Test Monitoring & Test Control during Software Test Execution

Submitted By

FAISAL MAHMUD

ID: 192-17-408

Department of Management Information System (MIS)

Daffodil International University, Dhaka

This report only meets a portion of the criteria. Master of Science Degree Requirements (MS) in Management Information System (MIS)

Supervised By

Professor Dr. Md. Ismail Zabiullah

Professor

Daffodil International University's

Department of Computer Science and Engineering

DAFFODIL INTERNATIONAL UNIVERSITY

DHAKA, BANGLADESH

January 2021

APPROVAL

This Thesis/Project titled **"Test Monitoring & Test Control during Software Test Execution"** submitted by FAISAL MAHMUD, ID:192-17-408, Daffodil International University has been approved as satisfactory for the partial fulfillment of the criteria Bachelor of Science in Computer Science and Engineering (BSCSE) by the Department of Computer Science and Engineering. MS in Management Information System and approved as to its style and contents. The presentation has been held on 10 January 2021.

BOARD OF EXAMINERS

Professor Dr. Touhid Bhuiyan **Chairman**

Professor & Head Department of CSE

Faculty of Science & Information Technology

Daffodil International University

Professor Dr. Md. Ismail Zabiullah **Internal Examiner**

Professor Department of CSE

Faculty of Science & Information Technology

Daffodil International University

Mr. Md. Sadekur Rahman Assistant **Internal Examiner**

Professor Department of CSE

Faculty of Science & Information Technology

Daffodil International University

Dr. Mohammad Sharif Uddin **External Examiner**

Professor Department of CSE

Jahangirnagar University

DECLARATION

Consequently, I declare that this thesis, "Test Monitoring & Test Control During Software Test execution," is valid. Software Test Execution" has been done by me under the supervision of Professor Dr. Md. Ismail Zabiullah, Department of Computer Science and Engineering, Daffodil International University. I further declare that this thesis, or any part of it, has not been submitted to any other school to receive a degree.

Supervised by:

Mr. Md. Zahid Hasan

Assistant Professor

Department of CSE

Daffodil International University

Submitted by:

FAISAL MAHMUD

ID: 192-17-408

Department of MIS

Daffodil International University

ACKNOWLEDGEMENT

First and foremost, I would want to convey my heartfelt gratitude to Almighty Allah for the great blessings he has bestowed upon me. It is feasible for me to finish the final year project/internship successfully. I am grateful and wish my profound indebtedness to Professor **Dr. Md. Ismail Zabiullah**, Department of CSE, Daffodil International University, Dhaka. My supervisor's deep knowledge & keen interest in the field of "Test Monitoring & Test Control during Software Test Execution" helps me carry out this thesis. His never-ending tolerance, academic instruction, persistent encouragement, frequent and vigorous supervision, constructive criticism, valuable suggestions, and reading many poor versions and fixing them were all invaluable to me. This thesis was completed over a series of steps.

I want to express my heartiest gratitude to Professor **Dr. Touhid Bhuiyan** and the Head, Department of CSE, for his invaluable assistance in completing my dissertation. Thesis and other faculty members and the staff of the MIS department Daffodil International University.

I want to thank my entire course mates in Daffodil International University, who took part in this discussion while finishing the course's requirements.

Finally, I'd like to thank you for everything you have done for me.

ABSTRACT

This guide is intended to **"Test Monitoring & Test Control during Software Test Execution"** interested in applying automation to software testing. It involves a system engineering process based on the scientific method to conduct and achieve an automation capability and the critical need to perform a return-on-investment analysis to make the business case for automation. For the first time, some businesses are considering automation. For that audience, we recommend following through with all of the steps outlined in the installation guide—others organizations with prior automation experience. There is also a plethora of information accessible. There is a wealth of information in textbooks and on the internet. Some software acquisition programs have had some automation exposure or experience and are looking to improve one or more parts of their automation process. Typical objectives are to select additional system functions or capabilities to automate, secure more Automators, or change or expand the tools used. Only a portion of the guide may apply to those individuals. The following phases of implementation are used to arrange the guide.

Table of Contents

Chapter 1: Introduction

1.1 Executive review

This guide is intended to assist you. Individuals working for the Pentagon who are interested in employing automation in software testing should apply. It employs a system engineering approach based on the scientific method of action. It accomplishes automation capabilities and the crucial necessity to automate the business case's return on investment analysis (ROI). Who is the student you're looking for? Some firms are exploring automation for the first time. We propose that you use all of the specified parts in the implementation guide for those audiences. It is necessary to enlist the help of automated knowledge teams from other sources. Seek their advice and make use of their knowledge.

The guidance is divided into the phases described below to help you incorporate an automated software testing life cycle into your testing program.

Prepare ahead of time by conducting a study, devoting time, and gathering data to make an informed conclusion concerning automation. Do a cost-benefit analysis to determine the return on investment (ROI), and be sure to incorporate any long-term advantages before deciding whether or not to employ them. Research into the system's assessment system, automation capabilities and possibilities, understanding of staff skills and resource requirements, and calculating the costs and benefits of automation are all examples of specific measurements.

Edit - identify and correct flaws in an automated software testing tool. Edit - Discover and prioritize test needs, identify and test relevant automation tools using immeasurable and visual metrics, identify impediments to automated usage, write an automation test framework, and define test script criteria to modify an automated software testing program. Design - examine the automation process in greater depth and determine ways to improve automation. At this step, automation tools are chosen and made available, automated testing conditions are carried out, test cases are defined, an output analysis approach is constructed, and regulatory suspensions are put in place, all of which led to a design review.

