive data collection and analysis system
- ❑ **Manage Planning and Execution of Initial Pilot**
  - ❑ Determine limited scope objectives
  - ❑ Develop pilot test plan
  - ❑ Create automation scripts and compare to manual tests
  - ❑ Monitor test and team performance
  - ❑ Report results
  - ❑ Determine next steps to scale pilot to additional requirements
- ❑ **Conduct Design Review**
  - ❑ Develop list of items required for automation capability
  - ❑ Assess each item with simple ratings scale (red, yellow, green, blue)
  - ❑ Identify gaps and develop mitigation strategy
  - ❑ Update POA&M
  - ❑ Brief chain of command on readiness for automated software testing

Figure 5.12.Manager Checklist for Design Phase

The design phase, Figure 5.12, further advances the program by procuring resources that will allow the team to begin creating automation capability. The hallmark of the phase is the initial pilot test scripts where the team can see progress and the impact from their efforts. Management must remain flexible to change as the team learns through trial and error which methods and tools are most effective. The Design Review will lock down the tools, framework, personnel, data collection system, and other attributes based on team input and pilot results. Managers must also communicate results and proposed plans to upper management along with stakeholders to ensure continued support and funding.

Figure 5.13 is the typical Gartner Hype Cycle which applies well for implementing automated software test. The Design phase will be the most volatile phase where different members of the team may embrace the change at varying levels. After the initial excitement, feelings of doubt and frustration may prevail until there are small wins which increase optimism. Views of the automation solution finally level

off at acceptance characterized by seeing there is significant improvement over exclusively testing manually.

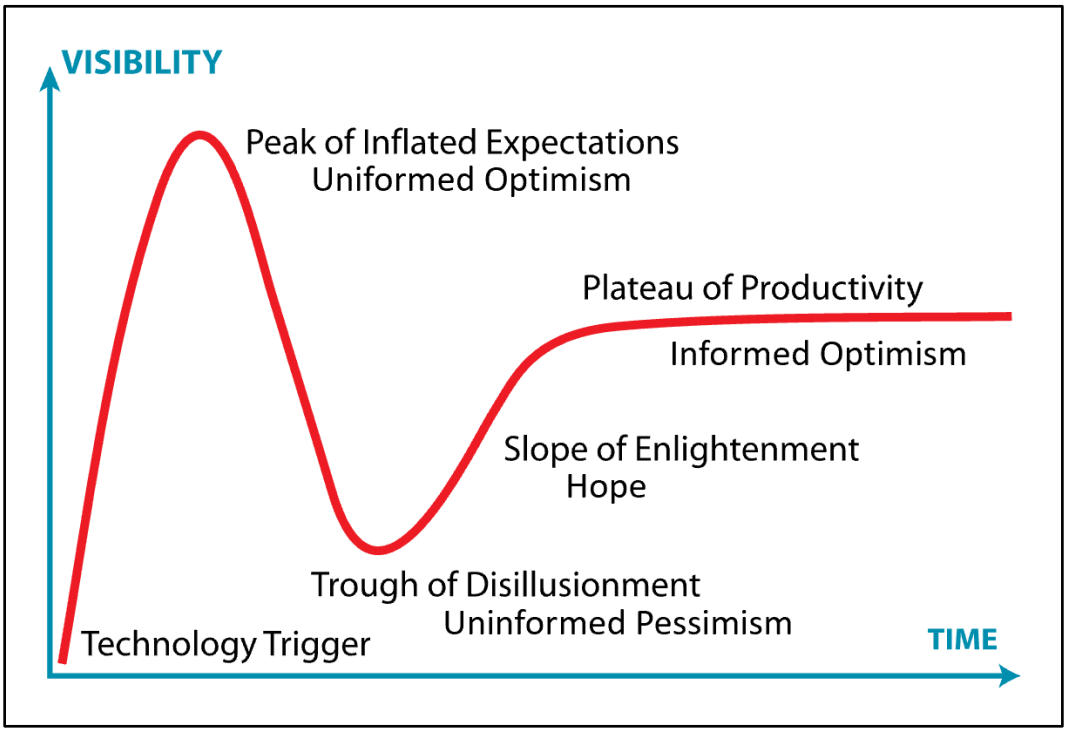

Figure 5.13. Gartner Hype Cycle for Automation[8]

5.3.1. Task: Develop the Automation Test Plan

Documentation is important to ensure everyone (management, test leads, software developers, system engineers, software engineers, automators, etc.) clearly understands the overall AST approach. The actual act of writing the test plan will spawn useful discussion for the team.

The AST plan is a living document and should clearly articulate the planned approach. Sections should include but not be limited to the following:

- Detailed system description
- Delineated requirements along with the DEF decomposition matrix down to automatable tasks
- Requirements prioritization methodology and ranking
- Planned AST architecture and framework design specification
- Candidate tools (and versions) along with capability evaluations
- Responsible Point of Contact (POCs) for critical tasks
- Test resources
- Process reporting requirements
- Data collection systems
- Anticipated analysis
- Timelines

[8] https://commons.wikimedia.org/wiki/File:Gartner_Hype_Cycle.svg. Accessed 3-Oct-2018

The manager should assign responsibility of each section to team members. Periodic meetings will be required to communicate the AST plan and to ensure development of automation capability continues on schedule. It may be helpful to have developers, product owners, SMEs, and other stakeholders attend some of the sessions to foster a more collaborative environment once testing begins. The automation test plan will serve as the source document for manager briefings to leadership on the automated software test program approach and for test automation development and execution.

5.3.1.1. *Inputs:*

- Original POA&M

5.3.1.2. *Deliverables:*

- Project POA&M and the Automated Software Test Plan

Bottom line: The Automated Software Test Plan is the living document leadership uses to manage and execute the test program. Determine the frequency of testing based on the software development cycle and testing needs.

5.3.2. Task: Procure Resources

The manager is very familiar with the resources required to successfully implement an automated software test program. The Design phase (referencing Figure 5.12) focuses on acquiring the required resources and putting them in place.

Personnel. An AST-enabled test team has software engineers, test engineers, software testers, and automated software testers. A high-performance team is crafted either by training and developing current testers/engineers into automators or by hiring personnel (full or part-time) with those skills, or by some combination of developing and hiring. The decision on how to achieve automation capability is a function primarily of the automation requirements, resources available (time and funding profile), and the in-place team's skillset as well as potential for growth. To grow an in-house automation capability, the manager has to ensure the team has the time and resources for all of the activities involved with formal training, self-teaching, networking, and mentoring. The manager may have to conduct significant coordination and negotiating activities to acquire instant expertise by "borrowing" neighboring automators. Contracting consultants can be a drain on the manager's time and budget and may require extensive lead times prior to the start of the Period of Performance. Hiring government full-time equivalents, whether military or civilian, would also require long lead times. The team needs to be learning and automating smaller, achievable tasks while the expert help is on-boarding.

Software Automation Tools. As the requirements and test environment have become more mature, the preferred toolset should become more obvious. Take the outputs of the tool decision process from the planning phase and acquire the tools best suited to your team's success. Be sure to select the automated test tools appropriate for the requirements and SUT. There will be additional research required to make the best tool selection that efficiently maximizes both capability and usability. Sources include guidebooks, texts, related studies, other DoD AST tools efforts, vendor sites, and demonstrations. It is likely that tools not initially considered will now enter the mix.

Know that there will likely not be a single tool the team will use for all functions. Some examples are:

- Tools in writing out automation requirements in understandable code such as Behavior Driven Development (e.g. Gherkin language with Cucumber)
- Tools to generate test data
- Tools to write scripts to execute the automation
- Tools to track requirements and issues
- Tools for analysis and reporting

It's important to understand how the seemingly disparate test organizations may be able to leverage each other's assets by using existing enterprise licenses or qualifying for price reductions through combining seats. Seek available expertise such as the team running the tools on the Hanscom Air Force Base MILCLOUD or the automation SMEs at SPAWAR's RITE group. Once the tools are chosen, learn them. Practice with easy test cases, on-line tutorials, and vendor documentation to become proficient as quickly as possible. Also, ensure the tools can be loaded on government computers and the selected tools can work together.

Test Automation Framework. Because the selected suite of tools will not necessarily work seamlessly with each other, there can be substantial integration issues that must be overcome. A test automation framework that effectively, efficiently integrates the selected suite of tools needs to be considered. A skilled test automation architect or seasoned programmer insourced or outsourced may be necessary for this function and to get the entire automation suite properly stood-up.

Be sure to consider the expected manpower and selected tools when making the framework/platform decision. A test team enabled with the right AST skillsets and tools needs to be resourced to create an adequate AST framework. A framework can be thought of as the environment containing the components, artifacts, and test libraries needed to automate testing the SUT. An example would be a simulation model that stimulates the SUT allowing automated operations to achieve their desired function. Considerations include the proper client/slave configuration, selecting an operating system, networking and potential use of the cloud, accounting for simultaneous user needs, existing information assurance and classification needs, and hardware requirements.

An automation framework can include one or several AST tools. The tools can control the entire test process including finding software failure and output analyses. A notional example of a simple framework from Distributed Common Ground Station-Navy Increment 2 (DCGS-N Inc 2) is shown in Figure 5.14.

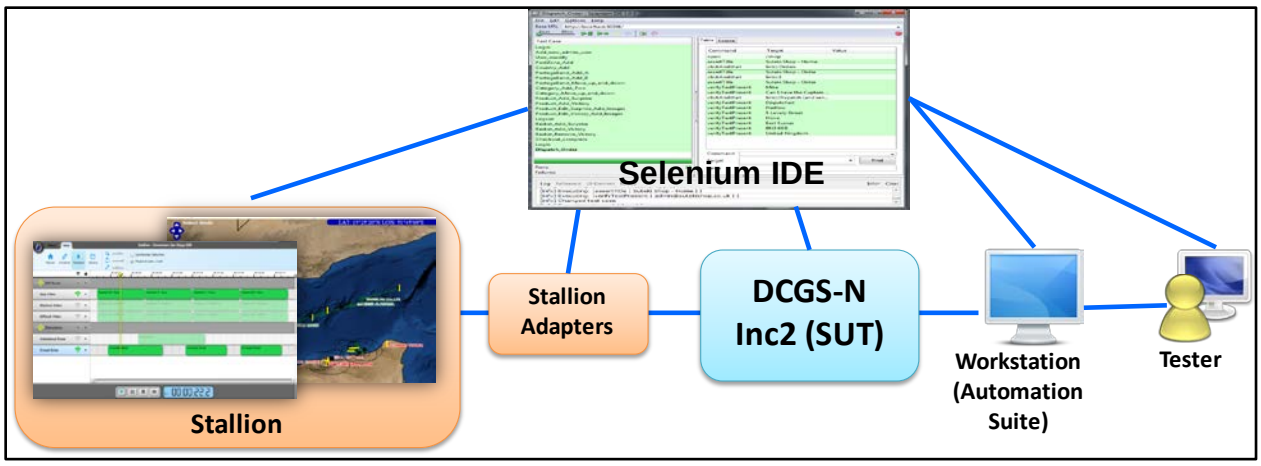

Figure 5.14. Notional DCGS-N Inc 2 Automated Test Tool Integration Framework

Data Collection and Analysis System. Another essential element of the test automation framework is the data collection and analysis system. It is best to first determine the needs and figure the right approach for collecting the data output from the automated testing. Then set the system up to capture the data, which may be as simple as a spreadsheet or as complex as a distributed database management system. Consider running prototype tests with negative testing (inducing faults) to identify areas where greater detail is required for log and output files. It will be important for the analysis effort that faults are either set apart into distinct files or easy to recognize among the output data. This work to identify where the software error occurred along with information relevant to the error will be useful in root cause analyses. Generally, it is important to decide how best to reduce, analyze, visualize, and post-process the data to maximize insight into SUT performance.

5.3.2.1. *Inputs:*

- Select resources needed based on past efforts

5.3.2.2. *Deliverables:*

- Procure resources for personnel, tools, the test automation framework and the data collection and analysis system

Bottom line: Recruit your automators internally or externally recognizing that only a few highly skilled automators are needed to be successful, but they need to be functionally-oriented and especially committed to automation. They also will need an appropriate suite of tools, solid framework building experience, and a comprehensive data collection and analysis system.

5.3.3. Task: Manage Planning and Execution of Initial Pilot

The successful automation journey begins with an initial pilot automation project based on a small subset of easy-to-automate requirements making the likelihood of team success high. This pilot effort will force the test team to start working together to overcome the essential challenges of automation. The team will likely have manually tested the pilot's requirements. If possible, work the software developers into the pilot project. The manager needs to be the encourager throughout the process but also attentive to areas needing additional resourcing. The pilot will form the basis for the future of automation. The steps to follow for the pilot project are to:

- Identify limited scope objectives
- Develop pilot test plan
- Create automation scripts and compare to manual tests
- Monitor test and team performance
- Report results
- Determine next steps to scale pilot to additional requirements

Pilot test cases are derived from the prioritized requirements matrix and should be viewed as relatively easy to automate. A limited-scope test case may incorporate several requirements and multiple test cases in one test scenario. Query the system experts or look to historical testing and develop test case scenarios that are relevant to future tests. This aspect of the process often consists of an experienced system operator or manual tester sitting down with the automator and automating a manual test case using record and playback functionality. Find areas to automate that contain similar controls (e.g., objects like calendar widgets) to those found in other areas or systems so that successes can be rapidly extended to similar systems that use the same controls.

Although limited in scope, the pilot project should be selected for its ability to project automation success across multiple systems. For example, the automation of either populating or retrieving data from a grid control (a table-like object with rows and columns of data) in one part of the SUT will result in any other similar grid control within the same or other systems to now be more easily automated. Test cases should be easy to update in order to grow the team's comfort with and confidence in automation making the team ready for deeper testing.

The manager should develop a test plan even though these are relatively small development test efforts. The test plan should follow the general structure of Automation Software Test Plan created in the Planning Phase. This overall test plan should be updated based on the results of the pilot project.

The team will be developing scripts and executing partial tests as capability matures. Managers should provide guidance and encouragement via regular meetings to help the team achieve interim goals. Conduct code reviews to enforce standards and documentation. Important metrics for the pilot will be how the automation compares to the manual testing in terms of setup, overall time, accuracy, repetitiveness, and coverage.

Requirements change frequently so a detailed documentation trail is helpful for future executions. Carefully consider the new automated testing paradigm, which may allow for broadening the test scenario coverage. Then work on the data feeds requirements, the method of fusion, and the approach. Based on results, determine the next steps:

- Automate more requirements
- Scale current automated requirements
- Continue with same requirements
- Pause with current requirements while team learns more
- Abandon automation

5.3.3.1. *Inputs:*

- Easily automated requirements, tools, and automation framework

5.3.3.2. *Deliverables:*

- Pilot test plan, findings, and recommendations

Bottom line: Do not underestimate the importance of the pilot automation project. Lead the effort with a solid test plan by automating requirements that will keep the team motivated while honing their skillsets.

5.3.4. Task: Conduct Design Review

The Design Review is the primary milestone the team should be working toward during the first three phases (Figure 1.1 Assess, Plan, and Design). This review will be the graduation exercise that will show upper level management that the team is prepared to conduct a robust program of automated test activities for the long term. The manager will lead this review with the goal to demonstrate to leadership the team's readiness to successfully execute test automation.

The Design Review should include many of the elements already addressed in the Automated Software Test Plan. Among these elements are well-documented program code from the pilot tests, variable naming standards, proper configuration management of automation artifacts, tool versions and release levels. Documentation can also include instructions for pre-test setup and steps to run the automation, and instructions on how and where to make maintenance changes to the TAS. Evidence of readiness should include a demonstration of initial capability with smaller prototype scripts. The manager should clearly explain the ROI of the project along with assumptions to provide confidence to stakeholders and senior leadership of the value for continuing with automation.

5.3.4.1. *Inputs:*

- Criteria for review readiness

5.3.4.2. *Deliverables:*

- The Design Review planned and executed
- Documentation of TAS

Bottom line: Showcase the team's accomplishments in automation from pilot results and articulate a clear path toward future automation success.

5.4. Test Phase

Manager Checklist for:
Test Phase

- ❑ **Approve and Schedule Project**
  - ❑ Update required changes from Design Review
  - ❑ Schedule resources as further automation is imminent
- ❑ **Manage to Plan**
  - ❑ Monitor test execution
  - ❑ Schedule regular standups and reviews
  - ❑ Provide team additional resources if needed
  - ❑ Review automation test output reports
  - ❑ Be aware impacts on automation solution of changing SUT
- ❑ **Identify and Manage Variances**
  - ❑ Monitor metrics for significant deviations
  - ❑ Adjust automation test execution based on deviations, unexpected events, and identified opportunities

Figure 5.15. Manager Checklist for Test Phase

Management roles during test execution, Figure 5.15, are focused on getting the team the resources they need for automation and keeping them on schedule within budget. Managers should be prepared to respond to unexpected events and costs with mitigation strategies planned in advance. There will often be cultural challenges and possible political battles where the team will need management support reaffirming the program's commitment to automation.

5.4.1. Task: Approve and Schedule Project

Upon the outcome of the Design Review, leadership will need to make a final automation decision. The 'Go' decision to proceed into automation is not binary as in the Assess Phase, Figure 5.2, as more granularity is needed to indicate the degree of automation that best serves the testing environment. Important factors for management to consider in the decision to proceed include:

- Results from pilot tests
- Readiness of the framework
- Test results of the automation solution (dog-fooding)
- Confidence in team from walk-throughs and technical interchanges
- Funding profile
- Scheduling

The macro-level schedule for test execution in the POA&M needs to be revisited. Tasks need to be updated and sequenced paying close attention to the interdependencies. Test milestones and in-progress reviews need to be scheduled outside of the continuous review cycle that managers often have ongoing during test.

Now that testing is imminent, the manager needs to de-conflict schedules to ensure resources are available to handle the current (or pending) workload. It is likely that the resources will be test assets shared between automated and manual testing or with other programs. These resources include:

- Personnel
- Facilities (e.g. software test integration lab)
- Equipment (laptops, servers, software applications…)

5.4.1.1. *Inputs:*

- Identified project, resources, and test automation maturity

5.4.1.2. *Deliverables:*

- Approval to proceed with updated detailed schedule

Bottom line: Management must proactively lead test execution with a detailed schedule ensuring the availability and timely use of critical assets.

5.4.2. Task: Manage to Plan

Management must actively track the team's progress during testing. The plan is obviously a good starting point to assess if the team is on track, but unknown limitations and execution challenges will inevitably require the manager to adapt the plan. Remember that resources actually used for automation can be difficult to accurately estimate so the need to update updates to the original plan. Constant communication is the key to identifying problems early enough to keep them from derailing the automation effort.

Managers need to be reviewing all available information on the status of the effort looking for departures from the plan, as well as impediments and unforeseen alterations. This Test Phase (see Figure 5.15) task is focused on the standard program management functions of controlling cost, tracking schedule, and assessing performance. Important resources for the overall automated software test quality assurance may include:

- Periodic team reports
- Regularly scheduled stand-ups and status update meetings
- Metric updates from test output reports
- Automation code reviews
- Accumulated and expected costs
- Labor hours.

Managers will need to be actively leading the team by engaging the team both as a group and individually as team members and providing the needed resource support. They will need to monitor closely the resource consumption and burn rates to ensure the program stays reasonably to the glide path. Additionally, managers will need to ensure that other personnel (e.g. database admin., SMEs, etc.) the automators need support from are available as scheduled.

An important aspect of test execution that automation must consider is the changing SUT and TAS. It is essential to have a reliable configuration management system in place as these changes to both the SUT and the automation framework are common across DoD applications. Configuration control is the

process for managing and tracking these changes. There are numerous tools available to help an AST program with configuration control including JIRA and Subversion-Source Control, which are hosted on the open-source (for DoD) Defense Intelligence Information Enterprise (DI2E) network. Independent of the tool selected, the team must enforce a disciplined process of keeping the database up-to-date regularly as some test cases may not run correctly on a different version of the SUT. Keep detailed records of changes and assign points of contact to track specific capability areas.

5.4.2.1. *Inputs:*

- Automation plans and actual test results

5.4.2.2. *Deliverables:*

- Management progress and assessment reports

Bottom Line: Actively manage the execution of tests by frequently engaging, but not stalling, the team through different channels while ensuring their resource needs are met. Expect, track and respond to changes in the SUT and the TAS.

5.4.3. Task: Identify and Manage Variances

Managers have the responsibility to proactively lead the automation effort to avoid show stoppers. They should mitigate the likely risks and respond with effective, timely solutions. Many of the tasks outlined in this guide have already addressed the need for management's continuous awareness of metrics and deviations from plan. To detect problems with the metrics, simple graphical displays and descriptive statistics are critical and likely all that will be required. The key is to determine up front how much variance from the expected metric levels is acceptable. Intuition should not be discounted, but the application of statistical process control methodologies (in the STAT toolbox) may be beneficial.

Variance analysis, though relatively straight-forward, can be improved with a few best practices. Figure 5.16 shows a graph of cost variance deviations for manual vs. automated testing over a 12-month period. Stephen Few recommends:

- Using bar charts and stacked bar charts as the most common graphs although they are not as efficient as line graphs.
- Focus should be more on understanding the differences—show a reference line and band of acceptable performance.
- The percentage change from a standard over time is usually most informative.

