2016

Test Automation Strategy

ITSE 1391 Introduction to Software Testing

Elena Stefanova

Austin Community College

# Table of Contents

[Table of Contents 1](#_Toc467929013)

[1. THE AST STRATEGY DOCUMENT 3](#_Toc467929014)

[2. SCOPE AND AUTOMATED TEST OBJECTIVES 3](#_Toc467929015)

[a) Deciding Which Tests to Automate 4](#_Toc467929016)

[b) Don’t Try to Automate Everything at Once—Take It Step by Step 4](#_Toc467929017)

[c) Consider Budget, Schedule, and Expertise—Not Everything Can Be Tested or Automated 5](#_Toc467929018)

[d) Automate Test Requirements Based on Risk 5](#_Toc467929019)

[e) Probability of failure of critical path functionality 5](#_Toc467929020)

[f) Impact or business risk 5](#_Toc467929021)

[g) Complexity 5](#_Toc467929022)

[h) Analyze the Automation Effort 5](#_Toc467929023)

[i) Analyze the Reuse Potential of an Automated Module 5](#_Toc467929024)

[j) Focus Automation on Repetitive Tasks—Reduce the Manual Test Effort 6](#_Toc467929025)

[k) Prioritize—Base Automation Requirements on a Prioritized Feature Schedule3 6](#_Toc467929026)

[l) Defining the Test Objectives: An Example 6](#_Toc467929027)

[3. IDENTIFY THE APPROACH 7](#_Toc467929028)

[a. Designing and Developing the Test Cases 7](#_Toc467929029)

[b. Adapt Manual Procedures to Automation 8](#_Toc467929030)

[c. Automate Test Procedure Documentation and Test Code Generation 8](#_Toc467929031)

[1. Test Case Development Techniques 9](#_Toc467929032)

[4. AUTOMATED SOFTWARE TEST FRAMEWORK (ASTF) 9](#_Toc467929033)

[a) ASTF Requirements 9](#_Toc467929034)

[b) ASTF Architecture 10](#_Toc467929035)

[c) ASTF Design Components 10](#_Toc467929036)

[5. AST ENVIRONMENT/CONFIGURATION 10](#_Toc467929037)

[a) Test Configurations 11](#_Toc467929038)

[b) Other Test Environment Automated Testing Requirements 11](#_Toc467929039)

[c) Automating the Test Environment Management—Automated CM 12](#_Toc467929040)

[6. AUTOMATING THE RTM 12](#_Toc467929041)

[a) Require Standard Test Case Templates That Are Usable for Your Automation Framework 12](#_Toc467929042)

[b) Hyperlink the Test Cases 13](#_Toc467929043)

[c) Update the Test Case Steps to Include Pass/Fail Results 13](#_Toc467929044)

[d) Update the RTM to Include Pass/Fail Results 13](#_Toc467929045)

[7. AUTOMATED DEFECT TRACKING 13](#_Toc467929046)

[SUMMARY 13](#_Toc467929047)

[Bibliography 14](#_Toc467929048)

(notes, p. 17)

Developing an automated test strategy is like laying out a floor plan, based on the blueprint. Within the automation strategy, we define the scope, objectives, approach, test framework, tools, test environment, schedule, and personnel requirements related to the automated testing effort.

Here we will discuss the concept of an overall automation approach: what steps to take and what considerations are needed to derive a successful automated testing strategy. The test strategy should also be documented, as it will become the floor plan for implementing automated testing. Everyone involved with the project—all the stakeholders—should be able to gain a clear understanding of what is being automated and get an overall sense of how it will be implemented by reading the test strategy document.

.

# THE AST STRATEGY DOCUMENT

We touched on the need for initial automated test requirements planning; here we will refine this process further. Now that you have reviewed various design documents, developed prototypes when needed, gathered enough information to establish requirements, and understand the task at hand, it is time to develop an automation test strategy.

A typical test strategy is documented and may have the categories described here. Each category is further discussed in the sections listed.