If time is of the essence and an automated choice must be taken right away, as well as short turns, be sure to examine the following: Examine the various automation options and determine which automation technologies are most effective. Be aware of the price. Introduce ROI to get leadership support and rapidly identify roadblocks to success. Learn which portions of your test are suitable for automation, establish a default framework, and decide which tools your automation program will require. Purchase, rent, or update automation equipment. Growing up isn't as difficult as you would imagine. Begin with easy automation chores and gradually raise the complexity as automation's power develops. Consider your options carefully.

1.2 Contribution of the thesis

The Department of Defense (DoD) and the sector have both benefited from test software. Throughout the software development process, the Department of Defense has not fully used the efficiency and improved functionality that automated approaches may provide. A lack of DoD understanding of the AST process is connected to the failure to fully use AST techniques. The goal of this Guideline is to offer a handbook to management and operators that outlines the foundation for using AST techniques in diverse applications. This endeavor is part of the Office of the Secretary of Defense's (OSD) Deputy Secretary-General for Defense, Development, and Testing (DASD (DT & E)) STAT COE's effort to improve educational programs on the battlefield.

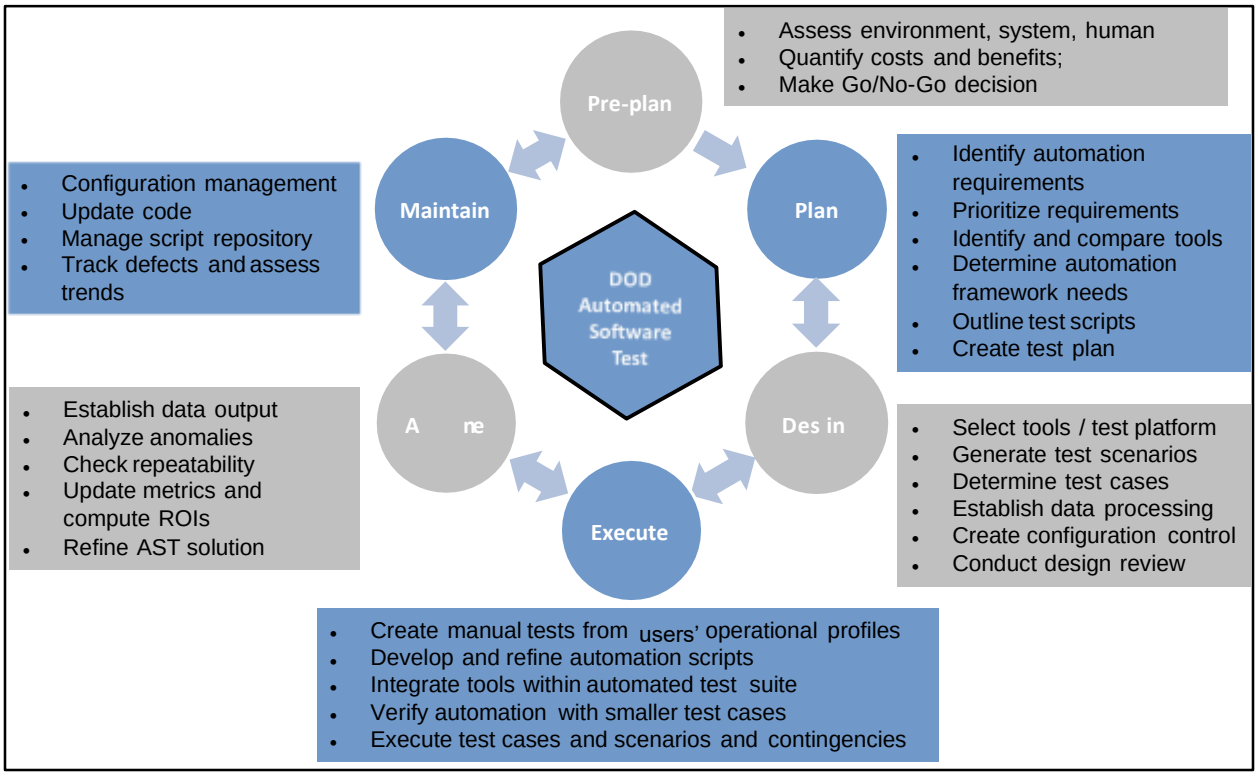

Figure 1. Major phases and tasks for AST programs.

The target audience includes program engineers, software engineers, software developers, software testers, and test automation (both program and test). Tasks are specified at a high level, and technical specifics are established from the perspective of someone who has no experience with software testing or automation. The AST flow approach was primarily created through conversations with professionals from all around the world.

Perhaps both the Department of Defense and business have succeeded and failed in automating test cases. Additional sources included previous DoD research, textbooks, technical journals, websites, blogs, and conference briefings. Even though each program's automation path appears to be distinct, there are certain commonalities.

Chapter 2: Related Work

2.1 Phase 0: Pre-Idea

Automation can enhance test performance and efficiency significantly, but it may necessitate a large investment. It is not necessary to automate all programs and needs. Take the time to assess the usefulness of an automated software testing program in terms of cost, not just in the near term but also in the long term. This ROI or business case study does not need exact data, but it does necessitate a logical engineering system and a decision-making procedure. Having a performance analyst with these abilities and expertise, or seeking outside aid or guidance, is crucial. In this pre-production stage, an insufficient order size (ROM) ROI balance is necessary.