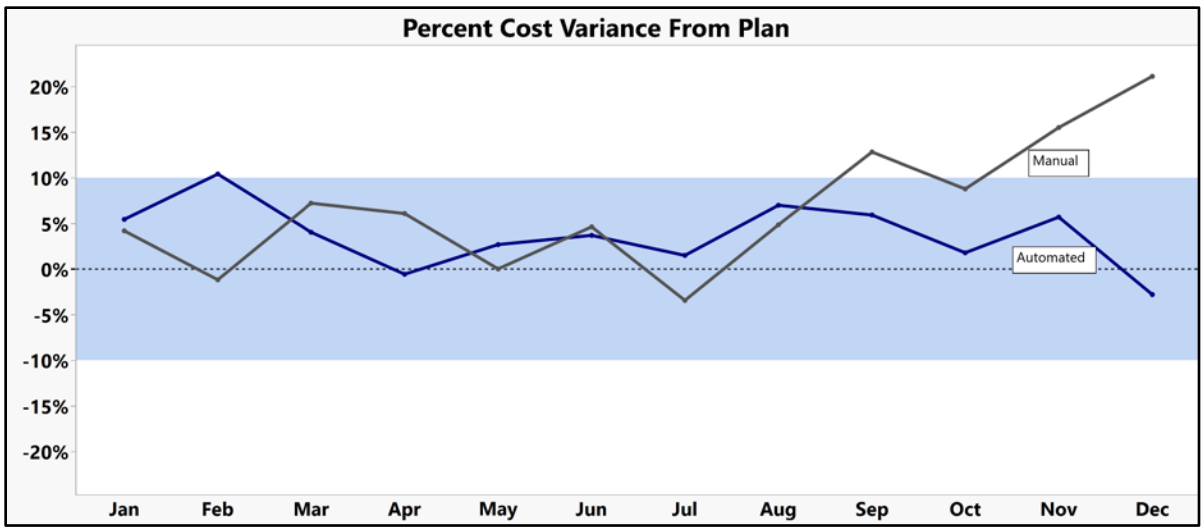

Figure 5.16. Notional Variance Analysis Focusing on Percent Deviation and Bounds

5.4.3.1. *Inputs:*

- Automation plans and actual test results

5.4.3.2. *Deliverables:*

- Variance analysis report on key performance metrics

Bottom Line: Meticulously conduct variance analysis on key performance metrics making sure they stay within reasonable bounds.

5.5. Analyze and Report Phase

Manager Checklist for:
Analyze and Report Phase

- ❑ **Determine Automation Effectiveness**
  - ❑ Calculate and assess coverage (lines of code, requirements, functional path) metrics
  - ❑ Calculate and assess time metrics
  - ❑ Calculate and assess defect identification and correction metrics
  - ❑ Calculate reusability metrics
  - ❑ Prepare report focusing on the trends
- ❑ **Lead Analysis Plan**
  - ❑ Summarize test analysis strategy from test oracle decisions
  - ❑ Develop analysis and findings strategy with analysts and automation team
- ❑ **Manage Defect Reporting Process**
  - ❑ Manage defect database and tracking system
  - ❑ Lead root cause analysis efforts
  - ❑ Manage software failure review board process
  - ❑ Analyze trends over time, apply reliability models as required
- ❑ **Review Costs**
  - ❑ Create or update the budget report
  - ❑ Develop cost containment strategy as required
- ❑ **Communicate Results**
  - ❑ Prepare dashboards of metrics
  - ❑ Engage stakeholders to resource additional automation requirements
  - ❑ Prepare ROI report with cost drivers, variances, and automation benefits
  - ❑ Decide next steps: status quo, automate more, automate less, or abandon automation

Figure 5.17. Manager Checklist for Analyze and Report Phase

Analyze and report phase, Figure 5.17, considers how to effectively understand and communicate the output from the automation effort/project. The automation process is often a continuous process frequently producing artifacts the team needs to analyze to determine if there are major issues or other key performance indicator deficiencies. The team should be updating metrics which will inform future automated test activity. Do not overlook the ability to automate the output analysis activities. The manager is responsible for reporting the overall program status and various metrics to stakeholders. The initial success of automation does not guarantee its continuing and overall success. The manager has to continually champion the effort and leverage these reports to demonstrate progress.

5.5.1. Task: Determine Automation Effectiveness

Leadership is always interested in the business case for automation as they look to right-size the automation program. The manager must repeatedly update long-term performance metrics that capture how well automation is working relative to the manual. The costs captured in the previous task

must be weighed against the value added. These measures of value quantify the improvement in software testing.

Automation's value can be quantified with the metrics discussed earlier in the Planning Phase (refer to Figure 5.10) and shown with notional data in Table 5.2 with trends highlighted in the right-most column.

Table 5.2. Metrics of Automation Effectiveness

Metrics	Last Period	This Period	Trend / Change
% automatable requirements tested	53%	58%	5%
% improvement in automation script development time	67%	68%	1%
% automated (#test cases automated/total possible to automate)	41%	44%	3%
% automated executed (# test cases executed with automation over # test cases	31%	35%	4%
% increased coverage in lines of code tested	37%	38%	1%
% increased coverage of expected operational paths and use cases	42%	40%	-2%
% time savings improvement automated test sequence over manual execution	25%	25%	0%
% test cycle time versus development cycle time	10%	8%	-2%
% Defects discovered from automation	76%	76%	0%
% of automated scripts that are reusable	10%	12%	2%

The manager also needs accurate, reliable, and realistic measures of the parameters themselves to make an accurate overall assessment. It is best to track and have visibility to a detailed level on all of the metrics. Recognize that communicating the overall program success to stakeholders and senior leadership using a set of metrics will require some degree of rollup. This rollup could be in the form of red, yellow, green, and blue ratings along with the current trend. Most useful is the improvement over a manual testing approach in terms of the depth and velocity of testing, as well as the volume of output data collected for analysis and subsequent decision making.

An important consideration in determining success is the morale of the team. Hopefully, the project has experienced some automation successes and the team has become energized with this innovative approach leading to more effective and efficient testing. The team also understands automation has not replaced the manual testers, but has allowed for a better overall quality software product.

5.5.1.1. *Inputs:*

- Automation status and metrics quantifying overall benefit

5.5.1.2. *Deliverables:*

- Status reports, improvements over baselines

Bottom Line: The promise of automation needs to be quantified realistically, understanding that success may not be immediate but can be gradually achieved over time.

5.6. Lead Analysis Plan

Prior to any test execution, except maybe the pilot test event, the manager should meet with automators and analysts and determine the proper plan for data output analysis. The primary driver of the test analysis strategy are the test oracle decisions, or what information is collected to indicate success, failure, or related issues associated with the software program as identified through testing. The practitioner is intimately involved with the test oracle challenge and decisions and this information

feeds the analysis plan. The manager then works with this information in launching the discussions on the analysis plan. Test objectives are also very useful to focus the analysis needs.

The manager's role in this effort is primarily one of guidance and leadership, with special attention to the needs of the program at this stage of the acquisition lifecycle. In particular the manager should facilitate the analysis plan discussion by ensuring the team recognizes what key information needs to be collected and analyzed in this test phase to inform the next stage of software development or enhancement. In the case of operational testing, the analysis plan should address fitness for operational use and quantify performance measures that can inform recommendations to field the software.

5.6.1.1. *Inputs:*

- Test oracle decisions, output data to be collected, test objectives

5.6.1.2. *Deliverables:*

- Analysis Plan

Bottom Line: Well ahead of any automation execution, a key success ingredient is a viable and effective strategy for output data analysis. Linked directly to test objectives, data collection and reporting needs, the analysis plan directs the analysis activity to come.

5.6.2. Task: Manage Defect Reporting Process

An essential management function in automation is tracking the trends in software defects. A disciplined approach to tracking software defects often documented in Software Trouble Reports or other similar format is necessary to fully realize the benefits of software testing. A closed-loop system such as a Failure Reporting, Analysis, and Corrective Action System (FRACAS) database provides excellent visibility and accountability. Some fields include conditions such as whether a failure occurred, time/date, system status, initial corrective action, point of contact to own failure, and current status. Many of the defects will be subject to Software Failure Review Board (FRB) actions, and the FRACAS database is created to provide the required background information and record FRB recommendations. The team should institute a process as part of the regular battle rhythm to update and review the FRACAS database.

By tracking automation execution time variances one can find clues for problems and errors yet to be discovered. For example, the following may cause automated tests to run slower than expected:

- Additional traffic on network
- Memory leak
- Server load balancing issues
- Test automation library code updates

It is helpful to classify the software failures based on mission criticality. This serves two purposes: 1) to quantify reliability in terms of defects per thousand lines of code and 2) to prioritize failure modes. The FRACAS database should allow the team to view defects over time and to assess how reliability is improving or degrading. Data visualization methods can easily stratify based on failure mode, failure criticality, requirement, test case, version, and so forth to see if there are trends over time. In addition, there are statistical models that can help quantify system performance.

If the SUT program management has placed value on finding and correcting software failures, then the defect rates should decrease. There are several software reliability growth models outlined in IEEE 1633, Recommended Practice on Software Reliability (2016, Annex C Supporting Information on Reliability Growth Models) that can be used to predict future failure rates based on the current failure rate, management aggressiveness in finding root causes, remaining test time, and the quality of the contractor's software development program. Note that defect analysis can be quite resource intensive and needs to be planned for upfront.

5.6.2.1. *Inputs:*

- FRACAS database, information on automation execution

5.6.2.2. *Deliverables:*

- Data visualization methods for defect reporting and analysis

Bottom Line: Prepare for defect analysis and reporting by establishing a defect data collection vehicle such as a FRACAS database. Go the next steps by planning for reporting major deficiencies to the Software Failure Review Board and recognize that the process is resource intensive.

5.6.3. Task: Review costs

The decision to automate was predicated on an expected benefit to cost ratio. It is common to have many unanticipated costs and costs that vary significantly from the original estimate. Management will need to periodically review costs and adjust program direction if necessary.

Consider preparing a budget report with the following entries:

- Cost category: automation tools, licensing/maintenance, hardware, training, personnel indirect costs, contractor costs, etc.)
- Estimated cost at Go/No Go decision point
- Revised estimated cost (if applicable)
- Actual costs with variance analysis
- Recommendations for way ahead

The budget report should show the actual and budgeted costs over time with explanations for large variances along with cost drivers. Managers need to provide recommendations for cost containment strategies to keep the automation effort viable.

It may also be helpful to show some estimated cost avoidance by discovering the software defects earlier in the lifecycle due to the automation effort. The National Institute of Standards and Technology (NIST) provides some guidelines stating it would be reasonable to expect a savings of 5x if the defect is discovered in unit testing versus post-release. Table 5.3 displays demonstrates a notional cost savings example.

Table 5.3. Cost of Defect Repair as a Function of Software Development Phase

Resource	Coding/Unit Test	Integration Test	Beta Test	Post-Release
Hours to Fix	**3.2**	**9.7**	**12.2**	**14.8**
Cost to Fix	**$240**	**$728**	**$915**	**$1,110**

After the initial investment to achieve AST capability, an evaluation needs to be made periodically on its worth. The team has a variety of options to consider:

- Go deeper with the current effort
- Scale up with more user load
- Automate new requirements
- Adopt different tools
- Transition from GUI to code-based API tests
- Conduct contingency test cases
- Abandon automation.

The manager should have a rough order of magnitude (ROM) cost for each of the feasible alternatives that can be weighed against the expected improvements in the metrics for success in Table 5.2.

5.6.3.1. *Inputs:*

- Expected costs, as allocated by resource category

5.6.3.2. *Deliverables:*

- Actual costs by category with understanding of what caused the variance

Bottom Line: Track costs by category and compare to expected costs noting the drivers along with potential cost containment strategies. Be prepared to discuss costs associated with the next step of automation—whether cancelling the effort, incrementally improving automation, or significantly expanding the program.

5.6.4. Task: Communicate Results

Managers need to keep leadership, stakeholders, and their team informed as automation progresses. Different levels of reporting results are required for each audience, but all must clearly state what is working well, where there are opportunities for improvement, and what the next steps in the automation journey are.

While a tester is mostly interested in each individual test outcome, the test manager is looking for overall test coverage from the automated tests executed. The program manager, on the other hand, is looking for trends and is expecting to see fewer defects with each successive release. Each stakeholder in this chain needs different information at varying levels of granularity to inform decisions or corrections, and maintain progress.

Managers need to ensure that automators capture data at sufficient fidelity so that it can be aggregated and presented to meet the needs of any and all stakeholders. Often, it's not the lack of data being captured that prevents management from receiving relevant reporting, but rather it's the lack of planning to use captured data in a meaningful way to inform all parties. Additionally, careful record

keeping of costs and benefits will go a long way in helping frame the future value for automation to stakeholders.

Dashboards are an effective way to display information because they enable data to be visually communicative with less of a requirement for detailed analysis. Figure 5.18 shows an example dashboard from the Micro Focus Quality Center website displaying the metrics of unresolved defects by severity code, test execution status, requirements coverage, and reviewed requirements. For analyzing trends, ratios, and min/max levels, dashboards are very effective. However, for root cause analysis of SUT errors, more detailed reporting or inspection of logs (both SUT and TAS) may be required.

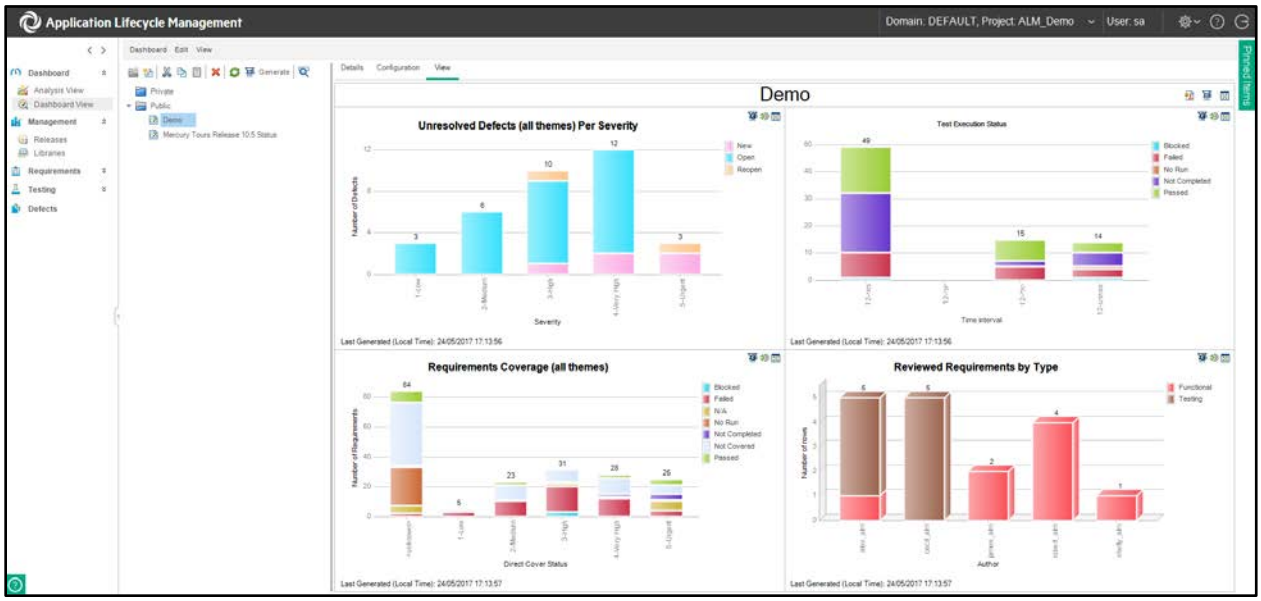

Figure 5.18. Example of Effectiveness Displaying Aggregate Test Data in a Dashboard[9]

5.6.4.1. *Inputs:*

- Stakeholder reporting requirements

5.6.4.2. *Deliverables:*

- Purpose-built dashboards, reports, logs, etc.

Bottom Line: Effective communications can be implemented for all stakeholders by being mindful of the best way to report information to each audience.

[9] https://software.microfocus.com/en-us/products/quality-center-quality-management/overview

5.7. Maintain and Improve Phase

Manager Checklist for:
Maintain and Improve Phase

- ❑ **Standardize and Document Approach**
  - ❑ Develop checklists to enforce standard operating procedures
  - ❑ Create detailed documentation on automation program
  - ❑ Ensure team regularly reviews and updates team documentation
- ❑ **Replicate Approach**
  - ❑ Determine portability of solution and most likely programs to benefit
  - ❑ Empower team members to participate in broader test automation communities of interest
  - ❑ Prepare ROI report with cost drivers, variances, and automation benefits
  - ❑ Decide next steps: status quo, automate more, automate less, or abandon automation
- ❑ **Maintain Automation Capability**
  - ❑ Actively monitor configuration changes to SUT and test automation solution
  - ❑ Manage repeatability and reproducibility issues
  - ❑ Fund manpower and tool updates for maintenance function
  - ❑ Track the maintenance burden to help inform alternative solutions going forward
- ❑ **Manage Continuous Improvement**
  - ❑ Aggressively search for new tools, methods, and personnel to improve test automation solution
  - ❑ Apply lessons learned and experience to automate new requirements more efficiently
  - ❑ Ensure robust funding is available for training the team to sharpen skills on current methods and learn alternatives

Figure 5.19. Manager Checklist for Maintain and Improve Phase

The Maintain and Improve Phase, Figure 5.19, addresses documentation of the existing automation capability, activities to maintain the automation, continuous improvement in the current TAS, and the expansion to other programs. The manager's role in leading these initiatives is essential as team members may be singularly focused on current automation tasks while not necessarily considering how to improve and replicate their methods.

5.7.1. Task: Standardize and Document Approach

Automation is most successful with an understandable, deliberate, and standardized process. The procedures should be organized and follow a "checklist" mentality. Standardization ensures stability in the test process so that any anomalies can be attributed to the SUT rather than the TAS. Continuous communication between managers and team members will ensure all changes are necessary, understood, documented, agreed upon, and beneficial to the testing needs.

A key component of the standardized approach is a repeatable and reproducible process. Automation may not be stable especially if the system is a GUI capture where the exact same test may not produce the same results with same tool (repeatability) or different tools may produce inconsistent results (reproducibility). The team will need to test the stability of scripts to see if they produce the same

output each time, for the same set of inputs in a known environment. For AST, repeatability would be getting similar results from the same tool or suite of tools and reproducibility would be consistent results across multiple tools and solutions. The team may consider recommending negative testing to trigger failures and determine if the software responds correctly.

Management will need to identify those processes that are more error prone. They'll want to find the commonality in robust solutions that work across many applications and flag those as best practices. The hope is the evolving automation effort and new ones benefit from a generalized approach found to be relatively stable across a variety of testing scenarios. Organizations can use the capability maturity model integration (CMMI) template (Figure 5.20) to help evaluate the maturity of their software automation process.

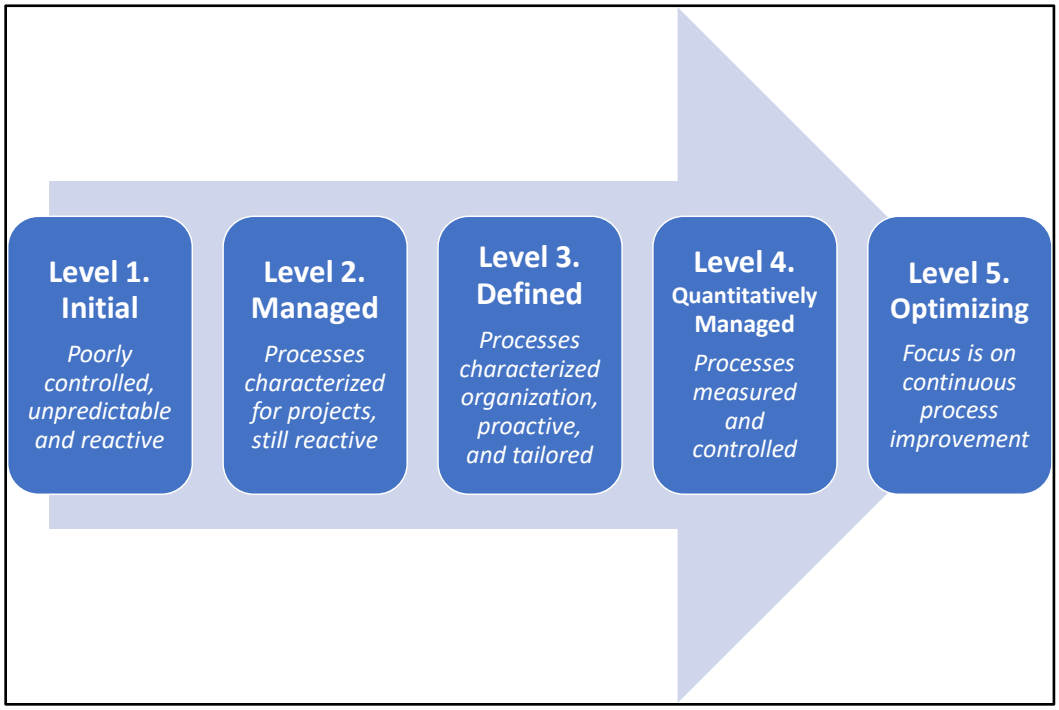

Figure 5.20. CMMI Provides a Template Which Automation Can Use to Ensure Process Maturity[10]

The complexities of automated test necessitate diligence in documentation. The manager must ensure the team is meticulous about documenting the entire automation program despite the resistance of team members to do so as the additional time burden may be difficult to fit in with testing demands. The documentation will allow new team members to quickly understand the environment and mitigate the consequences from personnel turnover. Example documentation includes:

- Internal comments in developed code
- Framework details
- Test procedures
- Data sources

[10] https://cmmiinstitute.com/learning/appraisals/levels/. Accessed: 4-Oct-2018

- Test event logs
- Analysis methods and results

5.7.1.1. *Inputs:*

- Identify level of documentation required

5.7.1.2. *Deliverables:*

- Completed software test documents

Bottom line: Regular documentation and updates will lessen the burden on the team and increase the quality of the TAS. A documented TAS will facilitate debugging and maintenance, and provide the blueprint for repeatability.

5.7.2. Task: Replicate Approach

The lessons learned and successful methods from the current program will form the foundation of an effective TAS that can be extended to new requirements. Furthermore, the solution will be standardized and well-documented so that it can be efficiently implemented across other programs saving valuable resources across the enterprise. The team will be able to mentor others in similar roles along their automation journey by sharing knowledge and recommending proven tools and methods.

The team's enhanced automation skills may result in a higher level of achievement. This improved credibility will be the key to gaining traction in spreading automation to other programs. Automation can be spread via existing AST groups, online resources, white papers, and case studies.

5.7.2.1. *Inputs:*

- Documented approach to automation

5.7.2.2. *Deliverables:*

- Implemented automation following documented approach

Bottom line: Repeated adoption of automation across projects and programs will require a well-documented initial blueprint so that early successes can help fuel future successes.