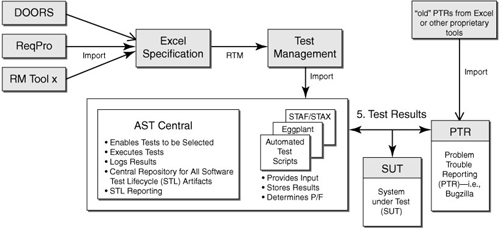
# SCOPE AND AUTOMATED TEST OBJECTIVES

The AST project scope and automated test objectives need to be clearly defined, documented, and communicated. The test objectives do not need to include detailed information regarding how the test is to be performed. A high-level description of what the AST effort is trying to accomplish is sufficient.

For example, we might decide that we want to automate the entire STL. Figure 1 provides an example of what this type of automation could look like.

Figure 1 shows an example implementation of the entire STL automation and support. For each testing lifecycle phase/tool, AST could include import/export from/to all ASTL tools to allow for integrated support of all STL testing phases, i.e., links from requirements management (RM) tools to test management (TM) tools to test automation (TA) tools (in order to track all automated and manual tests) and to problem trouble reporting (PTR) or defect reporting and tracking tools. The benefit of having an AST central portal is that all information is maintained in a central repository, tests can be run and monitored from a central location, outlines of PTRs can be generated for evaluation and then submitted/imported to the PTR tool; PTRs can be tracked in one place, and reporting across the testing lifecycle is provided. The RTM to summarize test requirements coverage and test pass/fail throughout the test program is automated as part of this effort. Details of RTM automation are discussed further in Section 6.6.

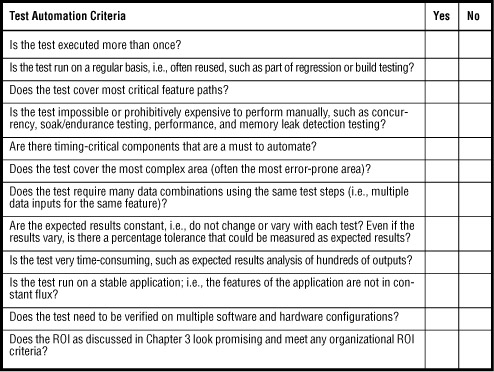
**Figure 1** Example of AST throughout the software testing lifecycle (STL) (Note: A direct import from the RM tool to the TM tool can take place, as preferred)



## Deciding Which Tests to Automate

Our experience shows that before defining what tests to automate, it is best to conduct an analysis using a criteria checklist that allows you to determine which tests are feasible to automate. Figure -2 shows a sample checklist that we have used on various projects to help decide whether test automation for a specific test objective is feasible or reasonable.[1](https://www.safaribooksonline.com/library/view/implementing-automated-software/9780321619600/ch06.html#ch06-1)

**Figure -2** Checklist for deciding what to automate



## Don’t Try to Automate Everything at Once—Take It Step by Step

Avoid trying to automate all test requirements at once. Take the example of a test team who decided to add all of their test requirements to the test management tool in support of system-level testing. The test team was eager to automate every possible test. As a result, they identified 1,900 test procedures that they planned to support by automation. When the time came to develop these test procedures, the test team discovered that the automated test tool was not as easy to use as they had thought. Expectations and objectives could not be met. As with all testing efforts, prioritization needs to take place.

## Consider Budget, Schedule, and Expertise—Not Everything Can Be Tested or Automated

The “what to automate” analysis needs to take budgets, schedules, and available expertise into account. Given limited resources (time and budget) and carved-in-stone deadlines, not all possible combinations and permutations of a program can be tested, and therefore not all possible tests can be automated. Additionally, some tests cannot be automated or it is simply cost-prohibitive to do so.