2.2 Test Program and Automatic Software Assessment Demeanor Study

Common knowledge of the problem is required. Do some study to see if an automated system is suitable for you. The technique that makes sense is to put the system's goals, objectives, and needs to the test and seek appropriate automation options. If you're being tested at the start of a unit, let us create software for complicated program components, tests that just need to be done once might not make sense. Automation, on the other hand, maybe useful, especially in regression testing, if a program established in software development continues, stating that integration testing with new capabilities is performed frequently.

As a regular processor contract obligation, the software development contractor may conduct AST internally. Try to become aware of what they've done or are doing, and see if you can gain access to their automated work by posting (e.g., Contract Data Requirement Lists or CDRLs) or visiting a site with exhibits and papers.

Below: Investigate the possibility of automating your program. Find the proper individuals to conduct the technical testing, learn about the contractor's automation efforts, and understand the continuous rotation of the same plan.

2.3 Recognize the Situation

Where there is sufficient capacity and support within the business, automation projects have a greater chance of succeeding. Most DoD organizations are unfamiliar with the idea of automatic testing. A single entity plays a vital role in guiding the leadership in AST promotion and active management of the process in all organizations, whether government or commercial. Culture may be both a necessary and insurmountable barrier. If manual testing is "the way we always do it," then moving forward will require a combination of leadership, policy, and the involvement of technical talents. As programs transition to an experimental development method (TDD) with Agile and DevOps, AST is a key component Below: Open the path to automation success by gaining leadership support, identifying significant components that need to be implemented, and comparing the resources required for the current state of your system. Get an idea of the obstacles you must overcome to build the strength of automation.

2.4 Assess Manpower and Skillset Requirements

Between software testers and Automator, there is frequently a significant gap in skills and abilities. With today's tools, testers with little or no software development expertise may learn to manually perform various test features. However, to build completely autonomous and reusable test scripts that take full use of automation capabilities, a fully automated, optimized, sustainable, and usable automation solution necessitates considerable software coding and development expenditure. Rarely is a single automated, user-friendly automation tool (GUI) the only answer to all changing requirements. Each tool serves a specific function (for example, browser apps, tracking, unit testing, and continuous testing).

2.5 Determine the automation's Potential Benefits

The primary objective of automated software testing is to discover flaws and enhance performance opportunities sooner and more efficiently than manual testing would allow. When comparing a complete manual to another automation, some metrics to examine are: Increased the number of lines of code that have been examined. Installation of anticipated operating systems and use cases has increased. In manual testing, especially repeated examinations (e.g., IQ tests), staff retention is important. Multiple users and locations may be measured, and high-level tests and parameters can be performed. • The test force's capacity to concentrate its efforts on high-priority/high-risk regions • Improved data output for analysis and reporting • A higher disco of defects

2.6 Calculate the Automation Effort Costs

The project's costs are both direct and indirect—standard projects. Software licenses and training; hardware and middleware components of the default test framework; cloud and network services; reading tools, writing, compiling materials, conducting tests, and analyzing results; and consultation costs with the contractor are all included in the direct prices of representatives.

Indirect costs are time spent doing the default that is hidden or unanticipated. Time taken away from the task at hand, time spent working on numerous charges, time spent notifying management, time spent learning how to use various tools, and time spent saving the default solution are all examples. The importance of the small details is sometimes neglected.

2.7 Decide Based on Expected Return on Investment

Not all conditions should be met automatically. If particular tests are performed in or on reliable software builds and facilities, the cost will not outweigh the advantages. Take some time to think about the cost, quantity, and quality of the automated effort.

It's a good idea to estimate the time it'll take to complete the automated operation till the manual test is finished. Most benefits and expenses are not evaluated as previously, as indicated in the benefit and cost categories. It's a good idea to spread this exam out across a few days. To show the sensitivity of the results, make a list of projected expenses and benefits (e.g., optimistic, realistic, joyful). Analytics should be used by leaders to balance their thoughts, expectations, and actions.

Chapter 3: Case Study

3.1 Phase 1: Plan

Now that you've opted to test the program in the default manner, it's time to develop a more thorough strategy. Fortunately, excellent planning has already established a viable business case: automated research, automation elsewhere, assessing the required resources, and selecting tool solutions and cost choices. In Section 1, we'll use a systematic planning method to apply what we've previously learned and, as important, to build on the links we've already created in Section 0. As automation becomes a new planning sector, leadership continues to play an important role in managing a changing environment.

On the program's site, detailed information is requested.

3.2 Determine your automation needs

Statements of projected performance requirements. Advanced planning is already taking place to handle software requirements. In the Program part, we build on that foundation by focusing on the most important needs that allow for the most successful automation. The objective is to look beyond conventional criteria and typical testing depth/breadth, since automation may enable complicated and comprehensive system testing.

In many phases of development assessment, dreams arise from needs, and careful planning for frequent testing begins with developing brief but thorough objectives. The requirements for software testing will have to be gathered from several sources. It's unusual to come across a

Incomplete needs must be estimated, including expected operating systems, information verification features, hardware systems, and other environmental elements. The AST framework's provisions should also start to be created, and they will be examined in the next phases.

Whether automation will be utilized to create full tests, input data and expected outcomes, or just input data is a crucial aspect in designing and performing automated testing. Oracle testing may be done in a variety of methods (see the section on Determine Approach to Test Oracle). The majority of these rely on official models or code-verified validity in some way.