5.7.3. Task: Maintain Automation Capability

Empirical evidence suggests that once an automated test is executed and analyzed, the job unfortunately is not done. A pervasive theme across all DoD services is how much effort is required, though not necessarily resourced, just to maintain their automation capability. Changing SUT configurations, updated tools, new tools, and adding new personnel are just a few of the dynamics that require an updated AST solution. Depending on the tools selected, Graham and Fewster suggest 25% or more of your budget may have to be devoted to the maintenance function.

Configuration control is a cornerstone of the maintenance function. Systems and architectures are not stable for long across the DoD enterprise. Fortunately, the design phase (refer to Figure 5.12) has already prepared the team by having them design a configuration control system to track changes linked to test cases and requirements. Complicating things even further are the many file management system options requiring a disciplined approach is critical.

Unless the team has advanced tools and skillsets to make the TAS robust, a change to the SUT will likely require some change to the automation. Consider for example a GUI-based AST, which only takes a minor change in the expected SUT output graphic for the automation to report an error of not finding the image. This error would be due to improper application of automation and not the system under test. The key question becomes "how to best manage updating the test scripts over time?" Different SUT configurations will correspond to specific AST framework versions. Another question is whether a single individual should be responsible for developing and executing the scripts or whether an automation team collectively updates the scripts. Periodic verification/validation programs are necessary to ensure effective compatibility between the SUT and automation solution.

The manager must ensure the automation team is allowing the TAS to be responsive to a changing SUT and its interfaces. The manager can help make the team aware of any forthcoming changes that would impact execution so they can expect and prepare for these changes appropriately. TAS code should be updated to account for new tools (which may be required to support any new interface controls) and updates to existing tools. Code may need to be modified to incorporate better coverage or improve other metrics.

Managers should insist the code and scripts be nimble enough to easily account for frequent changes. Script ease of maintenance is possible when code is reusable and data is abstracted so that the code itself does not need to contain SUT information that can be passed to it. Solutions developed with a "design for maintainability" mentality to meet changing system output or to better improve automation metrics will have a substantial return on investment. Plan for future automation success by being aware of 'level of effort' required to maintain your automation capability. Realize that the real power of automation lies in repeated application of the same or similar testing. Thus, it is imperative that minimal modifications are required to your library of functioning scripts. Updates to scripts should be minor, either to execute the same test later when the SUT or test environment changes or to adapt scripts to perform similar but different automation purposes. Know that maintenance loads can vary significantly depending on the automation tools selected.

Actively seeking opportunities to improve the automation can be challenging for a number of reasons:

- Frequently changing system configuration
- Automation tool updates
- Automation tools compatibility
- Migration of existing test data during migration

The manager should motivate the team to continually improve. The regular and consistent communication of metric status will lead to ideas and strategies to achieve better automation results. It is important to create an environment that fosters feedback and constructive critique from peers, mentors, leadership, and experts to sharpen applicable skills. Additionally, management should provide time and training resources for the team to learn new methods and participate in collaborative engagements.

5.7.3.1. *Inputs:*

- Changing system configuration and automation environment

5.7.3.2. *Deliverables:*

- Task plan on TAS updates to include code, tools, environment, data, etc.

Bottom Line: TAS maintenance will likely need to occur every time the SUT changes. Getting advance notice of these changes enables the team to begin thinking about how and what to update.

5.7.4. Task: Manage Continuous Improvement

The manager should give the team, especially those senior members, the opportunity to attend industry test and software automation conferences or workshops, which often have large vendor exhibit halls in order for them to keep up with the technology and see what new products might improve the current automation efforts. Encouraging team members to visit vendor or product websites, view online demos, and subscribe to online blogs and groups of like-minded automators and automation managers.

The requirement to replace an entire tool or toolset from a TAS is not one to be treated lightly as it can be complicated and cause downtime for the automation team. Events that can trigger major disruptions include:

- Tool vendor indicates end of life for product
- Tool vendor has announced new product with additional features and greater performance
- Tool is not operating in a consistent/reliable manner
- The Integrated Development Environment (IDE) for the SUT is being replaced by one which contains a whole new set of screen controls

Over time the manager should push the automation team to find ways to streamline existing automation. This could be accomplished by merging tests that have common components or combining very small tests into one larger test. An automated test is clearly faster to execute than a manual one due to the hundreds or thousands of automated tests performance tweaks. Another way to optimize test executions is to run tests concurrently over several computers, or virtual computers. The manager should alert the team of upcoming requirements and how those may translate into new functionality, which will need to be added to the existing automation framework.

5.7.4.1. *Inputs:*

- Known and expected SUT changes

5.7.4.2. *Deliverables:*

- Plan for incremental TAS improvement

Bottom Line: Automation needs to regularly assess current and anticipated SUT changes and use this opportunity to refine, if not replace, code and/or tools. Senior team members, both automators and automation managers should be encouraged and funded to attend industry conferences, and seek other resources to learn of developments that may impact their work.

6. Implementation Considerations for the Practitioner

The test automation practitioner is the hands-on engineer tasked with the process of building and maintaining a sustainable and expandable test automation solution (TAS). This process can be thought of as a software development lifecycle for test automation. There are technical and non-technical aspects to realizing a complete and sustainable solution that the test automation practitioner needs to be familiar with to succeed. With so many test tools available for use in automation, finding the most capable set of tools to customize for a specific automation task is a major undertaking. Fortunately, this guide will facilitate understanding recommended practices and architectural considerations which should apply to almost any test automation endeavor. Though the practitioner is responsible for many of the technical tasks, the most important jobs will be educating, coordinating with, and responding to management. Management may have unrealistic expectations based on vendor demonstrations and other sources of hype. Graham and Fewster have remarked that management actions often result in the death of an automation project, normally by manslaughter rather than murder. Figure 6.1 provides an overview of the tasks by phase for the practitioner. Note that although the phases are the same as the manager section, the tasks are generally more technical.

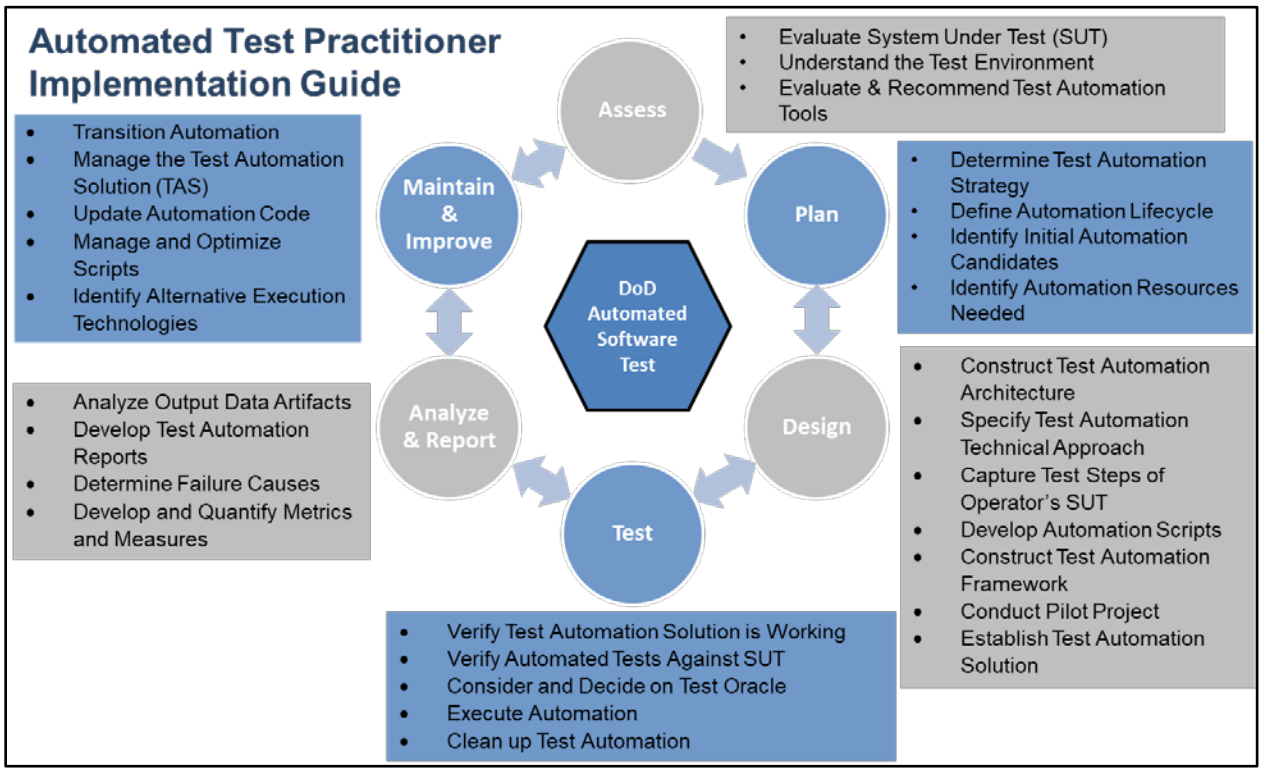

Figure 6.1. Phases and Tasks of Automated Software Test for Practitioners

6.1. Assess Phase

Practitioner Checklist for:

Assess Phase

- ❑ **Evaluate System Under Test (SUT)**
  - ❑ Understand SUT architecture
  - ❑ Determine current test approach
  - ❑ Identify system's operational lifecycle phase
- ❑ **Understand the Test Environment**
  - ❑ Interview the testers and test engineers regarding objectives
  - ❑ Understand test schedule, resources
  - ❑ Identify opportunities to automate
  - ❑ Know the limitations and constraints on test applications
- ❑ **Evaluate & Recommend Test Automation Tools**
  - ❑ Research possible sources for tool selection
  - ❑ Understand feature/function set of tools
  - ❑ Identify fit to organizational skills
  - ❑ Determine compatibility with SUT
  - ❑ Identify tool resources
  - ❑ Determine which tools are good candidates
  - ❑ Educate and inform program on tool capabilities and options

Figure 6.2. Practitioner Checklist for Assess Phase

The assessment phase, Figure 6.2, initiates a process that ultimately allows the practitioner to inform management on the viability of automation for the organization. There may be a temptation to pick the automation tool before the project, but this "cart before the horse" approach will likely backfire. Before even thinking about a tool, the practitioner must first fully understand the environment to be automated. This bigger picture includes the program under study and more specifically the application software, associated interfaces, and the network. The practitioner needs to have basic knowledge and a technical understanding of the overall system under test. There should be a recognition of the level of technical skills within the testing team as well as some awareness of possible technical skills potentially available outside the team. Additionally, through study, demonstration, or investigation, the practitioner needs to understand the capabilities of alternative automation tools as well as the level of effort required to effectively use them. Ultimately, no tool should be selected that is not first demonstrated in its intended environment. A multi-faceted recommendation will lend credibility to the investigation and help management in their decision to move forward.

6.1.1. Task: Evaluate System Under Test (SUT)

The system under test needs to be evaluated as a candidate for automation. SUTs are created using a wide range of commercial and purpose-built technology.

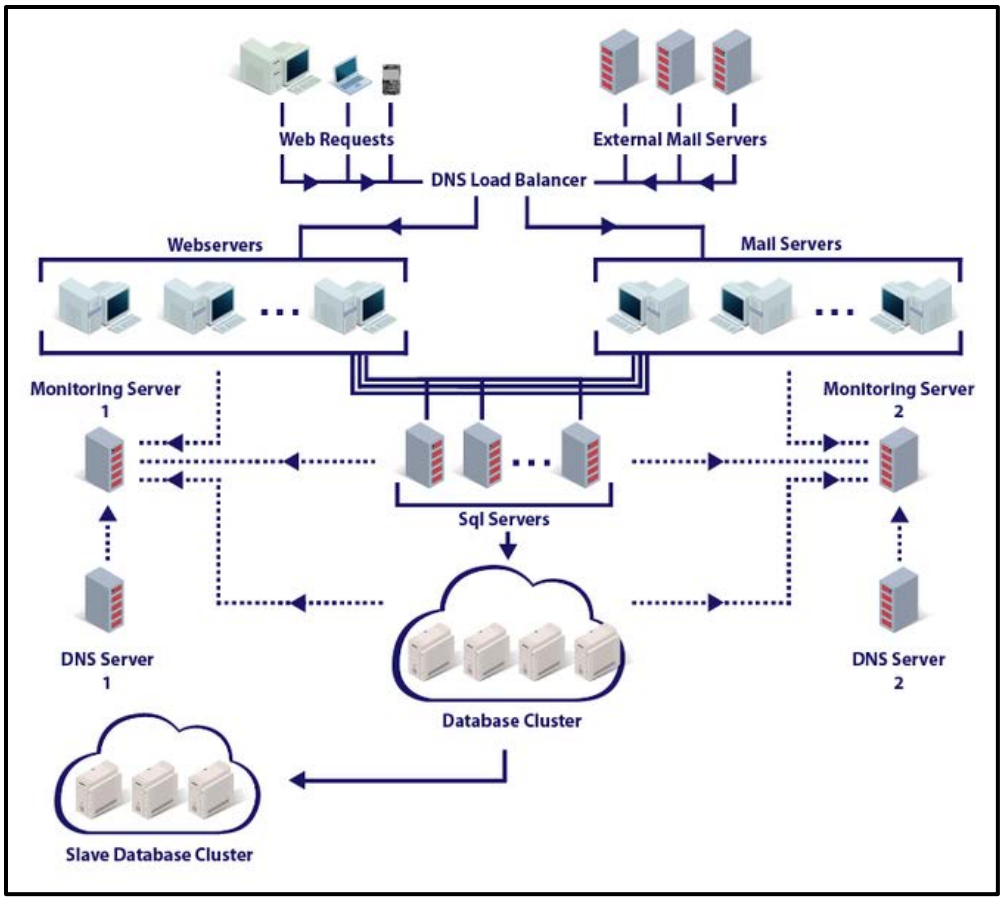

Figure 6.3. A Representative SUT Architecture with which Test Tools Interface[11]

The evaluation of the SUT is necessary to understand the overall SUT architecture and the components which make up that architecture. As shown in Figure 6.3, a complex system may have multiple interface points which may require testing and finding tools compatible with these interfaces. Figure 6.4 shows some of the SUT architecture dimensions that will influence the nature of automation. Understanding how the SUT is currently tested and what are the testing expectations will assist in this evaluation.

[11] https://commons.wikimedia.org/wiki/File:Protonmail_system_architecture_2014.png. Accessed: 5-Oct-2018

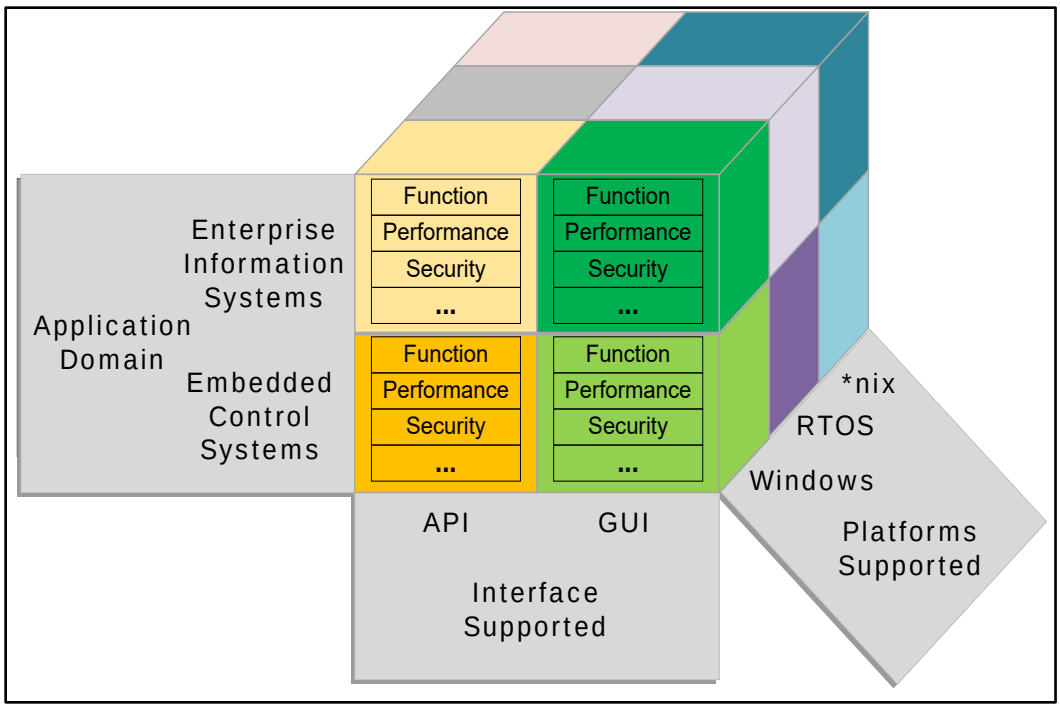

Figure 6.4. Test Automation Reference Architecture[12]

The next topic is the state of the SUT in its lifecycle. Is the current software version just coming online, being currently used in production, or getting ready for sunset and/or replacement? These considerations add additional dimensions to the evaluation for best fit for automation. If the SUT is early in the lifecycle, the software may still be changing often. Consequently, automation would not make as much sense because the automation software would rapidly become obsolete. Conversely, the system configuration may be more stable with only periodic updates, making automation much more effective. As the system nears the end of the Production and Deployment phase, beginning an automation program would not make much sense either unless bridged to a similar new system coming on-line.

6.1.1.1. *Inputs:*

- Selected SUT under consideration
- Current life cycle phase for the SUT

6.1.1.2. *Deliverables:*

- Feasibility of automation with assumptions and recommendations

Bottom Line: Research and understand your SUT architecture and lifecycle early on as it will provide the roadmap for what is possible with automation.

[12] Software Engineering Institute, Carnegie Mellon University, 2018. https://resources.sei.cmu.edu/asset_files/Presentation/2018_017_101_524901.pdf Accessed: 5-Oct-2018.

6.1.2. Task: Understand the Test Environment

In researching and planning for automation in test, you will need to understand the current test environment. Information such as the strengths and shortcomings in existing manual testing and the approaches taken for testing similar systems, will go a long way toward informing the possibilities for automation. To determine if the current test environment is conducive to automation, interview manual testers, test engineers, SUT SMEs, and other experts. Potential interview questions include:

- Do we have a lab set up where we can overlay automation tools?
- Do we need to order new hardware?
- Is there other infrastructure required to effectively automate?
- How long does it take manual testers to test?
- What are the difficulties in testing and the manual solution?
- Where are the errors, inconsistencies, and cases of incomplete testing?
- What is not being tested? Are those good automation candidates based on complexity, time to automate, and availability to automate?
- Are there opportunities to run automated tests overnight or during other off-hours?
- For the known test limitations—why are we not testing these configurations?
- Could automation improve overall software quality by increasing test coverage, effectiveness, and efficiency?

6.1.2.1. *Inputs:*

- Existing test environment, manual testing, SME interviews, and test infrastructure

6.1.2.2. *Deliverables:*

- Informal gap analysis quantifying areas where automation could provide value

Bottom Line: Know the problem you are trying to solve by understanding the current test environments as well as the strengths, shortcomings, and opportunities.

6.1.3. Task: Evaluate and Recommend Test Automaton Tools

Test tools can be obtained commercially, via open source, or custom developed using common programming languages. Identifying a candidate tool requires understanding the tool's capabilities, environments supported, and fit to the organization's skill set. While some tools may provide great flexibility and capability, they likely will also require an engineer with strong programing skills in order to understand and properly use this capability.

Tools come in a variety of packaged solutions. Some tools are more form-based, which allows the practitioner to select or input from various dialog boxes/windows for a simple and straight forward interface. Form-based tools can be used by a broader set of testers because they do not require software development and programming knowledge. However, simpler solutions may not provide the flexibility required to handle a project with a preponderance of complex testing requirements. Tools that allow for programming of scripts (typically in one or more programming languages), will often support more functionality. The downside is that the test scripts may need to be programmed from scratch. This coding can take weeks or months to accomplish. If done properly, the programming effort will be an investment in technology that will reap rewards in testing for years to come.

Figure 6.5, illustrates the categorization by function of several representative tools through the automation lifecycle. As part of the tool evaluation, decide which tools will work for which functions (requirements management, bug tracking, execution…).