## Automate Test Requirements Based on Risk

When reviewing the defined test procedures for the purpose of determining which to automate, take a look at the highest-risk functionality and its related test requirements and analyze whether those requirements warrant AST priority. The test team should also review the test procedure execution schedule when selecting test procedures to automate since the schedule sequence should be generally based on risk and dependencies, among other issues.

## Probability of failure of critical path functionality

Assess the priority of the test requirement. Is this a feature path executed by the user often, or rarely (if at all)? The best rule is to rank test requirements from the most critical functionality down to the least critical functionality. Focus on the most often executed functional feature paths, those that absolutely have to work.

## Impact or business risk

What impact would a feature failure have on system operations and the end users’ ability to perform their jobs? Does the failure represent a potential liability for the company?

## Complexity

Analyze which functionality is complex to test manually and almost cost-prohibitive to set up and test in a manual fashion. For example, memory leak detection and performance testing are cumbersome or almost impossible to conduct as manual tests. However, a user could have a tool like BoundsChecker (a memory leak detection tool) running in the background during manual testing to catch memory leaks. Complexity of testing is an important context driver of risk.

## Analyze the Automation Effort

The suggestion that the initial automation effort should be based on the highest-risk functionality has one caveat. The highest-risk functionality is often the most complex and thus the most difficult to automate. Therefore, analyze the automation effort first.

## Analyze the Reuse Potential of an Automated Module

Our experience has shown that when determining what test procedures to automate, it is good to keep reuse in mind. Suppose that the test team decided to automate the highest-risk functionality of the application, without contemplating the level of effort required to automate the test procedures or considering the extent to which test scripts could be reused. If an automated test that required quite some effort to implement cannot be reused, did it really pay off to implement it (i.e., was it worth the effort.

## Focus Automation on Repetitive Tasks—Reduce the Manual Test Effort

It is beneficial to focus automation efforts on repetitive tasks. If these are automated, test engineers are freed up to focus on testing more complex functionality.

## Prioritize—Base Automation Requirements on a Prioritized Feature Schedule[3](https://www.safaribooksonline.com/library/view/implementing-automated-software/9780321619600/ch06.html#ch06-3)

Software feature implementations need to be prioritized for each incremental software release, in the case of iterative development, or depending on the development lifecycle model used, and based on customer needs or the need to deliver some high-risk items first. Automated test procedure planning and development can then be based on priority and risk in addition to the software feature implementation schedule, since both can dictate the order in which features will be made available for testing.

• Highest to lowest risk**:** It is important to consider the risks when developing a project schedule and the automated testing strategy. Focusing development of automated testing on the highest-risk features is one way to prioritize.

• Highest to lowest complexity: Like risk, prioritizing by complexity attempts to develop and test the most complex features first, thus minimizing schedule overruns.

• Customer needs: In most projects, feature delivery tends to be prioritized by customer needs, since this is usually required to fuel the marketing and sales activities associated with a product.

• Budget constraints: It’s important to consider the testing budget allotted when prioritizing the features for test automation for a given release. Some features will be more important to the success of a program than others.

• Time constraints: Consider the time constraints when prioritizing the features for a given release. Again, some features will be more important to the success of a program than others.

• Personnel constraints: Consider available personnel and associated expertise when prioritizing the feature schedule. Key personnel required to implement a specific feature might be missing from the team, due to budget constraints or other issues. While prioritizing test automation features it’s important to consider not only the “what” but the “who.”

## Defining the Test Objectives: An Example

Continuing with the testing of the DDS example used in [Chapter 5](https://www.safaribooksonline.com/library/view/implementing-automated-software/9780321619600/ch05.html), and after conducting our “what to automate” analysis, our test objectives may be identified as the following for the AUT:

• Automate testing of basic features of DDS as defined in test requirements 1.x through 5.x.

• Automate testing of enhanced features of DDS as defined in test requirements 6.x through 10.x.

• Automate testing of quality of service (QoS) parameters, per requirement *y*.

• Automate system performance testing and produce automated performance metrics and analysis.