3.3 Prioritize requirements depending on the DEF

MITER was developed, and the Developmental Evaluation Framework (DEF) depicted in Figure 2 was authorized by DASD (DT&E) to provide a comprehensive approach to govern the testing system and better support adoption choices. This approach has been utilized throughout the Department of Defense and is directly applicable to the usage of AST. The DEF encourages a service deterioration process in which the flow principle moves from capacity to precisely specified but unmeasured demands. The Developmental Objectives of Development (DEO) are power, which we may convert into functions in the realm of software testing. System performance, cyber security, cooperation, dependability, and maintenance are among the topics covered in these

films. These objectives are replaced by substituting 'Decisions' for specific conditions of interest,' and the entries will be active test cases.

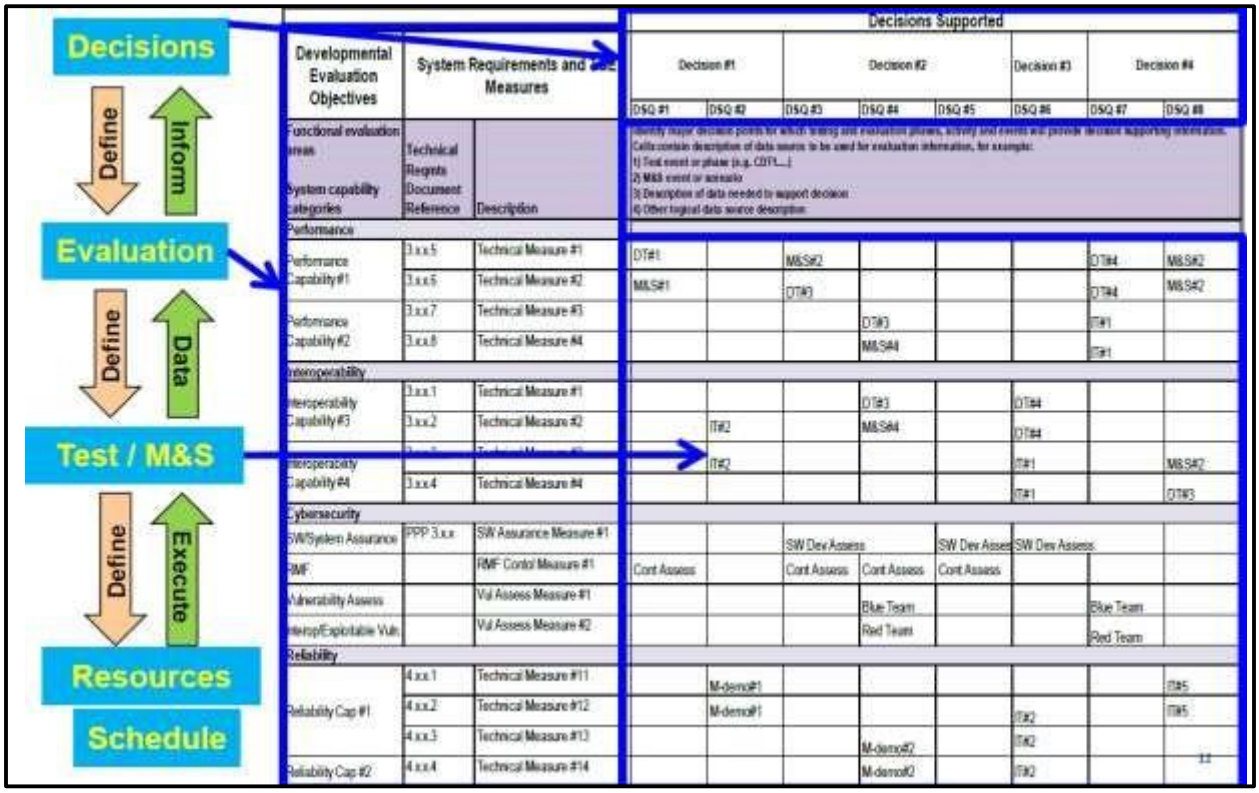

Figure 2. MITRE Developmental Evaluation Framework

The DEF process displays the resources available for each requirement and can assist in determining which automatic needs should be prioritized. Although it is intended that the most important skills will be gained, this does not guarantee that they will be converted into correct AST need priority. Automated risk has to be given more consideration. Depending on the requirement, automation might be difficult, and the implications of failure can be more serious than checking other conditions. These tradeoffs between value and risk should be assessed and included in a prioritized plan. It's important to pay attention to and weigh in on:

Participation in DEOs (which are ordered categories)

The chances of a default requirement failing are low due to a delay or a delay in enforcing it

3.4 Tools to identify and Compare

The amount and sorts of tools available for AST are often one-of-a-kinds. Each element of the automated testing procedure can require a different agency or collection of tools to be integrated with. The purpose of the Planning Phase is to gain a general understanding of the power of the correct tools and narrow down a few viable candidates that will work for your team, your schedule,

and the amount of automation difficulty you seek. A multi-tool solution will emerge from a complete assessment of demands.

Start with the tools that your team utilizes or is familiar with, and you'll be able to use them effectively for future AST. Due to the short learning curve and automatic improvement, these tools will be given top emphasis.

There are a few things to bear in mind while you go through this operation. Carry forward with the procedure. Look for possible tool solutions in the following areas:

Is it possible to describe the test types and distribute them across all application domains?

Is the tool written in a mainstream scripting language like VB, JavaScript, Python, or another?

Where can you undertake a spontaneous test or recovery?

What OS systems are compatible? Is it limited to web-based applications?

What is the most basic application, and how steep is the learning curve to improve machine performance?

Is it reasonable to expect the team to make excellent use of the tool?

Is it only capable of recapturing the GUI, or does it also support app (API) calls to the GUI? What are they?