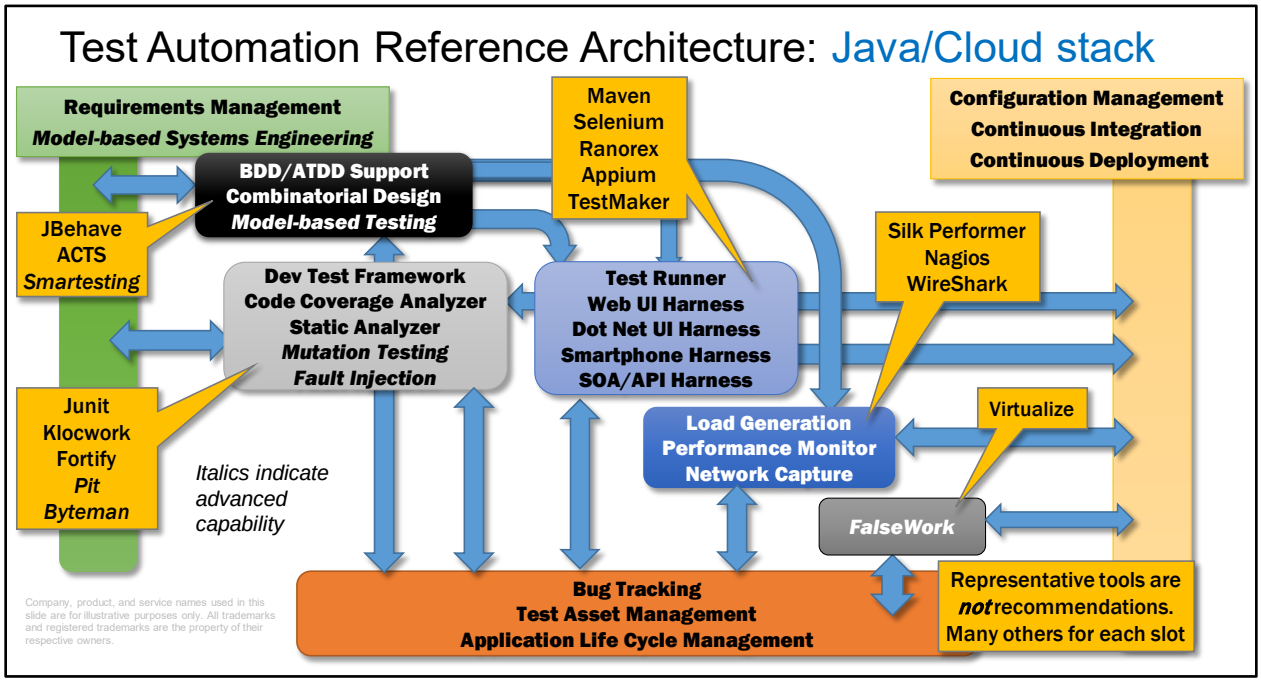

Figure 6.5. Test Tool Automation Reference Architecture[13]

There are numerous resources available to help the practitioner understand tool capabilities, limitations, and use. Some of these include:

- Other units in your organizations and other DoD users
- Trade publications (e.g., Gartner)
- Vendor sites and open-source sites
- Automated Software Testing blogs and forums
- Media clips (YouTube, www.testtalks.com, etc.)
- Conferences

Selection of good tools begins with having a good set of tool recommendations. Some test tools may already exist in your organization. Thus, a good tool has to be both good at what it does and good for the organization. In other words, a test tool can now be selected having satisfied both the compatibility requirement and the matching of organizational technical skills. Programs that rely heavily on operational users and functional experts to assist with testing may be well served by using a forms-based test automation solution. The operational user is often not a person with programming background and would likely get discouraged if required to move from domain-specific testing

[13] Software Engineering Institute, Carnegie Mellon University, 2018. https://www.sei.cmu.edu/ Accessed: 5-Oct-2018.

techniques to programming-specific automation architectures. Likewise, a technical tester who works very closely with developers may be motivated and capable of developing programmatic test scripts. The tools selected should meet the needs of current and prospective test team members.

There are several criteria to consider in evaluating automation tools. Prior to deciding on the best candidates and ultimately the tools to procure, practitioners need to think about:

- Compatibility with SUT
- Compatibility with the team's technical skills, abilities, and experience
- Supportability
- Flexibility
- Affordability
- Understandability, learning curve, and ease of use

Appendix C: Tool Evaluation Worksheet provides a worksheet to help the team evaluate possible tools.

6.1.3.1. *Inputs:*

- Identify test tools by function within the lifecycle for use in automation
- Identify test tool's technical skill requirements

6.1.3.2. *Deliverables:*

- Assessment of compatibility with SUT
- List of tool capabilities
- Tool recommendations
- Feasibility of automation with assumptions and recommendations

Bottom Line: Spend time researching tools to determine candidates that fit your expected automation needs and will be flexible enough to support future growth.

6.2. Plan Phase

Practitioner Checklist for:

Plan Phase

- ❑ **Determine Test Automation Strategy**
  - ❑ Identify key attributes for a successful strategy
  - ❑ Determine possible strategies that would work with current resources and constraints
  - ❑ Understand stakeholder requirements and incorporate them into strategy
- ❑ **Define Automation Lifecycle**
  - ❑ Understand existing SDLC in order to plan effectively
  - ❑ Align with software development approach
  - ❑ Understand milestones in process
- ❑ **Identify Initial Automation Candidates**
  - ❑ Identify straightforward automation candidates to ensure early success
  - ❑ Select an automation candidate that has high relevance/visibility to the organization
  - ❑ Ensure there is agreement from stakeholders on value of automation candidate
  - ❑ Eliminate highly complex systems from initial attempts at automation
- ❑ **Identify Automation Resources Needed**
  - ❑ Work with SMEs (Operations Analysts, Manual Testers, Developers, etc.) to understand system and needs
  - ❑ Identify attributes that will help guide the decision process on automation candidates
  - ❑ Coordinate with manager to establish automation resources required
  - ❑ Ensure software for test automation is stable

Figure 6.6. Practitioner Checklist for Plan Phase

Is your project using a traditional waterfall software development methodology? Has it transitioned to Agile and uses scrum teams? If so, how long are your sprints: 2 weeks, 3 weeks, or 4 weeks? Do you have access to the code or has the contractor delivered the software such that all you have to test is the GUI? All of these factors will affect how you plan to introduce automation into your software lifecycle. With automation, "record/replay" tools will get you automated tests quickly. But these scripts will easily break and require constant attention. How do you make sure that your automation plan sets realistic sights on what can and should be accomplished within the existing project development methodology?

There are many factors to consider when starting to plan for automation, Figure 6.6. These include the complexity of code base, test type and repetition frequency, level of software maturity, richness of the Integrated Development Environment (IDE) and its components, along with an understanding of the architecture and components (e.g., graphical, embedded, cloud). If in an agile development environment, there should be an alignment to the lifecycle so that within a 4-week sprint you can plan to deliver some automation capability. However, a 4-week sprint will not often allow for robust automation capability to be built. So, we need to ensure that incremental automation capability does not become a throw-away solution, as this will not help build a sustainable solution over time. Rather, use the limited sprint time to fully understand the system and the components that need to be automated and start building capability to do regression testing across sprints rather than within sprints.

Over time the core automation capability that is built will become the “plumbing” of a larger solution thereby making each incremental improvement easier to implement due to the existing automation infrastructure.

6.2.1. Task: Determine Test Automation Strategy

A structured approach to determining the test automation strategy will help define a path to its eventual success. Key attributes include the number and type of technical resources available as well as the timeline to grow an automation expertise. This core competence will help determine how aggressively and quickly test automation can be implemented. Additional attributes are the differences between system and software architectures that determine the complexity of the task ahead. The approach needs to be realistic, practical, and attainable. Early successes will provide confidence and acceptance of the automated solution, which in turn will facilitate increasing the automation scope and reach within and across systems.

The test automation strategy in the Planning Phase is not that technical yet and does not get into the details of the framework. Later in the Design Phase, provided in Figure 6.7, the practitioner walks through the development of the Test Automation Solution using the planning phase framework, technical approach, and architecture. Some examples of strategy include: use what we currently have for automation, look at other solutions, find alternatives and perhaps external sources since current tools and staff are inadequate, automate just from the GUI or the application program interface, or consider coding the backend. The selected strategy requires management buy-in and is based on technical skill and know-how.

Some of the questions the practitioner should ask, and likely work with the manager to answer, in developing a strategy are:

- What resources do we have available to help engineer an automated test solution vs. what resources are the “consumers” of a purpose-built solution?
- Are these resources available on a part-time, or fully dedicated basis?
- What is the plan for training, mentoring, and/or acquiring automation skills?
- What test tools does the organization currently own and which are within the realm of the possible to acquire?
- Are the tools adequate for our needs or do we need to look outside?
- What level of funding will be required and what level of funding is available?
- How does the plan for achieving automation fall into the overall software testing approach?
- How do we ensure that automation supports and empowers the test team rather than distracts from or diminishes what they do?

The automation strategy must also consider the stakeholders. Focus should be on identifying who they are, why they are stakeholders, what each will bring to the table and why you need to get their input and buy-in. Example stakeholders include program managers, test managers, engineering technical leads, SUT software development team, and others.

6.2.1.1. *Inputs:*

- Skills and availability of staff, tools, SUT characteristics, and test environment

6.2.1.2. *Deliverables:*

- Select the most appropriate automation strategies

Bottom Line: Match your automation strategy to an honest assessment of your resources and your capabilities to recruit, hire, and acquire.

6.2.2. Task: Define Automation Lifecycle

Test automation has a lifecycle much like software development and the selected methodology should be consistent with other established SDLC methods. The automation lifecycle needs to closely align with the development lifecycle which it intends to support (e.g., Agile, sequential, incremental, iterative, etc.). As many programs are moving towards agile and test-driven development practices, the test automation practitioner needs to plan for and adapt to a rapid sprint-based approach to development. This will have an impact on how automation is delivered to support the rapid deployment of code.

Historically, DoD software projects have adopted a Waterfall methodology where each phase is completed before the next phase starts. This approach has not always yielded consistent results and has often created schedule delays and cost overruns. More recently, DoD has been advocating more Agile methodologies across most of the acquisition programs. This lifecycle methodology departs from the traditional Waterfall approach by creating shorter, incremental deliverables from a collaborative team.

With the Waterfall methodology, the automation engineer had a longer schedule between releases under which to develop the automation solution. This allows for time to design an overall solution. With Agile projects which are measured in sprints (often 2 – 4 weeks in length) developing a robust automation solution may not seem realistic or feasible. Automation is still essential in Agile, it just may be a Sprint behind.

Figure 6.7 illustrates how the automated test scripts are offset from each sprint and form the basis of cumulative regression tests across sprints. Depending on what methodology the overall project or program has adopted, a corresponding approach to automation must be used that will align with the methodology.

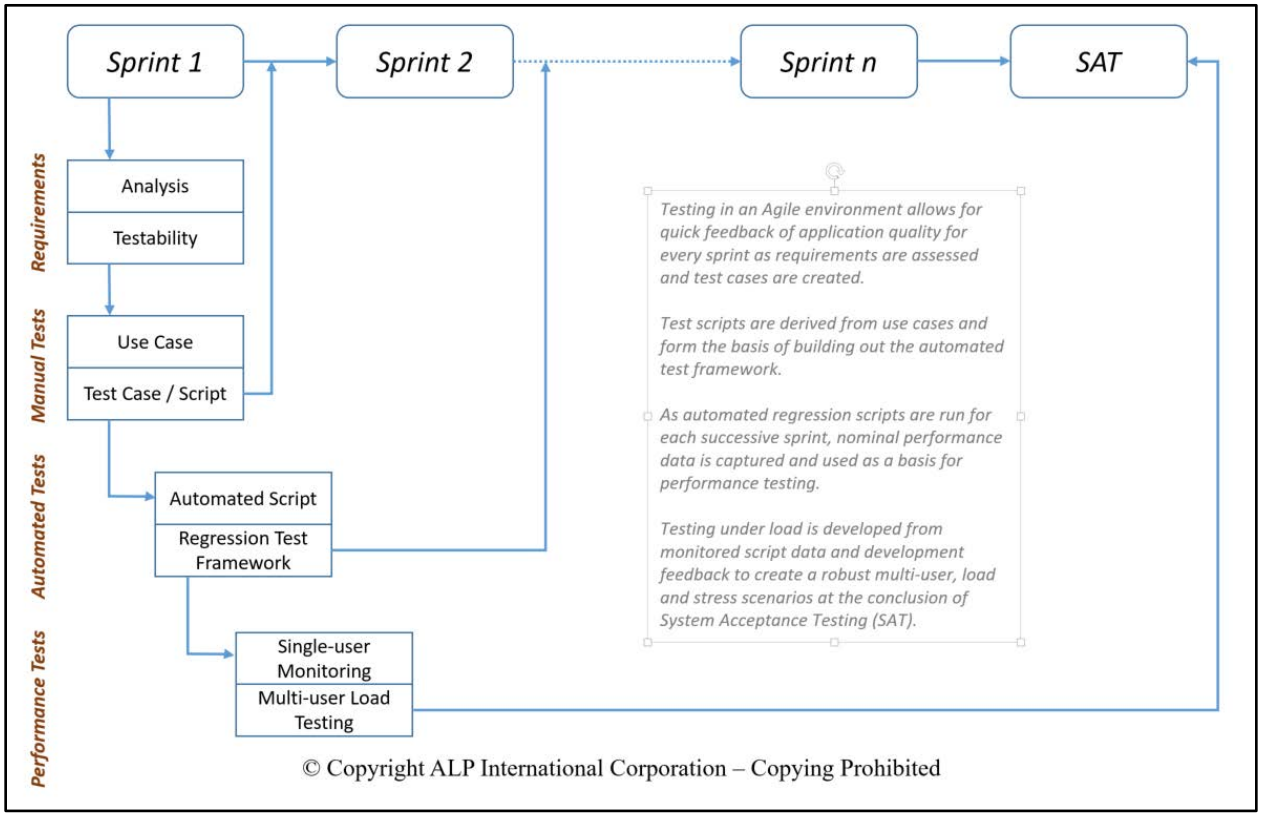

Figure 6.7. Automated Testing across Agile Sprints[14]

6.2.2.1. *Inputs:*

- System Under Test and candidate testing approaches

6.2.2.2. *Deliverables:*

- Alignment to SUT SDLC

Bottom Line: Shape the automation approach to the software and system development model.

6.2.3. Task: Identify Initial Automation Candidates

Selecting the right subset of requirements for automation is imperative for ensuring early success. Evaluate the applications that management deems as critical for test automation. Identify those applications with an architecture that is well understood and for which test tools fit. Do not pick the most challenging candidate first because this approach will delay implementation, show less value, and increase frustration levels along with skepticism. Start with those requirements and test cases that have the greatest chance for success. The targeted candidates should have sufficient visibility for successful automation to be noticed, appreciated, and help build momentum for additional automation.

[14] ALP International (ALPI) Corporation. 2018. http://www.alpi.com/. Accessed: 5-Oct-2018

When you are automating against a particular requirement or use case, you want to see if there are commonality options across the application. For example, automating a user interface (UI) with 5 radio buttons can be used on similar components within the system. Modularity and reusability are essential qualities of an automation program.

Characteristics of requirements that have the highest likelihood of success for automation are:

- Prioritized requirements that have meaningful impact to the user
- Stable requirements
- Well understood tasks
- Repeatable and highly redundant requirements
- Requirements that can be automated in a timely fashion
- Requirements that manual testing cannot adequately address or cover
- Critical requirements that failures would result in significant operational degradation or high consequences

6.2.3.1. *Inputs:*

- Requirements, test cases, capabilities of automated test tools, and manual test characteristics

6.2.3.2. *Deliverables:*

- Recommended application/system for initial automation
- Plan for automation on common components used across SUT

Bottom Line: Automate first the requirements that are well-understood, visible, and have a great chance of a successful automation solution.

6.2.4. Task: Identify Automation Resources Needed

The practitioner needs to work with stakeholders (e.g., business analysts, project owners, developers, etc.) to understand their testing needs and how these translate to automation. While automating existing manual tests is a start, it should not be viewed as the complete approach to planning for automation. There are tests that a manual tester or operations analyst may have not previously done due to complexity of setup and/or execution. These tests can now become automation candidates. Therefore, identifying requirements for automation testing allows for a broader and deeper level of testing requirements than what was possible through manual testing only.

Manual testers will often be helpful in providing feedback on automation as they experience first-hand the drudgery of typing precise commands on a keyboard for hours repeatedly to test a system. This is the “low hanging fruit” as they may have test scripts already in use that would likely benefit from automation. They manual tester will also have insights into various ways in which they can accomplish the same task, be it through shortcut keys, tabbing across screens, or other novel and innovative ways in which they use and test their systems. However, manual testing is time constrained so the practitioner should not be limited by what’s been done manually, but rather use this as a starting point. Also, manual tests that are automated likely will benefit from the refactoring or decomposition/re-composition cycle, and reorganization that automation can bring, making the testing more efficient and effective.

While there might be the temptation to automate testing of the newest functional changes to the application or system, this can cause problems because many other changes in the system might require updates to the automation itself. It is best to start automation against a more stable application area because automation can help ensure this stability continues for each subsequent release. Each functional test that is automated becomes tomorrow's regression test.

Since automation is all about reliable and repeatable testing, the target candidate tests should be those that will be, or should be, repeated regularly. For systems that are only tested once or twice a year, automation may be too brittle a solution. There may be too much development and rework needed with such infrequent testing that ultimately negates any benefit from automation. Additionally, there are qualitative requirements that just do not lend themselves to automation. For example, if a test case requires a tester to subjectively judge whether a system meets an aesthetics requirement of a display, automation is not the best choice.

Understanding how true end-users interact with the system on a regular basis will also help focus the automation effort on those areas likely to bring exposure to the end-user community should the system not be working properly. Finally, the practitioner brings very technical capabilities to the team. They focus on specialized tasks quite different from the manual testers, developers, and operations analysts. The team must function in a collaborative manner to bring all the knowledge necessary to effectively produce automation that is relevant.

6.2.4.1. *Inputs:*

- Stakeholder inputs for automation requirements

6.2.4.2. *Deliverables:*

- Assessment of stakeholder requirements
- Prioritization of stakeholder requirements

Bottom Line: Find those individuals in the organization who can bring insights into the system testing approaches so as to drive excellence into your automation solution.

6.3. Design Phase

Practitioner Checklist for:

Design Phase

- ❑ **Construct Test Automation Architecture**
  - ❑ Understand automation architecture components
  - ❑ Understand supporting management processes
  - ❑ Identify architecture which best aligns with target project
- ❑ **Specify Test Automation Technical Approach**
  - ❑ Identify technical tasks needed to construct a test automation solution
  - ❑ Identify technical skills and roles required for the creation of framework components
- ❑ **Capture Test Steps of Operator's SUT**
  - ❑ Work with the system SMEs or experienced operators to determine test steps, test inputs, and sequences of software and expected responses to inputs
  - ❑ Understand the manual test approach and desired outcomes
  - ❑ Generate a manual test for the purposes of automating and verifying automated test case equivalence
- ❑ **Develop Automation Scripts**
  - ❑ Generate from scratch, borrow like scripts from repositories, or refine previous scripts as necessary to meet the automation needs and intentions
  - ❑ Test and refine scripts
- ❑ **Construct Test Automation Framework**
  - ❑ Develop the necessary connections and linkages to integrate the automation tools, scripts, and libraries
  - ❑ Routinely ensure proper integration as additional tools are incorporated or tool versions updated
- ❑ **Conduct Pilot Project**
  - ❑ As applicable, understand the manual test approach and desired outcomes
  - ❑ With manager, develop the pilot objectives, scope and plan
  - ❑ Review and update the pilot plan
  - ❑ Generate pilot scripts
  - ❑ Perform test automation
  - ❑ Verify pilot results
- ❑ **Establish Test Automation Solution**
  - ❑ Identify all artifacts of a test automation solution
  - ❑ Apply configuration management to all components of solution
  - ❑ Catalog and document solution components
  - ❑ Identify external components necessary for automation solution (user/account roles, test data, database and server access, etc.)

Figure 6.8. Practitioner Checklist for Design Phase

Once a test automation strategy has been clearly defined in the Plan phase (Figure 6.6), the process of creating a purpose-built framework begins, Figure 6.8. Understanding the requirements needed for testing will lead to an automation approach that best aligns with the project. Each component has a

necessary function derived from a requirement in any given automation project. Coordinated by the practitioner, team members with multiple skill levels will build out the automation without delay.

The automated framework is the technical component of reusable functions and libraries that forms the core of the Test Automation Solution (TAS). Look at the target SUT to validate the architecture and know what components are needed to build an automation architecture. Then create the framework components which are the actual artifacts we develop that produce a tangible working automation solution. Once this functionality is developed, the team conducts a pilot project to validate the concept and approach. Development of automation is a project itself with scheduled tasks milestones and roles. There will be specific knowledge, skills, abilities and experience needed that may not be apparent at the outset, but quickly surfaces as a pressing need. Examples include reaching out to database analysts to resolve connectivity issues, or working with developers for specialized APIs.

Through design, development, and piloting of the TAS, we gain valuable knowledge of the adequacy of our design and implementation. This allows us the opportunity to make midcourse corrections prior to full scale development of our automated solution.

6.3.1. Task: Construct Test Automation Architecture

The automation architecture conceptualizes the various tool components needed to create a purpose-built framework that can interact with the SUT at various levels (UI, API, protocol, etc.). Defining a high-level architecture ensures that a consistent approach to automation is being realized and makes it clear what specific components will need to be created. The ISTQB has developed an Advanced Level syllabus for the Test Automation Engineer.[7] In this document a high-level architecture is proposed (Figure 22) that covers a broad range of test automation activities. The generic Test Automation Architecture (GTAA) forms the basis of a recommended set of “layers” that define the necessary functionality across an automation solution. From the GTAA, we develop a purpose-built Test Automation Architecture (TAA) containing the relevant layers and components.

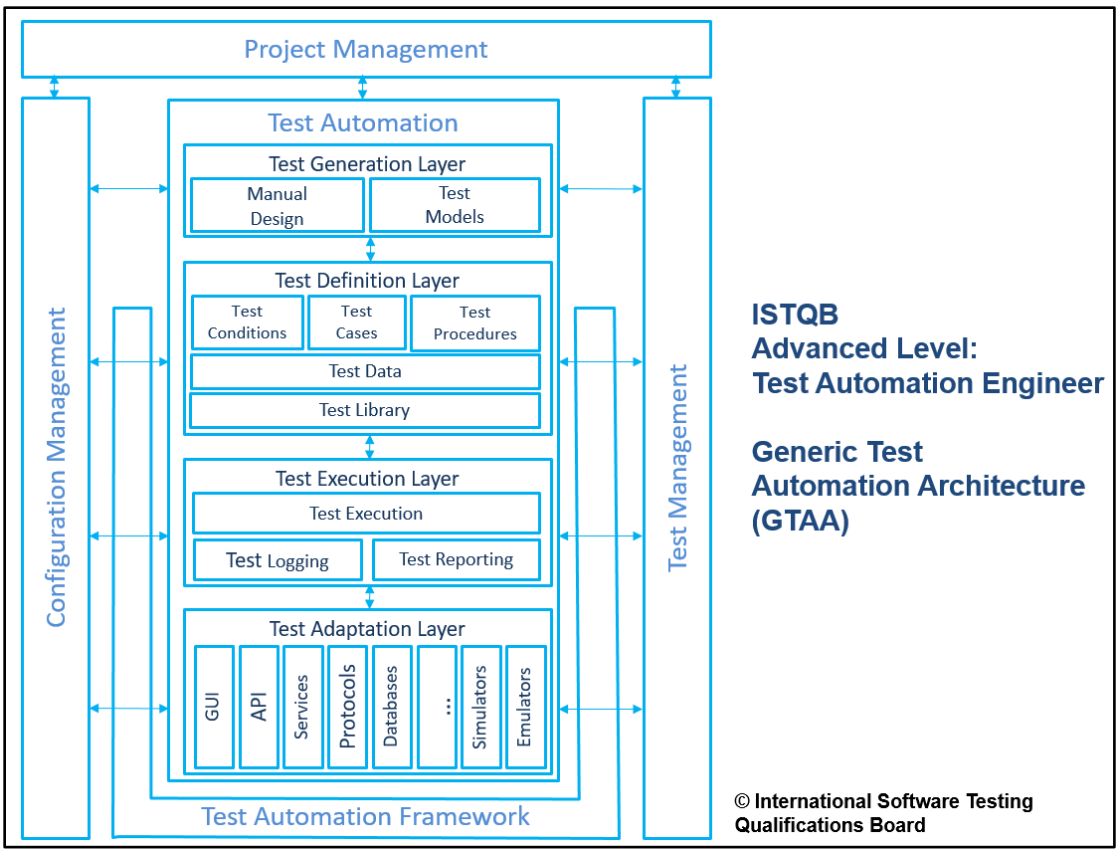

Figure 6.9. The Generic Test Automation Architecture (GTAA)[7]

The defined layers of this model include one each for test generation, test definition, test execution, and test adaptation. This generic architecture, the Test Automation Solution, can be adapted because not all components will be appropriate to the specific program. Each of these layers and the management structure is explained in further detail.