• For phase I of the project, test only priority 1 test requirements (with 1 being the highest priority). For phase II of the project, test priority 2s, 3s, etc.

# IDENTIFY THE APPROACH

## Designing and Developing the Test Cases

In order to effectively define the testing approach you will need to start designing and developing the test cases, which will help define the details of how you plan to accomplish the test objectives. At times when it comes to AST, manual test procedures might already have been designed and developed; in that case you would have to evaluate the existing test procedures for automation.

**• Detailed description of test cases for each testing phase**, such as functional, performance, security, and concurrency; types of testing techniques applied (see “Test Case Development Techniques” later in this chapter for further discussion); along with expected results, method of determining expected results, analysis of actual versus expected results, analysis of test outcome if not specific results but a range of data that needs to be analyzed, and more

**• Reusable components** and project-specific features required in the ASTF (see the ASTF architecture discussion for more detail on this)

**• Type of test data to be used**; how the test data will be derived or acquired

**• Number and content of phases** (or spirals, or iterations) of deliveries

• Reporting requirements

**• Test procedure ID:** Use a naming convention when filling in the test procedure ID.

**• Test name: Provide** a longer description of the test procedure.

**• Date executed:** Fill in when the test procedure is executed.

**• Test engineer initials**: Provide the initials of the test engineer executing the test procedure.

• Test procedure author: Identify the developer of the test procedure.

• Test objective: Outline the objective of the test procedure.

• Related use case/requirement number: Identify the requirement identification number that the test procedure is validating.

• Precondition/assumption/dependency: Provide the criteria or prerequisite information needed before the test procedure can be run, such as specific data setup requirements.

• Verification method: This field may state certification, automated or manual test, inspection, demonstration, analysis, and any other applicable methods.

• User action: Here the goals and expectations of a test procedure are clearly defined. Completing this field allows for clarification and documentation of the test steps required to verify the use case.

• Expected results: Define the expected results associated with executing the particular test procedure.

• Actual result: This field may have a default value, such as “Same as expected result,” that is changed to describe the actual result if the test procedure fails.

• Test data required: List the set of test data required to support execution of the test procedure.

• Include standards for design.

• Adapt manual procedures to automation.

• Automate your test procedure documentation.

• Automate your test code generation.

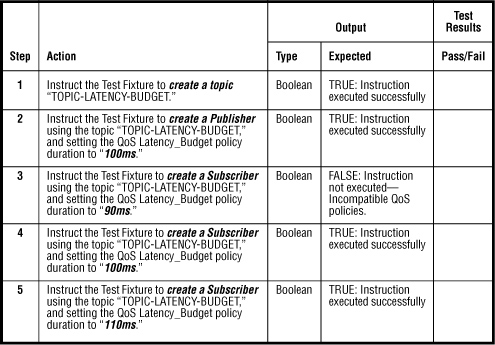
## Adapt Manual Procedures to Automation

An important point to note is that manual test procedures as written generally cannot straightforwardly be reused for automation. They generally need to be modified or adapted to allow for AST.

## Automate Test Procedure Documentation and Test Code Generation

We have developed an effective way to document and develop AST procedures as part of our “automating the test automation” paradigm; an example of this is provided here, using a GUI test.

**Figure 4** Automated test procedure example shows an example of binary steps in an automated test procedure. While the content looks similar to the manual test procedure, it has been generated in an automated fashion



# Test Case Development Techniques

As mentioned previously, a SUT cannot be 100% tested, nor can all tests be 100% automated. An analysis needs to take place and tests need to be prioritized as discussed previously in this chapter. Once the tests to be automated have been defined, testing techniques need to be applied to help narrow down a subset of test scenarios that if run still provide adequate testing coverage.

# AUTOMATED SOFTWARE TEST FRAMEWORK (ASTF)

Now that we have defined the AST scope and objectives, and understand the types of tests we want to automate, we are ready to develop our ASTF requirements.