3.5 Determine Automation Requirements

The term "automated software testing" has a wide range of meanings and definitions. AST can be somewhat complex depending on the system and manpower; it can take years for the testing team to develop a robust automation framework, or it can be quite simple when a few fundamental test processes are automated, such as utilizing Overnight or batch files. There should be a basic understanding of how things work. The Phase study will result in the development of AST. The level of discharge is usually the most crucial factor to consider. To put it another way, how does the system perform during testing? Live operators, visual messages, modeling, imitation, or a pre-recorded cable are all possibilities. Many other parts of testing will be influenced by this live, virtual, creative (LVC) environment.

3.6 Scripts for Tests in Outline

Creating test documents is a continuous process that begins with planning, continues with design, moves into implementation, and is done numerous times in the maintenance phase. The goal of planning is to explain the process of developing test documents and a few small task papers.

In this area, the most important responsibility is to request test scripts to determine if anything can be used or significantly modified to save time in the default time frame. Although there are no official DoD scripts for papers, they are likely to be tested by testers, and automatons from prior or existing AST programs will compose and document their work. Make sure you have access to all of the programs and resources available. the correct

Chapter 4: Proposed Automation

4.1 Design for Automation

The Test Plan lays the groundwork and prepares you for the design phase. This section attempts to give you more information about the automated software testing system and help you make important decisions about how the tests will be run. During the design phase, tasks are completed simultaneously and repeatedly rather than in order. As the design evolves, so does the thought, and numerous shapes can arise. Because technology requires a uniform AST structure in many functional domains, the entire design phase should be done in a highly collaborative atmosphere.

Design That Isn't Compliant: This part can be used to contribute to the content and functioning of thorough information reviews. This

4.2 What Should You Automate?

It should be apparent which of the most promising tests is the default after finishing the requirements definition, the development assessment framework, tool research, and interaction with test automation experts. The software development life cycle (unit testing/development, integration, performance, and performance) and test stages (DT, DT / OT, and OT) will affect decisions about what can and should be automated. The automation factor, the success of automation as required by other systems, the team's predicted skill, the possibility for test integration enhancement, and how often the test will need to be done manually are all factors to consider. Examine the assumptions and hazards that come with them. Select Tools

Phase 1 identified a potential set of tools for the anticipated AST. The preferred toolset should become clearer as the requirements and testing environment mature. Choose the right automated testing tools for the job. Additional research will be required to create an excellent choice of talent and application. Reference books, manuals, relevant lessons, other DoD AST experimental tools, vendor sites, and exhibitions are all good places to start. Devices that were previously unconsidered may now be added to the mix. There are additionally extremely effective DoD AST test tools available across the department, with considerable support and subject matter experts (SMEs): IDT's (Innovative Defense Technology) Auto

4.3 Choose your tools

Software developers, test engineers, software testers, and automated software testers make up the AST-enabled test team. An effective team includes current inspectors/engineers in automatons being trained and developed or employing employees (full-time or part-time workers) with those talents. The option to automate is generally based on changing demands, available resources (including time and money), home team competencies, and prospective expansion. To create internal automation abilities, the team requires time for formal training, self-education, communication, and training. Quick technology can be gotten "by borrowing" from self-employed neighbors, through contractor contracts, or equal means.

4.4 Create the capacity to automate

To build an adequate AST framework, a test team with the right AST skillsets and tools is required. The components necessary to test the SUT in an automated manner are referred to as a framework. For example, a simulation model that stimulates the SUT would allow automated methods to achieve the desired function. Consider the following client/slave configurations and hardware requirements: operating system selection, networking, and prospective cloud use, accounting for concurrent user needs, information assurance, and classification.

A set of AST tools is also included in the framework. The tools can monitor and control the entire testing process, as well as identify software failures.

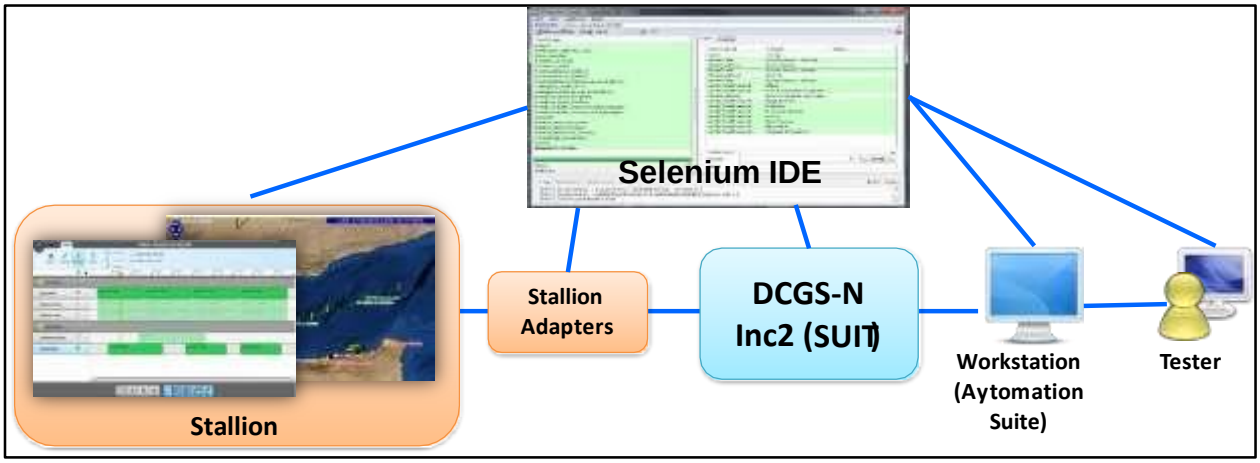

Figure 3. Notional DCGS-N Inc 2 automated test tool integration framework

4.5 Determine the platform and Framework for Automated Testing

Test cases should be "connected" to the provision to check input metrics because they are based on metrics of vital needs. A test case can cover a wide range of requirements, and multiple test cases can be used to illustrate a condition. Take note of whether or not the requirements are tested in the majority of test cases. The condition necessitated the use of a function or the strength of a SUT. Test scenarios should be straightforward to understand, and test cases may be required. Test cases should be simple to evaluate and adequately documented as needs evolve. Consider creating ‘smoke tests,' which combine several test cases to guarantee that important activities are carried out.