Test Generation Layer

The design of test cases from test requirements is performed in the Test Generation Layer and can be accomplished through manual processes or automation. The use of techniques, such as model based testing, allow us to accomplish this task. There are tools that can facilitate the design of test models and help automatically generate these models to form the basis of our test cases.

Test Definition Layer

The Test Definition Layer defines the code, reusable functions, modular components, and accompanying data requirements at both a high and low specificity level. Test procedures and conditions are also defined individually or as part of a group of tests. Navigation, sequencing, and timing information necessary for the complete definition of test scripts would be included in this layer. Tests at this level will be abstracted from the SUT to be independently maintained.

Test Execution Layer

The Test Execution Layer defines the automation "engine" which will execute test scripts. This will be a commercial or open source test tool that is compatible with the SUT. This does not preclude the organization from developing purpose-built automation execution programs, though this decision should be made after extensively evaluating commercial technology that benefits from regular update cycles. Another component of execution is the reporting function. This can be seen as two distinct components. The logging component provides a record (e.g., audit trail) of what the automation did through each execution cycle step. The reporting component provides information to assess if the SUT performed according to our expectations (i.e., for a given set of inputs it responded with a specified set of outputs).

Test Adaptation Layer

For automation to be compatible, the automated solution needs to interface with the SUT. There are a wide range of application interfaces, as a result of the variety of hardware platforms, operating systems, and software components. The adaptation layer provides the interface to the SUT via the GUI, API, or service, etc. These adaptors need to be selected for their SUT compatibility, or possibly need to be developed.

Test Automation Framework

The Test Automation Solution comprises the above 4 layers previously discussed. The Test Automation Framework may or may not include the Test Definition Layer as this component is usually not part of the overall execution of automated test scripts but rather supports the creation of data necessary to drive automated scripts. The framework allows for better test structuring, more consistent testing, better reusability, integration of non-coders, and improved reliability.

Configuration Management

Test automation components and artifacts should be saved and stored to enable reconstruction of prior versions of the automation environment. Test tools can be installed and configured with many different settings. Each combination will have an impact on the level of compatibility with the SUT. Only the necessary tool components and extensions should be initialized to prevent any inconsistent behavior. Instability can also be introduced as test tools sharing dynamic library links (DLLs) or other system level components may change. Knowing and understanding the underlying system baseline configuration at tool installation ensures the greatest chance for a successful recovery.

Test Management

The Test Management function coordinates the activities of automation including development, maintenance, and test execution. The number of licenses needed by the test automation team, the requirement for ongoing support and maintenance, and the possibility of identifying additional test tools or support libraries needs to be well-thought out in the overall automation solution.

Project Management

Developing a purpose-built automated solution is a project. Thus, the project needs to be managed and tasks and milestones need to be defined. The manager section of this implementation guide provides this type of direction across the automation life cycle.

6.3.1.1. *Inputs:*

- System architecture and functional/reporting requirements

6.3.1.2. *Deliverables:*

- Selected components from architecture required for target automation

Bottom Line: Carefully select the Generic Test Automation Architecture building blocks to maximize the impact of your Test Automation Solution.

6.3.2. Task: Specify Test Automation Technical Approach

An approach to designing a purpose-built Test Automation Architecture from the GTAA will depend on the chosen level of technical depth. Automation projects can be single-person initiatives, or multi-person, multi-role projects where each technical team member is responsible for certain elements of the automation. For example, on large, complex programs, it is not unusual for one engineer to focus on UI elements (naming structure, behavior) and their cataloging. Another engineer may be focused on data input/output functions, while another engineer aggregates data and creates dashboards that serve specific stakeholder needs. In all cases, there will be tasks to design reports, to create audit logs, and to develop reports for leadership consumption.

The practitioner needs to examine which test type the automation is meant to support, e.g., functional testing, performance testing, conformance testing, interoperability testing, etc. Additionally, which test levels does this apply to, e.g., component, integration, system? Identification of technologies used for the system under test will enable the design of appropriate solutions.

For the Test Generation layer, a decision will be made whether test selection occurs manually or is an automated process. The test selection strategy may employ methods such as combinatorial test design or generation. These methods create efficiencies in coverage, though tradeoffs will have to likely be made between depth of test and resource consumption. The data requirements for automated tests also need to be factored into the approach to automation.

The Test Definition layer requires the selection of how data flows through the automation solution. Examples of implementation approaches include data-driven, keyword-driven, pattern-based, model-driven, etc. The notation or template for how the data is represented can include tables, stochastic notation, spreadsheets, and domain-specific test languages. Example approaches to automation include:

Capture/Replay

This technique uses the paradigm of pressing a “record” button during the test and then a “play” button once the test has been recorded for test automation. While the simplicity of this approach is appealing, the reality of complex software systems does not match up to the promise. This technique certainly works well for demonstrations of vendor software test tools, but does not have the rigor to hold up for most software systems.

Data-Driven

In order to make tests reusable, we often vary the parameters used for each test execution. By abstracting the data elements from a test script, we can parameterize them and develop a set of inputs and expected outputs using techniques, such as covering arrays, to maximize the test

coverage. Figure 6.10 depicts activities of the analyst and automation engineer for the workflow to transition from manual to automated test in a data driven environent.

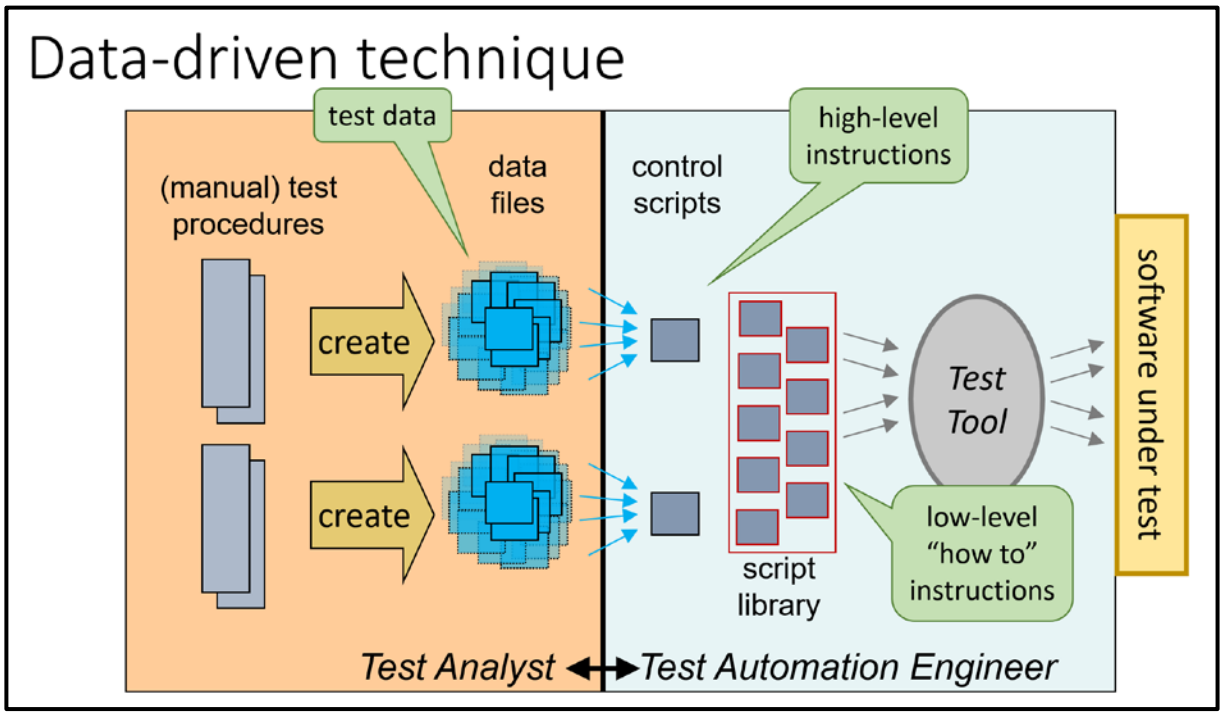

Figure 6.10. Test Analyst and Test Automation Engineer Contributions toward Creating an Automation Solution[14]

Keyword-Driven

The keyword-driven approach is a specialized form of a data-driven test that simplifies definition of test conditions by non-technical testers. Both use data to drive a test. However, in a data-driven test only the raw data is defined as data. In the keyword-driven test, keywords are defined which correspond to automated test functions or procedures. These keywords then drive the execution of specific functionality and can additionally define data as parameters.

Model-Driven

This approach is where the software itself is first modeled and then test cases are created from the model. This technique is more suited for test script and data creation rather than test execution.

Process-Driven

This approach is a data-driven method where all data elements needed for a test are abstracted and defined independently.

The Test Execution layer includes determining what test tool will drive the interaction with the SUT. Will this be executed in a virtual environment? What language can be used to codify test logic? Execution programming language typically is restricted to those supported by the test tool (e.g., C, C++, C#, Java,

Python, Ruby, etc.). Additional runtime execution features may be available via libraries or 3rd party tools.

The Test Adaptation layer defines how we interface with the SUT, typically in the form of the GUI and API. This may include simulating or emulating the SUT and monitoring the SUT during execution.

6.3.2.1. *Inputs:*

- Tester profiles and application testing needs

Deliverables:

- Input convention for selected approach (e.g. keywords for keyword-driven approach)

Bottom Line: Know your test team (level of technical comfort) to appropriately match their skills to testing needs.

6.3.3. Task: Capture Test Steps of Operator's SUT

The test team needs to be familiar with how the SUT is actually used operationally. A SME usually identifies the functions most commonly executed and those functions that are occasionally executed. The automator can then build the interface, or message-based script (GUI versus API) that captures the primary process flow for the mission threads along with related contingency paths.

The team should focus not only on the most frequently executed paths, but also on the highest risk ones in terms the SUT's ability to perform its intended functions. This can sometimes be verified by viewing logs of web servers that capture frequently accessed application pages. The team should also make an assessment of the automation potential for these mission threads—just because they are critical does not mean the technical challenge of implementing automated tests is any easier.

The manual test processes of those expected to be automated must be adequately and sufficiently captured. The test automator will participate with the software test engineers in a series of manual tests that are detailed in a step-by-step procedure. The automator will make note of promising constructs to integrate into the automated solution. If possible, other testers should run the test manually with the automator to check for consistency and equivalence and consider discussing alternative and innovative solutions. Take the opportunity to use the paradigm shift to automation to see if there are additional points of verification or points that can enhance verification steps.

When automating a manual test, we should be open to ideas beyond replicating the exact same step by step process as the manual tester. For instance, the use of covering arrays and other techniques to statistically generate test data will increase test coverage. Practitioners need to create automation in modular manner rather than a single-use linear fashion.

6.3.3.1. *Inputs:*

- Operational use profile and manual test scripts

6.3.3.2. *Deliverables:*

- Automation functional requirements

Bottom Line: Operators and manual testers are an invaluable resource to capture system usage and test. Recognize automation allows you to go much further than just replicating the current state.

6.3.4. Task: Develop Automation Scripts

In the design phase, scripts are developed through an iterative process in highly collaborative settings possibly aided by cloud hosting. There should be detailed code reviews by peers and continued work with SMEs to adequately verify and validate the Test Automation Solution. Automators involved in scripting should have training and experience in the programming language native to automation tools to maximize tool effectiveness and coding proficiency.

Figure 6.11 shows an example of automation code in Hewlett Packard's Quick Test Pro for a simple function using the calculator in Windows. Some attributes practitioners should keep in mind for script development are:

- Build reusable functions that can be shared where feasible
- Easily shared scripts are more generic and contain application specific information embedded
- Scripts with a common purpose (e.g. navigation, authentication) can be part of a well-organized library
- Scripting should use programming best practices
- Be attentive to naming conventions, code nesting, and code complexity errors
- Document along the way to facilitate transitioning the codes with turnover and help promote the use for other projects and programs.

There may be considerable refinement necessary for scripts that are "borrowed" from other users or a repository. They may have seemed to be plug-and-play, but rarely will these products require no modification. One common "borrowed" script is for user authentication – a Common Access Card (CAC), for example. This script ordinarily would require the tester to be present; however, simple scripts using open source tools (such as AutoIT) can be developed or acquired to automate this activity.

```
For i = 1 to 10000
    'VERIFY + AND * OPERATIONS ON THE CALCULATOR
    Window("Calculator").WinEdit("Edit").Set(i)
    Window("Calculator").WinButton("+").Click
    Window("Calculator").WinEdit("Edit").Set(i)
    Window("Calculator").WinButton("=").Click
    intResult_1 =   Window("Calculator").WinEdit("Edit").GetROProperty("text")

    Window("Calculator").WinEdit("Edit").Set(i)
    Window("Calculator").WinButton("*").Click
    Window("Calculator").WinEdit("Edit").Set("2")
    Window("Calculator").WinButton("=").Click
    intResult_2 =   Window("Calculator").WinEdit("Edit").GetROProperty("text")

    If intResult_1 <> intResult_2 Then
        Reporter.ReportEvent micFail, "VERIFY", "RESULT INCONSISTENT FOR DATA :" &i
    End If
Next
```

Figure 6.11. Test Tools Can Capture an Operator's Steps to convert into an Automated Script[15]

[15] https://commons.wikimedia.org/wiki/File:HP_Quick_Test_Professional_VBScript_Code.png. Accessed: 5-Oct-2018

6.3.4.1. *Inputs:*

- Automation functional requirement from operational profiles

6.3.4.2. *Deliverables:*

- Modular reusable scripts and functions

Bottom Line: Invest time to write scripts that are well-documented and are reusable.

6.3.5. Task: Construct Test Automation Framework

The Test Automation Framework (TAF) develops the necessary connections and linkages to integrate the various automation tools and scripts. The selected tools are part of the framework. A TAF is built using existing script, the script libraries, configuration files, API calls with initial base of functionality that allows for additional features and components for future growth. A module whose purpose is to interface with another system can be called by scripts requiring connectivity to that external system. The framework functionality shall include being able to read data from databases, drive data through application, capture verification, create output logs and create reports.

As the automated solution and environment matures, often additional tools are needed to achieve desired automation features. This is especially evident as more and more output data are generated and test teams are faced with challenges of how to use it. The TAF should have a structure so that as new controls/object types are added to the SUT, they can easily integrate into existing libraries without having to affect existing code.

The team needs to test communication between tools using prototype or simple test scripts. Routinely ensure proper connectivity as additional tools are incorporated or tool versions updated. Engage the software developers/coders if there are compatibility issues or ineffective linking and communication among software tools. For example, make sure the open source tools work effectively with commercial parent tools.

The TAF supports a consistent approach to conducting test through documentation and maintainability. This establishes the need to implement reporting facilities with comprehensive logging of automation. The audience for these reports is the tester, test manager, and the developer. The TAF needs to provide for easy troubleshooting to support root cause analysis in both the SUT and automation solution. The TAF should be developed as a consistent solution in a dedicated environment focusing on:

- Reliability and maintainability
- Usability, extensibility and scalability
- Flexibility to easily make changes and updates

6.3.5.1. *Inputs:*

- Scripts, libraries, reporting requirements, tools, test environment

6.3.5.2. *Deliverables:*

- Automation environment with library of scripts and functions

Bottom Line: Build a Test Automation Framework that produces a usable, reliable, maintainable, and expandable automation environment.

6.3.6. Task: Conduct Pilot Project

Once the TAF is in place, the team can conduct a limited scope pilot project on selected high-value requirements. Use subject matter experts, review manual tests, and watch operators to best choose automatable requirements. The most important part of the pilot is working with the manager to effectively plan the pilot with attainable objectives. The goal is to find the right subset of the system's functionality that you want to evaluate the framework against. The pilot should be conducted outside of other testing schedules as an independent activity and closely aligned to the Test Automation Framework.

While conducting the pilot, make sure you have all the right data, environment, and scripts to run automation. Ensure things are working right constantly in a debugging mode as care should be taken to fully understand what automation is doing at a low level of granularity. The team should spend considerable time in validating the framework functionality works as expected.

The analysis of the pilot test results informs future efforts. These projects vary in complexity from the next iteration of the pilot to the full-scale automation effort. The team should compare time estimates to actuals, evaluate technical issues and figure out how to address them for future implementation. Evaluate the shortcomings in the feature set that need to be developed and implemented. Make sure to get a sense of execution times and impacts to future testing as you think about migrating other tests to automation.

Be prepared to fail at times in the pilot. The goal is to learn as much about automation as possible, which will require experimentation with some false starts and wrong turns likely. These should all be controllable setbacks rather than debilitating, which would indicate the automation was too complex.

6.3.6.1. *Inputs:*

- Test scripts including input and verification data, the framework, automation environment

6.3.6.2. *Deliverables:*

- Proven automation capability and framework confirmation

Bottom Line: A challenging but manageable pilot project will set you on the path for excellence in automation.

6.3.7. Task: Establish Test Automation Solution

Following a successful pilot with a proven framework, we now have components that comprise the TAS. This includes the underlying automation architecture, the purpose-built framework, and all additional necessary components and artifacts required to implement and sustain a test automation solution. TAS includes the test data, possibly some developed with covering arrays and other STAT methods that you drive through the framework environment, scripts, and test design. In order to develop a TAS, the practitioner needs to work closely with the developers and understand the underlying technology used by the SUT. The team needs to identify external components necessary for the automation solution such

as user/account roles, test data, database and server access. They may need to make updates and corrections after the pilot for the initial version of the TAS. As development continues, additional functionality is added to the TAS.

The solution is controlled by detailed configuration management and sound documentation practices. Project management and test management need to be supporting and guiding these activities. At version 1.0, understand that we are running in a specified environment with a particular version of automation software tools, using locked-down versions of code, libraries, DLLs of API calls, data files, and so on. Know that this is a growing environment that will be adapted to meet future needs.

6.3.7.1. *Inputs:*

- Revised test automation framework from the pilot project execution

6.3.7.2. *Deliverables:*

- Documented and catalogued test automation solution

Bottom Line: In creating the Test Automation Solution, you have established automation liftoff!

6.4. Test Phase

Practitioner Checklist for:

Test Phase

- ❑ **Verify Test Automation Solution is Working**
  - ❑ Ensure automation is executing correctly before executing SUT tests
  - ❑ Identify a set of tests that can be run to verify automation functions and libraries are working properly
  - ❑ Update automation verification tests when new functionality is added to automation solution
- ❑ **Verify Automated Tests Against SUT**
  - ❑ Develop a set of baseline automated tests that have known working results against target SUT
  - ❑ Ensure baseline automated tests are equivalent to manual tests
  - ❑ Use baseline tests to help understand any irregularities when testing against SUT
  - ❑ Determine if baseline tests need to be updated based on changes to SUT
- ❑ **Consider and Decide on Test Oracles**
  - ❑ Carefully consider the process of determining whether a test passes or fails, the test oracle problem
  - ❑ Assess automated methods to solving the test oracle problem to include fuzz testing, metamorphic testing, match testing and pseudo-exhaustive testing
- ❑ **Execute Automation**
  - ❑ Run automation scripts
  - ❑ Review logs and other output data products for anomalies in framework code and TAS
  - ❑ Enter software failure information into the tracking system
- ❑ **Clean Up Test Automation**
  - ❑ Reset data to initial conditions
  - ❑ Update scripts and libraries to new versions

Figure 6.12. Practitioner Checklist for Test Phase

Test automation is equally susceptible to coding errors as any other software development. Therefore, ensure the automation solution is functioning as intended in the Test phase, Figure 6.12. This will include verifying the automation on its own and in conjunction with the SUT.

Anytime we run our automation solution, whether to test against the SUT or to validate an update to the TAS, we have pre- and post-execution conditions to meet to assure a normalized system. In doing so, we gain confidence that our solution is working. If there are error conditions, those need to be quickly flagged with detailed diagnostics for rapid root cause analysis and corrective action.

6.4.1. Task: Verify Test Automation Solution is Working

Once subsequent iterations of the TAS have been developed, they will need to be tested. This testing includes the various components that make up the framework to ensure they work reliably. A TAS may

include connectivity to external systems. After confirming that the external system/interface is operational, connectivity tests should be executed to validate the operation of the TAS. For example, a framework may include components to import file data, export file data, verify control attributes, and synchronize objects. A set of scripts can be constructed to exercise the critical functionality with a known data set. This can help prevent false negatives, where a verification step did not show a mismatch when it should have. Additionally, what components need to be installed, in what sequence, and in what folders should be documented to replicate the process.

Test tools may require updates to system files or DLLs, from vendor updates or from changes to the underlying operating system because of service packs or other changes. The team must continuously check compatibility with the environment and SUT to ensure that the test automation solution continues to perform as expected.

6.4.1.1. *Inputs:*

- Test scripts and data from additional requirements

6.4.1.2. *Deliverables:*

- Updated Test Automation Solution

Bottom Line: New requirements create the need to update and retest the TAS.

6.4.2. Task: Verify Automated Tests Against SUT

Once the team establishes the components that make up the framework are working properly, they conduct a set of baseline tests known to work against the SUT and have been proven to be equivalent to manual tests and/or operational profiles. This allows the tests to become reference points should inconsistencies be found in running automated tests. We can use this approach when the behavior of the current SUT does not match previous behavior to rule out that the TAS may be a contributing factor. This situation may require updating the TAS from confirmed SUT changes. The team should also verify the TAS using tests with known passes and failures, or at a minimum, known passes (i.e., happy path testing).

As additional tests are automated for a given SUT, the catalog of verification tests needs to be expanded to account for TAS changes to test the additional capability.

6.4.2.1. *Inputs:*

- Set of SUT baseline tests

6.4.2.2. *Deliverables:*

- Validation that the Test Automation Solution has been updated and is performing correctly

Bottom Line: Use baseline SUT scripts to confirm possible updates to the TAS.

6.4.3. Task: Consider and Decide on Test Oracles

The test oracle is the process of determining whether a test has passed or failed. Even with automated methods of generating input data and running tests, the oracle problem remains. Testing requires both test data and results that should be expected for each data input. This is generally the costliest part of the

testing effort, since extensive human involvement is needed in conventional approaches. However, a variety of methods are available to automate some or all of the test oracle generation. These vary in initial cost, level of sophistication, and domains of application.

One critical factor in planning and executing automated testing is whether the automation will be used to generate complete tests, input data and expected results, or input data only. Most of the methods to automate the oracle rely on formal models in some form, or assertions embedded in code which serve effectively as a partial formal model. Other options include partial correctness approaches such as metamorphic testing and other "sanity checks." In some cases, fuzz testing will be a useful preliminary step, and this should be included in planning discussion as well. References include Barr et al (2015), Bartholomew (2013), Kuhn and Okun (2006). Appendix E details example test oracles further.

6.4.3.1. *Inputs:*

- Input data, expected results, and complete tests

6.4.3.2. *Deliverables:*

- Report indicating pass or fail

Bottom Line: Automated test oracles improve test verification.

6.5. Task: Execute Automation

This execution task refers to the actual process of running the Test Automation Solution (TAS). The goal is to execute the TAS without test team intervention either off hours or in the background, not interfering with other test activities. There may still be a need to monitor the TAS logs as errors may occur in the SUT stopping the automation early, or there may be errors in the automation framework. For example, some GUI record and play tools get “hung up” looking for the correct image in order to proceed to a subsequent step. Error and event trapping can help in recovery in scripts and allow them to continue executing. Be aware of automation that requires interim manual inputs; here, the testing is not fully autonomous.

Prior to running automation, ensure all external interfaces and data collection systems are operational. The TAS should automatically log and report errors. Either automation or the team may have to enter software failure information into the tracking system to help in root cause analysis and system diagnostics. Tying automation execution into system and network resource monitoring will help to correlate system behavior to utilization.

The team should look for opportunities to correct or improve the automation. Identify the faults that stop the automation process and try to make the framework as robust as possible. Promptly correct scripts that execute incorrect logic and those scripts that fail to fully execute the intended functions.

The team likely will need to run test contingencies such as scaling for many more users, applying tests in different environments, and using different hardware and operating systems. For example, an automated test (GUI-based) ran fine in the SIL; but, when attempted at another location, it failed because the laptop used was older with lower screen resolution. Another important contingency is testing when the SUT is running in a degraded state. These contingencies may necessitate changes to the TAS. The benefit is that the enhancements make the automation more resilient and often will reduce the maintenance burden.

6.5.1.1. *Inputs:*

- Test Automation Solution

6.5.1.2. *Deliverables:*

- Understanding of problems that prevent the Test Automation Solution from running error-free

Bottom Line: Understand what activities and investigations are required for the TAS to run error-free.

6.5.2. Task: Clean Up Test Automation

At the end of test execution, a number of recovery activities are required. The team needs to account for post-testing activities such as archiving the code base and data used in the automated test run. They will need to perform any re-initialization activities to prepare the automation for subsequent runs. This should include resetting data input parameters, clearing out log files, and rolling back the SUT database populated from prior tests. As an example, a SUT may not take as input a value that was previously entered as unique unless the database is cleared of that value. The team will need to update scripts and libraries to new versions to prepare for future testing.

6.5.2.1. *Inputs:*

- Artifacts from automated test execution

6.5.2.2. *Deliverables:*

- Reinitialized and updated Test Automation Solution

Bottom Line: Tidy up after each run—your subsequent runs will be much smoother.

6.6. Analyze and Report Phase

Practitioner Checklist for:
Analyze and Report Phase

- ❑ **Analyze Output Data Artifacts**
  - ❑ Information on which test is currently executing, when it started and ended
  - ❑ Data used for test script execution
  - ❑ Detailed runtime data including script steps, timing data, screen shots
  - ❑ Status of script execution in order to assess normal execution, aborted execution, normal termination
- ❑ **Develop Test Automation Reports**
  - ❑ Customize reports to the needs of the audience
  - ❑ Determine what artifacts are needed to support various levels of reporting granularity
- ❑ **Determine Failure Causes**
  - ❑ Develop and institute an approach for tracking software defects
  - ❑ Determine whether a closed-loop corrective action system is appropriate and if applicable, implement the system
- ❑ **Develop and Quantify Metrics and Measures**
  - ❑ Consider and apply relevant metrics from a list of possible qualities related to the level of effort required to automate.
  - ❑ Identify both short term and long term benefits and savings for time and resources due to automation.

Figure 6.13. Practitioner Checklist for Analyze and Report Phase

The TAS needs to provide clear status indicators for every run for both the SUT performance and the automation architecture. Fortunately, we have engineered various output logs and files into the solution for this very purpose. Analysis, Figure 6.13, may require correlation of results to other software or system resources to further investigate diagnostics to determine root cause. Understanding what information is available and where it is stored enhances the defect isolation, root cause analysis, and corrective action cycle. Once we've analyzed data to understand behavior, we can develop reports that aggregate metrics so we have a clear indication of our current condition. Customized versions of these reports can be distributed to the various groups of stakeholders to help inform their special decision making processes.

6.6.1. Task: Analyze Output Data Artifacts

The TAS must include a robust logging function to provide clues at runtime of SUT or TAS errors. TAS logging should include:

- Information on which test is currently executing, when it started and ended
- Data used for test script execution
- Detailed runtime data including script steps, timing & synchronization data, and screen shots

- Status of script execution to assess normal execution, aborted execution, and normal termination
- All data captured prior to point of failure

When analyzing the log files, look for:

- Embedded error messages with diagnostic forensics
- Information on test execution time individually and across all tests
- Information on which test is currently executing, when it started and ended
- Data errors in input or output streams
- Validation through screen shots of error conditions
- Test tool run time messages

As much as possible, trace TAS log file errors to associated errors in the system under test logs, for example, web server and database logs. This set of errors should in turn be traceable to system and network behavior and log files.

6.6.1.1. *Inputs:*

- Log files

6.6.1.2. *Deliverables:*

- Analysis and conclusions to feed reporting requirements

Bottom Line: Log files are the first place to look to understand aberrant behavior and inform subsequent actions and reporting.

6.6.2. Task: Develop Test Automation Reports

Test automation reporting can and should be customized for the relevant stakeholder. This requires capturing data through TAS functionality or via external reporting tools (e.g., spreadsheets, report aggregation tools, dashboards, etc.). The key is that if the data collected is captured at the lowest level of granularity, it will be able to support any level of reporting requirements. Figure 6.14, below, and Figure 5.18 show different reporting approaches from automated testing to satisfy stakeholder needs.

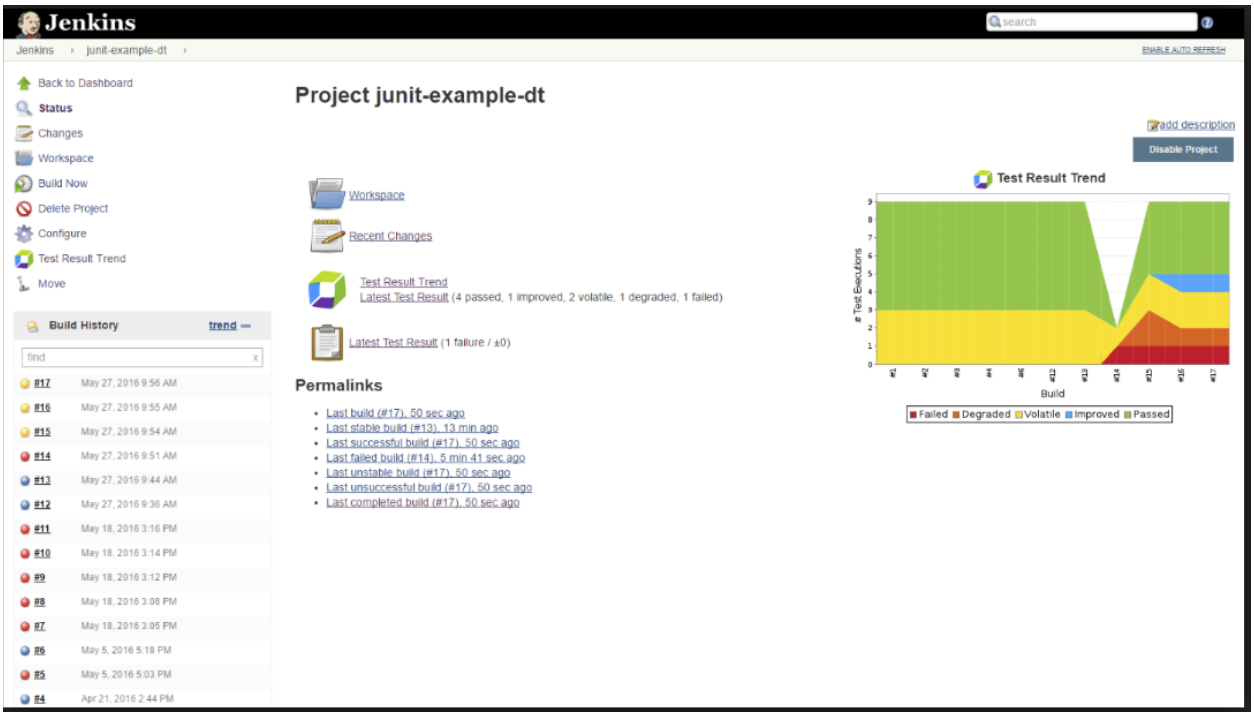

Figure 6.14. Example Test Reports Rolled-up From Test Logs and other AST Data Output[16]

The type of data, information, and reporting is a function of the stakeholder's role. A test automation engineer will need reports that show how the automation behaved when interacting with the SUT. A business analyst may want to see how the SUT performed during an automated test run. A test manager may be concerned with the overall time to execute automation and how many of the tests passed and failed. The test director will likely be interested in seeing trends in application quality.

Unfortunately, this task requires more effort than just assembling disparate data sources. The team will need to spend time analyzing each raw data product to determine how it impacts test. Additional tools may be needed. The operations analyst will be supporting the other team members as they collect, clean, and analyze data to search for impactful insights. These insights should be rolled up into concise reports, dashboards, and other products to help inform leadership of the current automation and SUT status.

6.6.2.1. *Inputs:*

- Reporting requirements and test execution artifacts

6.6.2.2. *Deliverables:*

- Tailored reports, dashboards, and analytical results

Bottom Line: Do not rely on simple tool-generated reports alone, because you will need to transform the raw output into actionable information.

[16] https://plugins.jenkins.io/dynatrace-dashboard. Accessed: 5-Oct-2018

6.6.3. Task: Determine Failure Causes

Failures may be attributed to the SUT, the TAS, or to a data error. The log files are the essential forensic evidence to help isolate the root cause of the failure. A well-designed TAS will go a long way to rapidly isolate the fault and help identify a path for corrective action. If deeper analyses are required, the team will need to use other resources such as interviewing the SUT engineers, automation architects, database analysts, and other domain experts. It may be helpful to use methods and tools such as Cause and Effect (Ishikawa) Diagrams, the 5 Whys, and the 8 Disciplines (http://asq.org/learn-about-quality/eight-disciplines-8d/).

A methodical approach to tracking software defects is also necessary to fully realize the benefits of software testing. A closed-loop database system, such as Failure Reporting, Analysis, and Corrective Action System (FRACAS), provides excellent visibility and accountability. Some of its database fields include conditions where failure occurred, time/date, system status, initial corrective action, point of contact to own failure, and current status. Many of the defects will be subject to Failure Review Board (FRB) actions, and the FRACAS database is set up to provide required background information and record FRB recommendations. The team should institute a process to update and review the FRACAS database as part of the regular battle rhythm.

6.6.3.1. *Inputs:*

- Test logs and output data

6.6.3.2. *Deliverables:*

- Entry in failure database and identification of root cause

Bottom Line: Isolate failure cause through your detailed logs and enter comprehensive failure data into a software defect tracking system.

6.6.4. Task: Develop and Quantify Metrics and Measures

We are also interested in metrics that provide measures of quality, efficiency, and cost effectiveness for the TAS. The kind of metrics allow us to gain better understanding for how automation is performing and offers a separate focus to the metrics that we use for evaluating the performance and quality of the SUT.

With test automation metrics, we seek to understand efficiency and effectiveness of the implementation and impacts resulting from changes to the TAS. These could be the result of a new feature implementation, a new execution engine, or a new underlying hardware and software platform. Metrics can include:

ROI of test automation

This ROI can be quantified in terms of a financial investment of people, software, and hardware (covered in more detail in the management section) and the quantifiable benefits that automation brings. This often is expressed in reduction of time to execute tests, reduction of effort to complete a test cycle, freed up resources, and increased test execution frequency. More specifically, additional tests can now be executed within the same timeframe, which can provide greater test coverage.

Level of effort to build automation

Like most software projects, automation has a full software lifecycle that tends to be front-loaded. Depending on the complexity of the SUT, it can take from weeks to months until initial capability can be demonstrated. Having the right technical resources with relevant experience, training, and certification, can ensure that schedules are met and milestones are achieved. The effort expended to build, execute, and maintain automation needs to be compared to the equivalent manual test effort (EMTE).

Level of effort to analyze SUT errors

This effort can often be more difficult to analyze than that of a manual tester due to the added complexity of analyzing the TAS components that may have contributed to the error. This effort can be reduced by using an overall automation architecture that makes debugging easier through the use of modular components and documented scripts. Adding good reporting capability (as described in the Section 6.6.1 Task: Analyze Output Data Artifacts above) provides data points that can be helpful in identifying the error's root cause.

Level of effort to maintain automation

Automation must evolve with changes to the SUT and updating of scripts and underlying components. Many organizations underestimate the effort involved in automation maintenance and eventually the automation ceases to work leading testers back to a manual process. Tracking the effort required to update the TAS by SUT release provides a measure of resources required for subsequent releases.

Execution time of automated tests

Measuring the time to execute an automated test is simple to do and often is already part of the logging function of a test automation execution engine. If not, this can be added as a function to the component of the TAS that provides log reporting. As regression test beds become larger, this metric will have increased significance.

Overall number of automated tests

The number of tests that are automated will tend to increase over time as the current functional test (which may be executed manually) becomes a part of the overall regression test bed. Comparing the number of automated tests to the overall possible number of automated tests provides insight into the potential payoff from automation. Programs that have implemented a TAS that have only a few automated tests often are not using automation in the most efficient manner because of the significant upfront costs needed to staff and maintain a TAS.

Ratio of automated to manual tests

For programs with large amounts of manual tests, we expect to see a high rate of conversion to automation. This automated-to-manual conversion process should be planned, estimated, and scheduled. As the conversion process progresses, the number of manual tests being executed will naturally diminish (and the ratio of automated to manual tests grow) until only manual tests not being converted remain.

Performance of automation components

The TAS has several components that are used frequently. These components include functions that support read/write, perform object interrogation, perform logging/reporting, etc. These functions can sometimes be called hundreds or thousands of times during test execution, so knowing the relative performance of these frequently-used individual functions is key to optimizing the overall speed and efficiency of the TAS. The net effect of such optimization is to have the same automated test scripts run faster but with no loss of reliability.

Automated test code quality

Purpose-built automated test solutions often are customized via programming languages. This program code can be susceptible to poor coding techniques, including excessive nesting *(identified through cyclomatic complexity, etc.)*, coding errors, and inconsistent variable naming and documentation standards. This condition of the code will affect the ability to quickly identify root causes of errors and to quickly make maintenance updates to the TAS.

Coverage of SUT code

Although not exclusive to automated testing, understanding how much of the SUT's underlying code is exercised during automated test runs helps provide a measure of test coverage. As additional manual tests are automated, we would expect to see additional coverage. When there is little change in coverage, it indicates that there is likely to be some duplication in the manual code that was automated and calls for a refactoring of the automated test cases.

6.6.4.1. *Inputs:*

- Measurement data collected from development, execution, and maintenance of automation

6.6.4.2. *Deliverables:*

- Quality metrics that drive actionable steps to improve automation

Bottom Line: Compute and track metrics that matter to improve automated testing and system quality, demonstrate the value of automation, and help estimate schedules for maintenance and new automation.

6.7. Maintain and Improve Phase

Practitioner Checklist for:
Maintain and Improve Phase

- ❑ **Transition Automation**
  - ❑ Ensure that automated test solution is equivalent to existing manual testing so that it can be relied upon for future testing
  - ❑ Identify areas where automation will change current testing practices (e.g. by replacing, expanding, or initiating testing activity)
- ❑ **Manage the Test Automation Solution (TAS)**
  - ❑ Intentionally review and revise the TAS as appropriate as the SUT evolves over time
  - ❑ Measure the performance and robustness of the TAS components to changes in the SUT
- ❑ **Update Automation Code**
  - ❑ Schedule time for minor and major updates for automation code revisions
  - ❑ Revise the automation code based on changes to versions of the test tools, the GUI, operating system, or the SUT
- ❑ **Manage and Optimize Scripts**
  - ❑ Create, populate, and manage a script repository
  - ❑ Research and investigate script repository libraries for useful scripts
- ❑ **Identify Alternative Execution Technologies**
  - ❑ Evaluate new tools versions or new products that can serve as the underlying program executable
  - ❑ Consider commercial and open source alternatives

Figure 6.15. Practitioner Checklist for Maintain and Improve Phase

Automation must adapt to the changing SUT and test environment to continue to yield a significant ROI. Eventually, automation should not only replace the appropriate manual tests but also vastly improve the overall test program, Figure 6.15. Consequently, manual testers will have more time to conduct exploratory testing, and automation will run deeper and more efficient tests. With computers now helping to do the work, we are able to more clearly see the full landscape and impact of testing possibilities.

The maintainability of your TAS begins with the Design Phase step, (see Figure 6.8) which ensures all the necessary “ingredients” for maintenance are “baked in” to the Test Automation Solution. Planning for end-to-end automation success at the start is critical because there will be multiple systems, staff, technologies, and challenges involved in the automation and the maintenance function.

Technology changes often, therefore, the automation solution must adapt or will soon become inoperable. Beyond looking inside to our work, we need to look outside to leverage industries’ new, improved, and innovative solutions to fend off obsolescence.

6.7.1. Task: Transition Automation

The automated test process must be an integral part of the overarching test process to provide the best chance for success. Automated tests need to complement or replace existing testing capabilities, or create new testing capabilities. These capability changes all still need to be coordinated to run and report within the established software testing cycle. Any preparatory automation work, which is altogether different from the manual test activities, needs to be accounted for and scheduled in advanced of the test event. It is important to clearly demonstrate to operations analysts, manual testers, and developers that the converted automated tests are equivalent to their manual versions. Working with these stakeholders will go a long way towards finding common ground and getting consensus on the benefit of automation on the program.

Additionally, the practitioner must account for any post-testing activities. Examples include archiving the code base and data used in the automated test run and any re-initialization activities to prepare automation for subsequent runs.

6.7.1.1. *Inputs:*

- Manual test candidates where automation equivalence is sought and identification of other test processes within the lifecycle

6.7.1.2. *Deliverables:*

- Verification of automation capability that replaces existing testing and integration points for automation in testing lifecycle

Bottom Line: Verify your automation not only achieves performance equivalent to that of manual tests but also measurably improves the overall test program metrics.

6.7.2. Task: Manage the Test Automation Solution (TAS)

The TAS is built from components in an architecture satisfying the requirements for the SUT. Over time the SUT evolves and so too should the TAS. Try to find ways to improve and fine-tune the efficiency and effectiveness of these components by measuring the performance and robustness of the TAS. These components may include specific functions and libraries that have been purpose-built to support the TAS, or third-party utilities and test tools integrated into the TAS.

The TAS needs to be modular to be easily understood and maintained. Modularity allows for changes to be made in one area without the possibility far reaching effects. A module is a self-contained component of a system with a well-defined interface to the other components. Figure 6.16 illustrates how modules can be easily extended or replaced. As an example of modularity, a change to a data Input/Output (I/O) library will potentially affect all tests that call that function library. For stability purposes, application software components are not always set up to automatically update, given the havoc this could cause on a functioning system. However, software updates on components should be evaluated on a regular basis as improvements might make them more reliable, resilient, and less susceptible to security issues. For these reasons, changes made to a TAS need to be regression tested against a known set of baseline scripts whose purpose it is to validate the proper operation of the TAS.