During the tool selection phase, many production tools should be tested. This can assist; nevertheless, most of the job will be done by others.

4.6 Create Test Cases and Scenarios

The percentage of SUT code lines utilized in the test program is referred to as reading. In the situation or cases of comprehensive testing, it can also be used to count the number of requirements that have been assessed. Coverage is sometimes defined as the number or percentage of possible usage cases or procedures tested in all feasible combinations. Determine the best approach to increase the installation of software operational conditions by identifying enough coverage and determining the best way to increase the installation of software operating conditions. To construct a test design, spend some time executing actions that explain SUT's expected performance. Among the possible strategies and methods are:

The focus of risk-based testing was on important workplaces with a high risk of failure.

Matrixes based on mathematics are based on the const

4.7 Choose a method for testing Oracle

One of the benefits of AST is that it can be done without a lot of human intervention. Tests can be run in the evenings, on weekends, or in the background while testers are working on other projects. During the 'Go/No Go' process, this flexibility would have been examined first. Teams must decide on the realistic and desired frequency of testing based on this flexibility. Are tests to be run after agile sprints (monthly), capability delivery (quarterly), or as Command Run On (CRON) operations overnight during sprints?

The duration of individual test cases and the permitted time can affect the flexibility of test timing. The length of the test is frequently determined by the subject.

4.8 Construct a Configuration Control Structure

Determine the requirements and the appropriate method for collecting the data generated by automated testing. Set up the data collection system, which could be as basic as a spreadsheet. Consider conducting prototype tests using negative testing (fault induction) to discover areas where more detail in the log and output files is required. Concentrate your efforts initially on locating the software fault and providing any relevant information that could aid in a root cause analysis. Determine the appropriate method for reducing, analyzing, and post-processing the data to gain the most information about SUT performance.

4.9 Conduct design review

The technique of deciding if a test has passed or failed is known as the test oracle. The oracle problem persists even with automated ways for generating input data and executing tests. Testing necessitates both test data and expected results for each data input. Because traditional procedures require substantial human involvement, this is usually the most expensive aspect of the testing effort. However, there are several approaches for automating some or all of the test oracle generation. The starting cost, amount of sophistication, and application domains vary. The following are examples of automated techniques of solving the oracle problem:

Chapter 5: Experimentation

5.1 Phase 3: Execute

In software, the term "execute" refers to the process of running a program. There is a job in AST where the scripts run to do the automated test. In this methodology, however, Phase 3 refers to gathering operations that enable a test to become automatic. This step encompasses the process of manually testing the system, building scripts to automate it, and lastly performing the automated test. The design phase includes purposeful redundancy, which sets the stage for the execution phase.

5.2 The system's Application by the Capture Operator

The test team should be conversant with how the SUT is used in real-world situations. An SME usually performs the most often performed functions as well as the tasks that are only performed on occasion. The automaton can then create a pixel- or code-based script (GUI versus API) that captures the mission threads' principal process flow, as well as all connected contingency paths.

In terms of the SUT's capacity to perform desired functions, the team should also focus on the most often executed paths and the greatest risk ones. The team should also look at the automation potential for these mission threads—just because they're important doesn't mean they're easy to automate.

5.3 Create Manual Tests

To begin, make sure that the manual test processes for the tests that will be automated are adequately and sufficiently documented. The test Automator will work alongside software test engineers on a set of manual tests that will be detailed in a step-by-step process. The Automator will produce a list of potential constructs for inclusion in the automated solution. Other testers should run the test manually with the Automator if possible, to ensure consistency and to look for alternative and new methods.

5.4 Decide on automated testing environments

The test team will have to weigh several test execution alternatives, some of which will be influenced by the SUT's software development lifecycle and maturity. There may be no choice as to where the test takes place: in a software integration lab (SIL); on a cloud server; via the web, Secured Internet Protocol Router Network (SIPRNet), or Joint Worldwide Intelligence Communication System (JWICS); in a controlled operationally representative environment/range; or in a fully operational environment (e.g., onboard a ship or aircraft). There could be a variety of ways to excite the system to test if it works. Live, simulated live (virtual), recorded life, and recorded live are the four stages of abstraction.

5.5 Integrate the Automated Test Framework with Tools

with the aid of

The framework should already incorporate the tools chosen throughout the design process. Prototype or simple test scripts can be used to test device communication. If there are issues with compatibility or inadequate communication and connection between software tools, contact the program developers/coders. Check that open-source tools, for example, work effectively with commercial parent tools.

Additional tools are frequently required as the automated solution and environment evolve to accomplish desired automation features. This is especially true as more and more output data is collected, posing problems to test teams in determining how to use it.

5.6 Create and improve automation Scripts

For the majority of the test cases, the design process resulted in draft automation scripts. The hands are finished through an iterative process in a highly collaborative setting, with the execution phase possibly hosted on the cloud. To effectively evaluate and validate the automated software test, peer thorough code reviews and continuous work with SMEs should be undertaken.