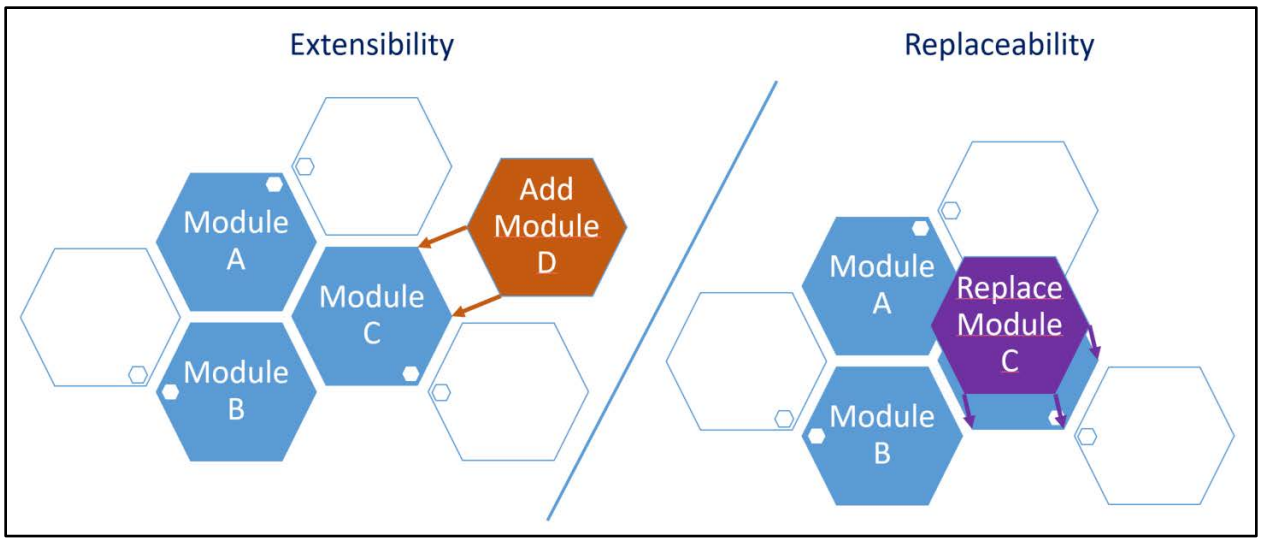

Figure 6.16. A Modular Test Automation Solution Facilitates Updates/Replacements to Key Components

6.7.2.1. *Inputs:*

- Requirements that cause updates or changes be made to the TAS

6.7.2.2. *Deliverables:*

- Fully functioning, re-tested TAS

Bottom Line: Keep the TAS current and verified because modifications are inevitable due to changes in the SUT, the environment, and the test tools themselves.

6.7.3. Task: Update Automation Code

Do not underestimate the effort required to update the automation code to be responsive to changing SUT and interfaces. The team needs to be aware of any changes that would impact execution. The code should also be updated to account for new tools and updates to existing tools. The code may need to be modified to improve coverage, diagnostics, execution speed, readability, and other metrics.

The design of the code and scripts should be nimble enough to easily adapt to frequent changes. In-line documentation of the code can facilitate these changes. Often, someone other than the original automation developer may need to maintain it. Maintaining another person's code is challenging particularly if it is not clearly documented, well structured, and using appropriate naming conventions. This "design for maintainability" mentality helps meet the ever-changing needs to adapt our code to allow the TAS to be synchronized with the technology platform shared with the SUT.

6.7.3.1. *Inputs:*

- Code modules requiring updating

6.7.3.2. *Deliverables:*

- Updated and documentation, all adhering to best practices in software development

Bottom Line: Design maintainability into your automation code because you will be continuously updating it as the system, TAS, and technologies mature.

6.7.4. Task: Manage and Optimize Scripts

Each program should create its own script repository, which will enable accessing and updating scripts and tracking versions via the configuration control system. The team can share scripts internally with other developers/automators or externally with other organizations which may be able to reuse them for efficient automation. The script repository should also include scripts acquired from research efforts and other organizations. Scripts can be cataloged for ease of adaptability to new testing needs.

Over time the number of automated test scripts will increase to satisfy functional and regression testing requirements. Executing one test script may duplicate testing performed by other test scripts. This duplication produces inefficiencies and increases total test execution time. Tests need to be analyzed so that duplication is reduced. Additionally, automated tests which were converted from manual tests need to be re-evaluated because a more efficient process is likely to yield better results. Tests which are very large may benefit from decomposition so that a failure does not halt an entire process test. Alternatively, small, quicker tests could be combined into a suite to more easily manage their execution and reporting.

6.7.4.1. *Inputs:*

- Existing automated scripts

6.7.4.2. *Deliverables:*

- Optimized automated scripts

Bottom Line: Regularly catalog and review your scripts to see if breaking them apart or consolidating them will yield greater automation efficiencies.

6.7.5. Task: Identify Alternative Execution Technologies

The TAS relies on an underlying executable program which runs automated tests. The automation framework is built on this executable. With the ongoing advances in technology, it may be beneficial to evaluate new versions or new products for the underlying program executable. Once identified, the framework components need to be adapted to use the improved executable. A good example of this technology evolution has been browser-based test automation. In years past there were few commercial vendors that supported testing from a browser. Over time vendors adapted their existing programs to work with browsers—some better than others. Additionally, communities of the open source software movement began to develop browser-only testing tools. Thus, the market for browser-based testing has many offerings of varying purpose and capability.

6.7.5.1. *Inputs:*

- TAS executable

6.7.5.2. *Deliverables:*

- TAS executable with current technology

Bottom Line: Stay on top of the rapidly evolving automation technology market to optimally execute your automated tests.

7. Summary

Automating software testing has gained significant momentum across the Department of Defense. Because the speed, quality, and cost of testing can be dramatically improved via automation, confidence has grown in the quality of our systems. Many organizations are at various levels of taking advantage of the innovative technologies that transition the often tedious and time-consuming manual testing to an automated solution. These manual testers may fear they are being replaced, but there are never enough resources for testing so they can spend more time in exploratory testing to help improve the overall test program or perhaps consider upgrading their skills to become automators or software developers. The hope is that leadership, managers, and practitioners will all embrace the challenges and rewards of automated software test as they learn more about the technologies.

This AST Implementation Guide provides the tactics, techniques, and procedures (TTPs) across the automation lifecycle phases: Assess, Plan, Design, Test, Analyze & Report, and Maintain & Improve. It is purposely divided into the manager and practitioner roles as each have different functions to ensure automation success. The manager, though focused on the cost, schedule, and performance of the automation, must first lead the team as the principal automation advocate. Managers must properly obtain, train, and motivate qualified staff as well as acquire appropriate technology resources such as software automation tools. On the other hand, practitioners are busy understanding the SUT, determining automation feasibility across the requirements, building the TAS, executing tests, analyzing results, and rolling them up into meaningful products for leadership.

The checklists define the tasks and associated subtasks required during each phase for both Manager and Practitioner roles. The text that follows provides context and advice for best practices. Each task has the inputs feeding the task and the output deliverables expected upon successful task completion. The bottom line for the tasks summarizes the core concepts and should be viewed as a call to action.

The overall thrust for this guide is to have a Return on Investment mentality when automating. The benefits are increased efficiency of limited test resources along with the improved effectiveness from greater coverage and deeper testing to surface more defects sooner. The costs are the additional staff and skillsets needed, the initial transition costs, and the hardware & tool costs. The ROI will have to be focused long-term because there is a substantial initial investment to acquire the right resources. Based on ROI calculations, it may or may not be in the best interest to automate some or all test requirements.

This guide provides actionable information for both the manager and the practitioner across the entire automation lifecycle following the template from DoDI 5000.02 (Enclosure 4, paragraph 5.a.(12)).[3] The STAT COE site contains useful collaborative resources, https://www.afit.edu/STAT.

8. References

Page, A., Johnston, K., and Rollison, B. (2009). *How We Test Software at Microsoft*, Microsoft, WA.

Pedron, L. (2015). *Software Test Automation: Getting Started Guide for QA Managers, Quality Engineers, and Project Managers*, Independent Publisher.

Ammann, P. and Offutt, J. (2017). *Introduction to Software Testing, 2nd ed.*, Cambridge, New York, NY.

Mathur, A. P. (2008). *Foundations of Software Testing, 2/e*. Pearson, NY.

Microsoft Corporation (2007). *Performance Testing Guidance for Web Applications,* Microsoft, WA.

Molyneaux, Ian (2015). *The Art of Application Performance Testing: From Strategy to Tools, 2/e.* O. Reilly Media, NY.

Dustin, E., Garrett, T., Gauf, B. (2009). *Implementing Automated Software Testing: How to Save Time and Lower Costs*, Pearson Education, MA.

Graham, D. and Fewster, M. (2012). *Experience of Software Test Automation: Case Studies of Software Test Automation*, Pearson Education, NJ.

Fewster, M. and Graham, D. (1999). *Software Test Automation*, ACM Press, Edinburgh, UK.

Paskal, G. (2015). *Test Automation in the Real World: Practical Lessons for Automating Testing*, MissionWares, CA.

Gregg, Brendan (2014). *Systems Performance: Enterprise and the Cloud.* Pearson Education, NY.

Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, *41*(5), 507-525.

Kuhn, D. R., & Okun, V. (2006). Pseudo-exhaustive testing for software. In *Software Engineering Workshop, 2006. SEW'06. 30th Annual IEEE/NASA* (pp. 153-158). IEEE.

du Bousquet, L., Ledru, Y., Maury, O., Oriat, C. and Lanet, J.L. (2004). A case study in JML-based software validation. In *Automated Software Engineering, 2004. Proceedings. 19th International Conference on* (pp. 294-297). IEEE.

Bartholomew, R. (2013). An industry proof-of-concept demonstration of automated combinatorial test. In *Automation of Software Test (AST), 2013 8th International Workshop on* (pp. 118-124). IEEE.

Liu, H., Kuo, F., Towey, D. and Chen, T. (2014). How effectively does metamorphic testing alleviate the oracle problem? *IEEE Transactions on Software Engineering*, *40*(1), pp.4-22.

Kuhn, D., Kacker, R.,Y. and Torres-Jimenez, J. (2015). Equivalence class verification and oracle-free testing using two-layer covering arrays. In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on* (pp. 1-4). IEEE.

Kruse, P. (2016). Test oracles and test script generation in combinatorial testing. In *Software Testing, Verification and Validation Workshops (ICSTW), 2016 IEEE Ninth International Conference on* (pp. 75-82). IEEE.

Kuhn, D., Hu, V., Ferraiolo, D., Kacker, R.N., and Lei, Y. (2016). Pseudo-exhaustive testing of attribute based access control rules. In *Software Testing, Verification and Validation Workshops (ICSTW), 2016 IEEE Ninth International Conference on* (pp. 51-58). IEEE.

Mosley, D. and Posey, B. (2002). *Just Enough Software Test*, Prentice Hall, NJ.

Binder, R., Carney, D. and Novak, W. (2015). *Navy Software Test Automation Study: Product and Technology Background*. Software Engineering Institute, Carnegie Mellon University, CMU/SEI-2015-SR-047.

Few, S. (2009). *Now You See It: Simple Visualization Techniques for Quantitative Analysis*, Analytics Press, CA.

9. Glossary of Terms

The International Software Testing Qualifications Board (ISQTB) provides an on-line glossary of relevant terms and phrases for not only AST but for all of software test. The following is mostly a subset of the glossary available at http://glossary.istqb.org/search/.

API testing (Reference: ISTQB)

Testing performed by submitting commands to the software under test using programming interfaces of the application directly.

automated testware (Ref: ISTQB)

Testware used in automated testing, such as tool scripts.

capture/playback (Ref: ISTQB)

A test automation approach, where inputs to the test object are recorded during manual testing in order to generate automated test scripts that could be executed later (i.e. replayed).

confirmation testing (Synonyms: re-testing) (Ref: ISTQB)

Testing that runs test cases that failed the last time they were run, in order to verify the success of corrective actions.

coverage (Synonyms: test coverage) (Ref: ISTQB)

The degree, expressed as a percentage, to which a specified coverage item has been exercised by a test suite.

data-driven testing (Ref: ISTQB)

A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table. Data-driven testing is often used to support the application of test execution tools such as capture/playback tools.

equivalent manual test effort (EMTE) (Ref: ISTQB)

Effort required for running tests manually.

generic test automation architecture (Ref: ISTQB)

Representation of the layers, components, and interfaces of a test automation architecture, allowing for a structured and modular approach to implement test automation.

GUI (Ref: ISTQB)

Acronym for Graphical User Interface.

GUI testing (Ref: ISTQB)

Testing performed by interacting with the software under test via the graphical user interface.

keyword-driven testing (Synonyms: action word-driven testing) (Ref: ISTQB)

A scripting technique that uses data files to contain not only test data and expected results, but also keywords related to the application being tested. The keywords are interpreted by special supporting scripts that are called by the control script for the test.

level of intrusion (Ref: ISTQB)

The level to which a test object is modified by adjusting it for testability.

linear scripting (Ref: ISTQB)

A simple scripting technique without any control structure in the test scripts.

maintainability (Ref: ISO 9126)

The ease with which a software product can be modified to correct defects, modified to meet new requirements, modified to make future maintenance easier, or adapted to a changed environment.

maintenance (Ref: IEEE 1219)

Modification of a software product after delivery to correct defects, to improve performance or other attributes, or to adapt the product to a modified environment.

metric (Ref: ISO 14598)

A measurement scale and the method used for measurement.

process-driven scripting (Ref: ISTQB)

A scripting technique where scripts are structured into scenarios which represent use cases of the software under test. The scripts can be parameterized with test data.

regression testing (Ref: ISTQB)

Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed.

structured scripting (Ref: ISTQB)

A scripting technique that builds and utilizes a library of reusable (parts of) scripts.

stub (Ref: IEEE 610)

A skeletal or special-purpose implementation of a software component, used to develop or test a component that calls or is otherwise dependent on it. It replaces a called component.

system under test (SUT) (Ref: ISTQB)

A type of test object that is a system.

test adaptation layer (Ref: ISTQB)

The layer in a test automation architecture which provides the necessary code to adapt test scripts on an abstract level to the various components, configuration or interfaces of the SUT.

test automation (Ref: ISTQB)

The use of software to perform or support test activities, e.g., test management, test design, test execution and results checking.

test automation architecture (Ref: ISTQB)

An instantiation of the generic test automation architecture to define the architecture of a test automation solution, i.e., its layers, components, services and interfaces.

test automation engineer (Ref: ISTQB)

A person who is responsible for the design, implementation and maintenance of a test automation architecture as well as the technical evolution of the resulting test automation solution.

test automation framework (Ref: ISTQB)

A tool that provides an environment for test automation. It usually includes a test harness and test libraries.

test automation manager (Ref: ISTQB)

A person who is responsible for the planning and supervision of the development and evolution of a test automation solution.

test automation solution (Ref: ISTQB)

A realization/implementation of a test automation architecture, i.e., a combination of components implementing a specific test automation assignment. The components may include commercial off-the-shelf test tools, test automation frameworks, as well as test hardware.

test automation strategy (Ref: ISTQB)

A high-level plan to achieve long-term objectives of test automation under given boundary conditions.

test definition layer (Ref: ISTQB)

The layer in a generic test automation architecture which supports test implementation by supporting the definition of test suites and/or test cases, e.g., by offering templates or guidelines.

test execution automation (Ref: ISTQB)

The use of software, e.g., capture/playback tools, to control the execution of tests, the comparison of actual results to expected results, the setting up of test preconditions, and other test control and reporting functions.

test execution layer (Ref: ISTQB)

The layer in a generic test automation architecture which supports the execution of test suites and/or test cases.

test execution tool (Ref: ISTQB)

A type of test tool that is able to execute other software using an automated test script, e.g., capture/playback.

test generation layer (Ref: ISTQB)

The layer in a generic test automation architecture which supports manual or automated design of test suites and/or test cases.

test logging (Synonyms: test recording) (Ref: ISTQB)

The process of recording information about tests executed into a test log.

test management tool (Ref: ISTQB)

A tool that provides support to the test management and control part of a test process. It often has several capabilities, such as testware management, scheduling of tests, the logging of results, progress tracking, incident management and test reporting.

test reporting (Ref: ISTQB)

Collecting and analyzing data from testing activities and subsequently consolidating the data in a report to inform stakeholders.

test script (Ref: ISTQB)

Commonly used to refer to a test procedure specification, especially an automated one.

testability (Ref: ISO 9126)

The capability of the software product to enable modified software to be tested.

testware (Ref: ISTQB, Fewster and Graham)

Artifacts produced during the test process required to plan, design, and execute tests, such as documentation, scripts, inputs, expected results, set-up and clear-up procedures, files, databases, environment, and any additional software or utilities used in testing.

verification (Ref: ISO 9000)

Confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.

Appendix A: Example Return on Investment Worksheet

The following example focuses on the cost savings and Return on Investment for test automation *execution*.

To understand the cost of test execution and the possible savings with automation, we need to ask ourselves the following:

How often?

Major release testing
Minor release testing
Testing sprints

How many?

Number of tests
Number of people

How much?

Number of hours
Hourly rate

What is the Initial investment?

Test lab
Test tools
Training & certification
Consulting services

A hypothetical project may have the following attributes:

- 4 major releases per year
- 6 minor releases per year
- 2,500 existing manual tests
- 25 manual testers
- $30 average hourly rate
- 160 FTE hours per test cycle

For an automation program to begin, there will be an initial investment of:

- $60,000 to upgrade test lab (e.g., additional computers, etc.)
- $0 for open source test tools (use of commercial tools will have a cost)
- $60,000 for training and certification of test team (e.g., testing methods, testing tools)
- $240,000 for initial automation consulting implementation services (e.g., first set of tests automated)

Total initial investment: $360,000

The table below summarizes a hypothetical testing project:

	BASE YEAR	YEAR 1		YEAR 2	
PROJECT METRICS	**Manual**	**Manual**	**Automated**	**Manual**	**Automated**
How Often?					
Major Release	4	4	4	4	4
Minor Release	6	6	6	6	6
How Many?					
Number of Tests	2500	2000	500	300	2200
Number of People	25	20		3	4
How Much?					
FTE Hours	160	160		160	40
Hourly Rate	$30	$30		$30	$30
Investment Cost			*$360,000*		
Operational Costs					
Total Hours/Test Cycle	4000	3200	N/A	480	160
Hours/Test	1.6	1.6	N/A	1.6	0.1
Cost/Test	$48	$48	N/A	$48	$2
Cost/Cycle	$120,000	$96,000	N/A	$14,400	$4,800
Cost/Year	$1,200,000	$960,000	$360,000	$144,000	$48,000

The operational costs are calculated as follows:

Total Hours per Test Cycle = People * FTE Hours

Hours per Test = Total Hours per Test Cycle / Number of Tests

Cost per Test = Hourly Rate * Hours per Test

Cost per Cycle = Number of Tests * Cost per Test

Our investment is a non-recurring year 1 cost calculated as follows:

Investment Cost = Lab Upgrades + Test Tools + Training & Certification + Implementation Services (labor)

The table highlights the following:

Baseline Year Manual Testing – The current level of effort and cost to perform software testing manually. This assumes all 2500 tests are executed for each cycle by the all-manual test group.

Year 1 Year Automated Testing – The initial year of automation is subcontracted at a fixed priced cost of $240,000. Additional investment of $120,000 for upgrading the test lab with additional computers, training and certifying team members on test and automation topics, and providing hands-on training on test tools.

Year 1 Year Manual Testing – The current level of testing is reduced by 500 tests which are now executed through automation. The remaining 2000 tests are executed by the manual test group.

Year 2 Automated Testing – With the automation infrastructure in place from year 1, a total of 2200 tests are converted to automation. Minimal staff requirement to run these automated tests. However, existing staff are now more heavily involved in pre-execution activities (e.g. defining test cases, test data, etc.) and post-execution activities (e.g., analysis, reporting, etc.).

Year 2 Year Manual Testing – The remaining 300 manual tests are now executed by a few manual testers.

Conclusions:

BASELINE YEAR MANUAL TEST EXECUTION COSTS **$1,200,000**

YEAR 1 FOR BOTH MANUAL AND AUTOMATED TEST EXECUTION COSTS **$1,320,000**

COST(SAVINGS) OF YEAR 1 OVER BASELINE YEAR **$120,000**

YEAR 2 FOR BOTH MANUAL AND AUTOMATED TEST EXECUTION COSTS **$192,000**

COST(SAVINGS) OF YEAR 2 OVER BASELINE YEAR **($1,008,000**)

COST(SAVINGS) OF YEAR 1 & YEAR 2 OVER BASELINE YEAR **($888,000)**

THEREFORE, WE CAN EXPECT TO RECOUP OUR INVESTMENT IN TEST AUTOMATION OVER A 2 YEAR PERIOD AND PROVIDE SIGNIFICANTLY LOWER ONGONG COSTS OF EXECUTION FOR AN EXCEPTIONAL ROI WELL INTO THE FUTURE.

Appendix B: Certification, Education, and Resources for AST

The following resources provide Certification and Accredited Training for Software Testing and Test Automation topics:

International Software Testing Qualification Board

URL: https://www.istqb.org/

As of December 2017, ISTQB® has administered over 785,000 exams and issued more than 570,000 certifications in over 120 countries world-wide. The certification relies on a Body of Knowledge (Syllabi and Glossary) and exam rules that are applied consistently all over the world, with exams and supporting material being available in many languages.

American Software Testing Qualification Board

URL: https://www.astqb.org/

The mission of ASTQB is to promote professionalism in Software Testing in the United States. They provide and administer quality exams for the ISTQB, ASTQB and IQBBA certifications, by supporting and facilitating software training providers in delivering high quality courses, by actively engaging in the ISTQB working groups, and by supporting efforts to develop and encourage people who are already in or are entering the software testing profession.

ASQ

URL: https://asq.org/cert/software-quality-engineer

With individual and organizational members around the world, ASQ has the reputation and reach to bring together the diverse quality champions who are transforming the world's corporations, organizations and communities to meet tomorrow's critical challenges. The Certified Software Quality Engineer understands software quality development and implementation, software inspection, testing, verification and validation, and implements software development and maintenance processes and methods. There are some components of automation in test covered across the topics.

QAI Global
URL: http://www.qaiusa.com/software-certifications/software-testing-certifications/

As the IT industry becomes more competitive, the ability for management to distinguish professional and skilled individuals in the field becomes mandatory. Quality Assurance International (QAI) Global Institute is the global program administrator for the International Software Certification Board (ISCB). Software Certifications has become recognized worldwide as the standard for information technology quality professionals – having certified over 50,000 professionals. ISCB test centers are located in 135 countries across 6 continents. Software certifications cover five major domains and provide eleven professional certifications. These internationally-recognized, examination-based and vendor-independent programs provide full career paths for professionals at all levels.

The following resources provide reporting on automated test tool topics:

Magic Quadrant for Software Testing Tools

URL: https://www.gartner.com/home
The need to support faster time to market with higher quality is driving the demand for effective functional test automation tools. We evaluate vendors in this space to help application leaders who are modernizing software development select test automation tools that best match their needs. *(note: may require subscription for access to reports)*

Carnegie Melon University Software Engineering Institute - The Importance of Automated Testing in Open Systems Architecture Initiatives
URL: https://insights.sei.cmu.edu/sei_blog/2014/03/the-importance-of-automated-testing-in-open-systems-architecture-initiatives.html

Carnegie Melon University Software Engineering Institute - Five Keys to Effective Agile Test Automation for Government Programs
URL: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=503507
In this discussion-focused webinar, Bob Binder and Suzanne Miller discuss 5 key questions that government organizations contemplating embarking on adopting automated test techniques and tools in an Agile environment are likely to have.

The following resources provide for collaborative discussions around test tool topics:

SW Test Academy
URL: https://www.swtestacademy.com/

SW Test Academy (STA) is focused on mainly technical testing topics. In this site, you can find comprehensive descriptions and examples of test automation, performance testing, mobile testing, web service testing, API testing, DevOps, continuous integration, and similar topics.

QA Testing Tools
URL: http://qatestingtools.com/
Quality Assurance (QA) Testing Tools is an innovative platform and is the only website that gives you an Opportunity to read technical reviews on every software-testing tool, simultaneously giving you in-depth technical information, and comparison tables that direct you towards the most suitable group of tools to fulfill your requirements.

Automate the Planet
URL: https://www.automatetheplanet.com/resources/
Learn how to write automated tests through working real-world examples.

Stack Overflow
URL: https://stackoverflow.com/
Each month, over 50 million developers come to Stack Overflow to learn, share their knowledge, and build their careers.

Software Testing and Quality Assurance Forums
URL: http://www.sqaforums.com/forums/
The online community for software testing and quality assurance professionals.

Open Source Testing
URL: http://www.opensourcetesting.org/
The open source testing site aims to boost the profile of open source testing tools within the testing industry, principally by providing users with an easy to use gateway to information on the wide range of open source testing tools available.

Test Automation Group on LinkedIn
URL: https://www.linkedin.com/groups/86204
LinkedIn is the world's largest professional network with more than 562 million users in more than 200 countries and territories worldwide. The Test Automation LinkedIn group is for people that are interested in QA test automation. The following issues can be found in the group discussions: Automation frameworks, Selenium, Quick Test Pro (QTP), Web automation, Automation ROI, TestComplete, XUnit, JUnit, NUnit, JSystem, Automation strategics, Mobile testing (Android, iPhone, Blackberry), Load, agile, jobs and more! *(Note: There are several additional groups in LinkedIn that cover test automation topics and specific tools)*

Appendix C: Tool Evaluation Worksheet

To evaluate automated testing tools and to establish a proper fit for the project or program, we define requirements for each of the following categories:

Platform

Identify the various platforms which need to be tested and make sure that the tool you select can either a) run on the platform, or b) remotely test that platform. If there is a requirement to test directly on a platform (e.g. specific version of OS, like Windows 10), ensure the tool supports that platform.

Test Types

Tests are structured based on what they intend to achieve. Each test type and level may require a different strategy and possibly a different test tool. Evaluate what tests you are looking to automate in order to select tools that can support the test type and level required.

Test types may include:

- Functional
- Non-Functional
- Structural/Architectural
- Performance
- Security
- Regression

Test levels may include:

- Unit
- Component
- Integration
- Sub-system
- System
- System Integration
- System of Systems

Technology

There are test tools that support one specific technology (e.g. browser testing) and others that support a multitude of technologies. One would think that the tool supporting the most technology would be the best candidate. However, with technology progressing at such a rapid rate, a “bundled” solution may not always have each component of the bundle up-to-date. Therefore, it is important to also consider tools that target a specific technology and do it effectively. This ` la carte solution may be integrated with yet other collaboration/integration tools in order to provide a seamless solution.

Scripting Language

The scripting language is what gives the tool the ability to customize the automated test for your project or program. Having a robust scripting language will provide flexibility and expandability to the automated testing solution. Tools employ a variety of languages (e.g., Java, Python, C#, C++, VBScript, etc.) to perform operations and provide customization. These languages require that automators have an understanding and skills necessary to properly create maintainable automation solutions.

Logging

The logging function of a tool is important in determining what errors may have occurred specific to the automated tests running. These logs are sometimes automatically generated by the test tool; otherwise, they need to be created via the scripting language. It is important to be able to view messages regarding access, warnings, and errors, which may have prevented an automated test from executing properly.

Reporting

The reporting features of a testing tool should provide displays of data and analysis that support decision making, which may include having to re-run tests using different parameters. Tools generally have built-in functionality for reporting but should also allow for the customization of reports and generation of dashboards. The capability to export test data is helpful for integrating it into project management planning and reporting systems.

Ease of Use

Test tools should be intuitive and easy to understand and use. They should be well-documented so that common answers to questions and techniques can be quickly identified. For example, one may need to know what function is used to verify the content on a calendar widget or how to export a test report in comma separated values (CSV) format.

Although some test tools may claim that there is no need to use scripting for test development, most medium-to-complex test requirements will likely require a level of scripting to meet standards and requirements. Therefore, “ease of use” is a relative term. For those automators with strong programming skills, it will be easy to develop scripts with a test tool. However, for the traditional manual tester, the requirement to write program code may be difficult and counterproductive to their domain-specific skills.

Vendor

Traditionally, test tools were purchased from a commercial entity that produced them and provided support and maintenance services. While this is still the case, the open source software community has developed robust offerings through a platform of sharing and innovating. Therefore, both sources of test tools should be considered. Security certification and accreditation may also be a requirement to the introduction of tools into certain programs or environments.

Support

Support is no longer only available from vendors. There are a multitude of forums on the internet that focus specifically on particular testing tools. While these should not be viewed as traditional vendor support plans, with a customer expectation of receiving a response, there are no doubt many other

users willing to contribute their technical know-how and helping like-minded testers in solving automation problems. The quality of support may vary greatly and it is not always the case that paying for support provides better service.

Cost

Test tools from commercial entities have a variety of structures for licensing fees. These can include individual, per seat, and floating. Understanding the intended usage by your team will help define which license structure that is best for you. Additionally, commercial entities have support and maintenance fees. Support handles technical questions or issues while maintenance usually provides for minor and major updates to the testing tool. Working with the most recent version of a tool will often help resolve technical issues (e.g., compatibility, defects, etc.). Training should also be included as a cost, both for vendor and open source solutions.

EVALUATION TABLES

The evaluation categories should be assigned a weight, given their relative importance to one another. The total weight must add up to 100%. This facilitates comparisons across different tools that are being evaluated.

Category	Weight
Platform	10%
Test Types	5%
Technology	20%
Scripting Language	20%
Logging	5%
Reporting	5%
Ease of Use	10%
Vendor	10%
Cost	5%
Support	10%
TOTAL	**100%**

(sample values for illustration)

Each category may have several supporting factors, each which need to be evaluated. A score can be assigned for each factor based on the evaluation criteria for the sub-factor. The sub-factor scoring may consist of a score ranging from 0 to 5 where each score indicates the following about the desired attribute:

Score	Value	Description
0	N/A	Not Available
1	Poor	Most or all defined requirements not achieved
2	Fair	Some requirements not achieved
3	Good	Meets requirements
4	Excellent	Meets or exceeds some requirements
5	Outstanding	Significantly exceeds requirements

For each category there may be several factors that need to be evaluated:

Category	Factors
Platform	4
Test Types	5
Technology	10
Scripting Language	4
Logging	2
Reporting	2
Ease of Use	4
Vendor	2
Cost	2
Support	2

(sample values for illustration)

For example, an evaluator could assign the following values to the four factors of the Platform category for tool “X”:

Tool "X" Platform	Value
Windows 10	5 / 5
Windows 7	2 / 5
Mac OS	0 / 5
Android	0 / 5
TOTAL	7 / 20

In summary, in the scenario above we find that:

- The support tool “X” receives 7 out of a possible 20 points across the required platforms.
- Tool “X” receives 3.5% of the maximum possible 10% for the Platform category.

There can be as many or as few factors defined for each category as you see fit for your project or program. To save time and effort, consider first screening all tools only on technical categories (Platform, Test Types, Technology, Scripting Language, Logging, Reporting, and Ease of Use) and only evaluate tools that are technically acceptable on the non-technical categories (Vendor, Cost, Support).

Finally, no tool evaluation is complete without trying the tool out in the intended environment where there is an intent to automate. This will ensure that the tool can actually perform the intended functions within the constraints of the test team’s skills and resources.

Appendix D: Considerations for Automating Test Requirements and Test Cases

This appendix provides additional insight into the requirements generation process and the evaluation of test cases.

Requirements:

Defining test requirements is difficult and requires a process decomposition to ensure the SUT is adequately tested. One decomposition approach created by MITRE and endorsed by DASD (DT&E) is the Developmental Evaluation Framework (DEF) shown in Figure D.1. This provides an analytically tractable way to manage a test program to best support acquisition decisions. This methodology has also been used throughout the DoD and is directly applicable to AST implementation. The DEF supports the requirements decomposition process where the goal is to flow from a capability to precisely defined and testable requirements. The capabilities are the Developmental Evaluation Objectives (DEO), which we can translate to the software testing world as functions. These DEOs are categorized into broad areas such as system performance, cyber-security, interoperability, and reliability & maintainability. These objectives are broken down even further with technical measures assigned to each. The AST process would break down these system requirements and test & evaluation (T&E) measures even further into the functional components and tasks. This extra level of detail should be added to the existing DEF matrix to ensure requirements traceability. These tasks form the basis for test cases and scenario generation. The right side of Figure D.1 could be adapted by replacing the 'Decisions' in the second row with 'Scenarios' and the entries would be the applicable test cases.

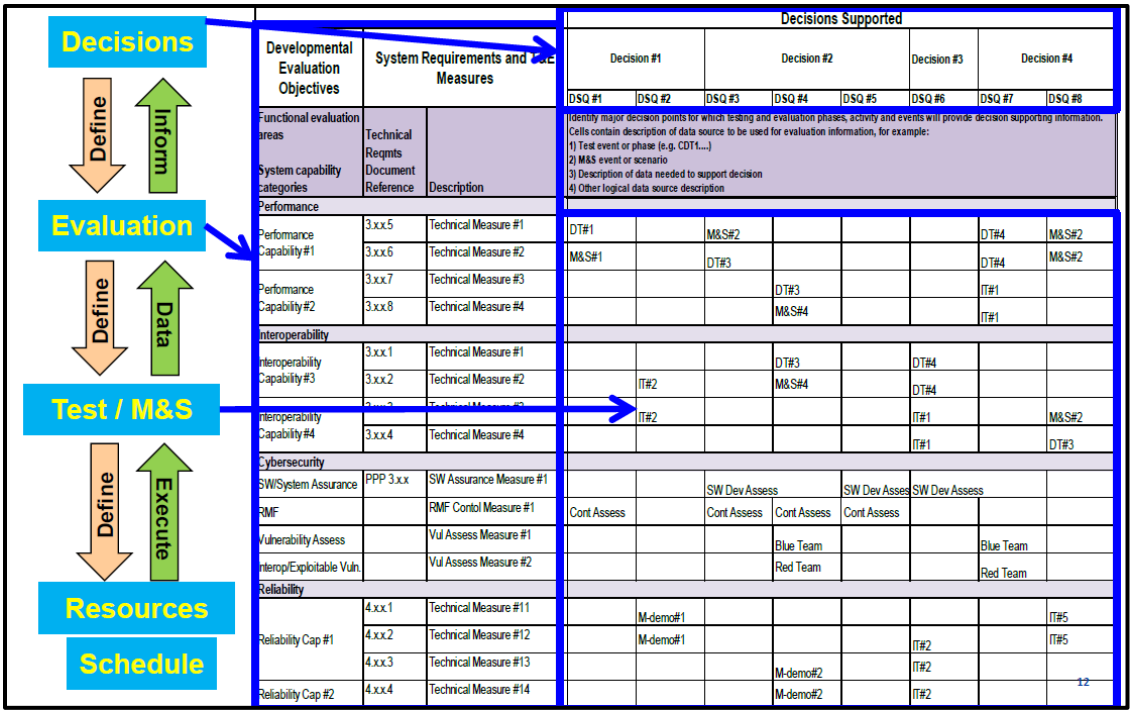

Figure D.1. MITRE Development Evaluation Framework[17]

[17] https://ndiastorage.blob.core.usgovcloudapi.net/ndia/2016/Test/Cortes.pdf. 2016.

The DEF process documents sources for each requirement and can be used to help prioritize which requirements get automated. Although in general the desired is that the most important capabilities receive top priority, but this process does not necessarily translate into the correct prioritization of the AST requirements. Additional consideration needs to be given to the risks of automation. Automation can be more complex depending on the requirement and the consequences of failure can be higher for testing some requirements. These value/risk tradeoffs should be evaluated and integrated into an overall prioritization scheme. Attention and weights should be considered for:

- Contribution to DEOs (which are rank ordered)
- Probability that automating the requirement fails based on complexity or script development delays
- Consequences of failing to automate the requirement
- Ability of manual methods to effectively test the requirement

There are many ways to compute a priority number, such as a weighted average. The test team should integrate the prioritized list of requirements in all activities across the AST lifecycle.

Automated software testing has many definitions and interpretations. Depending on the program and staffing, AST can be quite complex, taking the test team years to develop the right automation framework or it can be quite simple when a few basic steps of a test are automated, like running batch files overnight. There should be a general understanding of how complex the AST will be coming out of the Assess Phase research. An important overall consideration is the level of abstraction. That is, how does the system operate during test? It could be live operators, virtual scripts, modeling and simulation, or running a pre-recorded mission thread. This Live, Virtual, Constructive (LVC) environment will influence many other facets of the testing to include choice of venue (virtual network, software integration lab (SIL), contractor facility, or operational environment).

Based on the automation requirements and potential available tools, determine possible matches to best execute an AST in both the short and long-term. Possible considerations include:

- Bitmap capture-replay that limits how test inputs may be applied and how system responses are evaluated versus the de facto standard (currently) of direct programmatic access to GUI API's
- Need to maintain system integrity and mimic operational performance by hosting the tool apart from the system under test by not altering its source code (client/slave relationship)
- Planning to ensure convenient reports that show evidence that the test properly executed
- Ability to have data-driven tests with unique data entry flexibility while accounting for the reality of a reduced installation and test time
- Requirements for shorter tool learning curves
- Minimizing sustainment/maintenance time for software version changes and tool version changes
- Ability to conduct both positive and negative testing where faults are intentionally inserted

Appendix E: Test Oracle Approaches

Automated approaches to solving the oracle problem include:

- ***Fuzz/crash testing*** is typically done by recording some data input, then randomly permuting data fields to generate a large number of tests (possibly tens to hundreds of thousands). These are run to determine if any inputs cause the system to crash or freeze. Fuzz testing is used extensively by commercial software developers (less for functionality) for early detection of major faults with numerous tools and references available.
- ***Embedded assertions*** is a popular "light-weight formal methods" technique to embed assertions within code to ensure proper relationships between data, i.e., as preconditions, post-conditions, or input value checks. Tools such as the Java Modeling language (JML) can be used to introduce very complex assertions, effectively embedding a formal specification within the code. The embedded assertions serve as an executable form of the specification, thus providing an oracle for the testing phase. Reference: du Bousquet, L., Ledru, Y., Maury, O., Oriat, C. and Lanet, J.L., 2004, September. "A case study in JML-based software validation." In *Automated Software Engineering, 2004. Proceedings. 19th International Conference on* (pp. 294-297). IEEE.
- ***Model based test generation*** is a formal (mathematical) model of the system under test, typically expressed in temporal logic or as an extended state machine, or in design notations such as Unified Modeling Language, UML. A simulator or model checker is then used to generate expected results for each input. If a simulator can be used, expected results can be generated directly from the simulation, but model checkers are widely available and can also be used to prove properties such as liveness in parallel processes in addition to generating tests. Conceptually, a model checker can be viewed as exploring all states of a system model to determine if a property claimed in a specification statement is true. Reference: Bartholomew, R., 2013, May. "An industry proof-of-concept demonstration of automated combinatorial test." In *Automation of Software Test (AST), 2013 8th International Workshop on* (pp. 118-124). IEEE.
- ***Metamorphic testing*** uses a small set of tests whose expected outcome has been determined manually. Then system properties are used to generate other tests with different inputs, whose expected outcomes can be produced from the original test. For example, in testing a sine function, it must be the case that $\sin x = \sin(\pi - x)$. Thus, the program is tested for a correct result for sin *x*, then a new test can be generated using the input $\sin(\pi - x)$. Reference: Liu, H., Kuo, F.C., Towey, D. and Chen, T.Y., 2014. "How effectively does metamorphic testing alleviate the oracle problem?" *IEEE Transactions on Software Engineering*, *40* (1), pp.4-22.
- ***Match testing using two-layer covering arrays*** suggests that test settings for an input factor may represent ranges of values (called equivalence classes) for which the output is expected to remain unchanged. For example, a shipping program may charge the same rate for any package under one pound, a second rate for packages one pound to 10 pounds, and a third rate for packages over 10 pounds. Values within each of these ranges are equivalent with respect to the cost calculation. Any value within an equivalent range may be substituted for any other and the program output should be unchanged. The test method works by generating two test arrays: a primary array and a secondary array. The entries of primary array represent names of equivalence classes of input factors. For each test row of the primary array, a second array is computed. The settings in second array are the values from equivalence classes corresponding to the names of equivalence classes in the primary array. If the outputs corresponding to one row of the primary array differ, then either the equivalence classes were defined incorrectly or the code is faulty in some way. Reference: Kuhn, D.R., Kacker, R.N., Lei, Y. and Torres-Jimenez, J., 2015, April. "Equivalence class verification and oracle-free testing using two-layer covering

arrays." In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on* (pp. 1-4). IEEE.

- ***Classification tree method*** is a graphical method for analyzing program inputs and their value partitions, then turning these into test cases. A tree structure is defined with one branch for each parameter or factor in the program inputs. For each of these, branches are then defined for each equivalence class of that parameter or factor. Equivalence partitioning is done as in other test approaches. After the tree has been constructed, weights can be attached for the frequency of occurrence of factor values in inputs, which are then used in optimizing and prioritizing tests. Reference: Kruse, P.M., 2016, April. "Test oracles and test script generation in combinatorial testing." In *Software Testing, Verification and Validation Workshops (ICSTW), 2016 IEEE Ninth International Conference on* (pp. 75-82). IEEE.
- ***Pseudo-exhaustive testing*** relies on the fact that not all outputs depend on every possible combination of input variables. The method depends on exhaustive testing of all combinations of variable values that truly matter using combinatorial arrays along with the automated generation of test oracles for model checking. An advantage of this method is that it can be used to produce a complete test set in the sense that all negative cases as well as all positive cases are verified. Two arrays are generated, one for positive tests and one for negative. Reference: Kuhn, D.R., Hu, V., Ferraiolo, D.F., Kacker, R.N. and Lei, Y., 2016, April. Pseudo-exhaustive testing of attribute based access control rules. In *Software Testing, Verification and Validation Workshops (ICSTW), 2016 IEEE Ninth International Conference on* (pp. 51-58). IEEE.

Appendix F: Acronym List

Acronym	Description	Acronym	Description
AFB	Air Force Base	**JWICS**	Joint Worldwide Intelligence Communications System
ALPI	ALP International	**LVC**	Live, Virtual, Constructive
API	Application Programmer Interface	**NIST**	National Institute of Standards and Technology
AST	Automated Software Testing	**NMCI**	Navy Marine Corps Internet
ASTQB	American Software Testing Quality Board	**OS**	Operating System
BCAC	Business Capability Acquisition Cycle	**OSD**	Office of the Secretary of Defense
CAC	Common Access Card	**OT**	Operational Test
CDRLs	Contract Data Requirement Lists	**PM**	Program Manager
CMMI	Capability Maturity Model Integration	**POA&M**	Plan of Action and Milestones
COE	Center of Excellence	**POC**	Point of Contact
CRON	Command Run On	**QA**	Quality Assurance
CSV	Comma Separated Values	**QAI**	Quality Assurance International
DASD(DT&E)	Deputy Assistant Secretary of Defense, Developmental Test and Evaluation	**QTP**	Quick Test Pro
DCGS	Distributed Common Ground System	**Ref**	Reference
DEF	Developmental Evaluation Framework	**RITE**	Rapid Integration and Test Environment
DEO	Developmental Evaluation Objectives	**RM**	Requirements Management
DI2E	Defense Intelligence Information Enterprise	**ROI**	Return on Investment
DLLS	Dynamic Link Libraries	**ROM**	Rough Order of Magnitude
DoD	Department of Defense	**SAT**	System Acceptance Testing
DODI	Department of Defense Instruction	**SE**	Software Engineer
DT	Developmental Testing	**SIL**	Software Integration Lab
DT/OT	Developmental Testing/Operational Testing	**SME**	Subject Matter Expert
EMTE	Equivalent Manual Test Effort	**SPAWAR**	Space and Naval Warfare Systems Command
FRACAS	Failure Reporting, Analysis, and Corrective Action System	**STA**	Southwest Test Academy
FRB	Failure Review Board	**STAT**	Scientific Test and Analysis Techniques
FTE	Full Time Equivalent	**SUT**	System(s) Under Test
GTAA	Generic Test Automation Architectures	**T&E**	Test & Evaluation
GUI	Graphical User Interface	**TAA**	Test Automation Architecture
I/O	Input/Output	**TAF**	Test Automation Framework
ICSTW	IEEE Conference of Software Testing, Verification, and Validation Workshops	**TAS**	Test Automation System
IEEE	Institute for Electrical and Electronics Engineers	**TDD**	Test-Driven Development
IQBBA	International Quality Board for Business Analysts	**TEMP**	Test and Evaluation Master Plan
ISCB	International Software Certification Board	**UFT**	Unified Functional Testing
ISO	International Organization for Standardization	**UI**	User Interface
ISQTB	International Software Qualifications Testing Board	**UML**	Unified Modeling Language
JML	Java Modeling Language	**VB**	Visual Basic
JMPS	Joint Mission Planning System	**VV&A**	Verification, Validation, and Accreditation