Scripts that were "stolen" from other users or a repository may require significant improvement. These products may appear to be plug-and-play at first glance, but they rarely are. For example, a Common Access Card is a common "borrowed" script for user authentication (CAC). Normally, this would necessitate the use of

5.7 Verify the results of the Automation Pilot

Phase 2's output is an estimate of how difficult it will be to automate the test cases that have been provided. To give the AST program the best chance of success, simpler test cases should be automated initially. Walkthrough the results output to the log file in a collaborative setting. These outputs can be screenshots or messages indicating if a step was completed successfully or not.

Negative testing of the automation is beneficial in ensuring that it detects input problems. Negative testing will also assist the team in determining whether the output can be manually parsed or written to the relevant files with proper formatting. The message architecture may change.

5.8 Carry out the Automation

The actual process of running the AST program is referred to as this execution task. The goal is to run the AST program without the involvement of the test team, either during off-hours or in the background, so that it does not interfere with other test activities. There may still be a need to monitor the AST if failures in the SUT occur, causing the automation to end prematurely, or if there are errors in the automation framework. It's also very uncommon for some programs to become "stuck" when hunting for the right image to move on to the next phase. Be wary of automation that necessitates manual input at intervals; in this case, the testing is not automated.

The team should seek ways to improve or fix the automation. Recognize the

5.9 Contingencies must be implemented

Even if the AST program is sound and runs well, there are always more conditions to test than time allows. There should have been a prioritized list of criteria during the design phase. It may be useful to look into contingencies from the baseline test case once the fundamental criteria have been covered in test cases.

Scaling for a larger number of users, running tests in diverse environments, and employing alternative hardware and operating systems are all examples of contingencies. In the SIL, for example, an automated test (GUI-based) worked properly, but when it was attempted elsewhere, it failed since the laptop used was older and had a lower screen resolution. Another important scenario is when the SUT fails.

Chapter 6: Analyze Output

6.1 Phase 4: Analyze

The analysis step converts the output from the performed test cases into decision quality information to evaluate SUT software performance and the usefulness of the AST framework. An analyst knowledgeable with scientific test and analysis techniques (STAT) and other applicable statistical methods may be required by the team of testers, software developers, software testers, and Automator.

6.2 The Data Output Format

A good AST program must include a complete description of everything that happened during execution. This is useful for a variety of reasons, including rapid fault separation, improved debugging, a thorough understanding of state transitions, and assurance that the AST framework is running successfully. Transparency into system status/execution phases, as well as traceability back to requirements, are critical qualities.

Detailed logs (perhaps color-coded to identify defects), images of system status upon failure, and data written to output files are examples of representative output products. The following items must be included in an output file:

The ID of the test case ID of the requirement

Steps and sequences for testing data

6.3 SUT Anomalies should be investigated

The purpose of software defect detection is to figure out what's wrong with the program. Software bugs, flaws, mistakes, and anomalies will arise; detecting the issue and determining the source of the discrepancy is critical. The output reports from a correctly built AST architecture should allow for speedy fault isolation. You'll have to take action if this isn't the case. Engage the services of an attorney. It's time to start a formal root cause analysis program. The 5 Whys method, cause-and-effect (Ishikawa) diagrams, and the car industry's 8-step process are all examples of approaches.

It's vital to figure out whether the abnormality is due to the SUT or the AST framework. As an example,

6.4 Requirements should be summarized Void have been Tested and Identified

The test log should be detailed enough to identify which requirements were put to the test. It should also include the amounts of the input variables, allowing the team to assess how well the test area was covered. Examine the system's requirements and identify any gaps or places that still require testing. For reporting on the requirements' fulfillment, the team should present preliminary data analysis findings and conclusions. Include conversations about the test space's coverage and the importance of automation to the overall test effort.

6.5 Check Repeatability and Reproducibility

In practice, it is typical to find that the same tool and test do not always produce the same findings. This is particularly true when it comes to GUI testing. The team will need to test the scripts' stability to determine if they consistently produce the same results. The term "repeatability" refers to a tester's ability to achieve consistent results when doing the same process. Variation in equipment is taken into account by reproducibility. For AST, repeatability is achieving consistent results from the same tool or suite of tools, whereas reproducibility means getting consistent results across numerous devices and solutions. Negative testing causes failures to see if the software works properly.

In the end, automation can be a source of frustration.

6.6 Calculate and keep track of automation Metrics

Estimates of automated test advantages were useful in making a ‘Go/No Go' decision in Phase 0. Now that the test program has been automated, the team requires an accurate, trustworthy, and realistic assessment of the effort's effectiveness. This statistic will be used to update the ROIs and guide future AST work. Some metrics aren't accurate.

Percentage of criteria that were tested

Increased the number of lines of code that were tested.

Coverage of predicted operating paths and use cases has been expanded.

It's time to put the automated test sequence to the test.

Time to build automation capability; time to manually perform

Rate of defect finding

In the evenings and on weekends, time is being tested.

Automated script reusability

In conclusion: Leaders

6.7 Compute ROIs and Think About considering Future AST Program

After making the first expenditure to attain AST capability, it's time to assess its value. The team has several choices. The following are a few of the more notable ones to consider:

Take the existing effort to a new level.

Increase the number of users ((

Automate the creation of new requirements.

Use a variety of tools

The switch from graphical to code-based API tests

Conclusion

We have presented a comprehensive overview of software testing concepts, techniques, and processes. In compiling the survey, we have tried to be comprehensive to the best of our knowledge, as matured in years of research and study of this fascinating topic The approaches overviewed include more traditional techniques, e.g., code-based criteria, as well as more modern ones, such as model checking or the recent XP approach. Two are the main contributions we intended to offer to the readers: on one side, by putting into a coherent framework all the many topics and tasks concerning the software testing discipline, we hope to have demonstrated that software testing is a very complex activity deserving the first-class role in software development, in terms of both resources and intellectual requirements. On the other side, by hinting at relevant issues and open questions, we hope to attract further interest from the academy and industry in contributing to evolving the state of the art on the many remaining open issues. In the years, software testing has evolved from an "art" [46] to an engineering discipline, as the standards, techniques, and tools cited throughout the chapter demonstrate. However, test practice inherently remains a trial-and-error methodology. We will never find a test approach that is guaranteed to deliver a "perfect" product, whichever is the effort we employ. However, what we can and must pursue is to transform testing from "trial-and-error" to a systematic, cost-effective, and predictable engineering discipline.

Appendix A: Acronym List

Acronym	Description	Acronym	Description
API	Application Programmer Interface	**LVC**	Live, Virtual, Constructive
AST	Automated Software Testing	**NMCI**	Navy-Marine Corps Internet
ATRT	Automated Test and retest	**OSD**	Office of the Secretary of Defense
CAC	Common Access Card	**OT**	Operational Test
CDRLs	Contract Data Requirement Lists	**POA&M**	Plan of Action and Milestones
CMMI	Capability Maturity Model Integration	**POC**	Point of Contact
COE	Center of Excellence	**RITE**	Rapid Integration and Test Environment
CRON	Command Run On	**RM**	Requirements Management
DASD(DT&E)	Deputy Assistant Secretary of Defense, Developmental Test, and Evaluation	**ROI**	Return on Investment
DEF	Developmental Evaluation Framework	**ROM**	Rough Order of Magnitude
DEO	Developmental Evaluation Objectives	**SIL**	Software Integration Lab
DI2E	Defense Intelligence Information Enterprise	**SIPRNet**	Secure Internet Protocol Router Network
DoD	Department of Defense	**SMEre**	Subject Matter Expert
DT	Developmental Testing	**SPACEWAR**	Space and Naval Warfare Systems Command
DT/OT	Developmental Testing/Operational Testing	**STAT**	Scientific Test and Analysis Techniques
FAA	Federal Aviation Administration	**SUT**	System(s) Under Test
FRACAS	Failure Reporting, Analysis, and Corrective Action System	**T&E**	Test & Evaluation
FRB	Failure Review Board	**TDD**	Test-Driven Development
GUI	Graphical User Interface	**TEMP**	Test and Evaluation Master Plan
HP	Hewlett Packard	**UFT**	Unified Functional Testing
IDT	Innovative Defense Technologies	**UML**	Unified Modeling Language
JML	Java Modeling Language	**VV&A**	Verification, Validation, and Accreditation

References

1. T. Ball, "The concept of dynamic analysis", *Proctor joint 7th ESTEC/7th ACM FSE*, Toulouse, France, vol.24, no. 6, October 1999, pp.: 216 – 234.
2. L. Baresi, and M. Young, "Test Oracles" Tech. Report CIS-TR-01- http://www.cs.uoregon.edu/~michal/pubs/oracles.html
3. R. Barták, "On-line Guide to Constraint Programming", Prague, http://kti.mff.cuni.cz/~bartak/constraints/, 1998,
4. F. Basanieri, A. Bertolino, and E.Marchetti, "The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects", *Proc. 5th Int. Conf. UML 2002*, Dresden, Germany.
5. V.R. Basili, and R.W. Selby, R.W. "Comparing the Effectiveness of Software Testing Strategies", *IEEE Trans. Software Eng.* Vol. 13, no.12, pp. 1278—1296 1987.
6. K. Beck *Test-Driven Development by Example,* Addison Wesley, November 2002 B. Beizer, *Software Testing Techniques* 2nd Edition, International Thomson Computer Press, 1990.
7. G. Bernot, M.C. Gaudel, and B. Marre, "Software Testing Based on Formal Specifications: a Theory and a Tool", *Software Eng. Journal*, vol. 6, pp. 387 405,1991.
    A. Bertolino, "Knowledge Area Description of Software Testing", Chapter 5 of SWEBOK: *The Guide to the Software Engineering Body of Knowledge*. Joint IEEE-ACM Software Engineering Coordination Committee. 2001. http://www.swebok.org/.
8. E.W. Dijkstra, "Notes on Structured Programming" *T.H. Rep. 70- WSK03 1970.* http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF
9. W. W. Peng, and D.R. Wallace, "Software Error Analysis", *NIST*
10. *SP 500-209, National Institute of Standards and Technology, Gaithersburg MD 20899,* http://hissa.nist.gov/SWERROR/ December 1993.
11. W. Perry, *Effective Methods for Software Testing*, Wiley 1995.
12. S.L. Pfleeger, *Software Engineering Theory, and Practice*, Prentice-Hall, 2001.
13. S. Rapps, and E.J. Weyuker, "Selecting Software Test Data Using Data Flow Information", *IEEE Trans. Software Eng.* vol.11, pp. 367—375, 1985.

14. G. Rothermel. and M.J. Harrold, “Analyzing Regression Test Selection Techniques”, *IEEE Transactions on Software Engineering,* vol. 22, no. 8, pp. 529 – 551, 1996.

15. TGV--Test Generation from transitions systems using Verification techniques http://www.inrialpes.fr/vasy/cadp/man/tgv.html