As part of an ASTF you may want to be able to run various STL support tools; therefore, along with the ASTF requirements, automation test tool requirements need to be captured. These requirements describe what the new automation tool as part of ASTF needs to accomplish. They differ from the SUT requirements that describe the system being testing. These requirements describe the tool in support of the various testing phases, such as functional testing versus performance testing.

## ASTF Requirements

• Requirements management and associated RTM

• Test cases/procedures documentation, run, results reporting

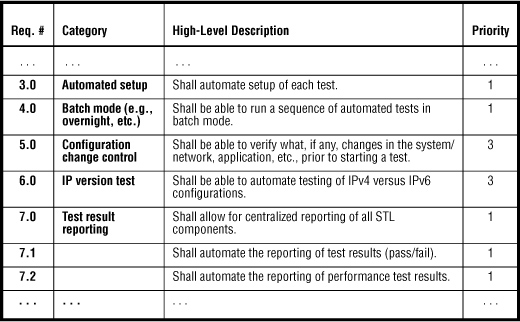
• Central control where all STL components are managed

• AUT

• Test results/defect tracking

• Configuration management

**Figure 5** Sample ASTF requirements



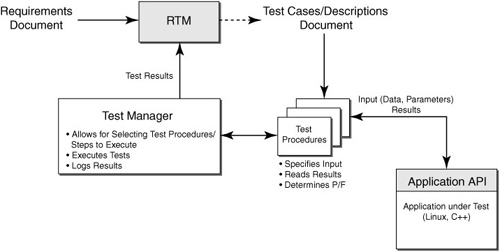
## ASTF Architecture

Along with a written description of the automated testing framework and related ASTF requirements, include an architecture diagram in the testing strategy document.

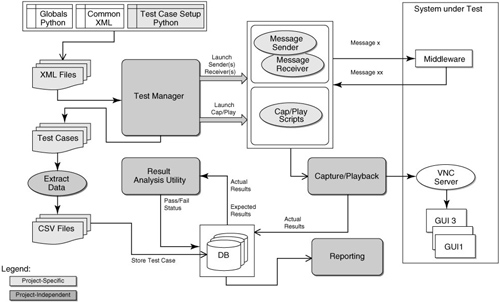
## ASTF Design Components

Once the high-level architecture diagram has been designed, the various components need to be defined, as part of a detailed software design document

**Figure 6** High-level architecture diagram



**Figure 7** High-level ASTF architecture design



# AST ENVIRONMENT/CONFIGURATION

Before evaluating tests to be automated, it is important to understand the test environment.

The automated testing environment should be separate from other environments, such as the development or production environments, and it should have the following characteristics:

• **Clean**

The environment should be isolated, secure, controlled, and stable. Changes to the environment, such as to hardware, software (including the automated test framework), network, parameters, etc., are monitored and known. In a clean environment the AUT will not need to be modified during the testing process.

**• Predictable and repeatable**

Automated tests in this clean environment should behave as expected, and automated tests should have the ability to be rerun with known results. If tests are not predictable or not repeatable, then investigate possible configuration changes or framework errors.

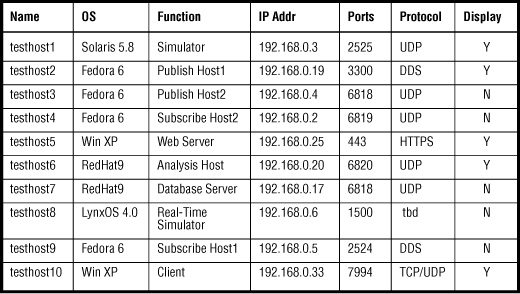
• **Functionally equivalent**

The environment should have functional components equivalent to those of the target environment. If target environment functionality cannot be met, then develop an approach that best mimics the component (e.g., use of simulators, etc.), but use extrapolation techniques sparingly.

• **Accounts for required test configurations**, as described next

## Test Configurations

Test configurations need to be understood in order to finalize the AST environment, and they need to be specified using well-known entities.



## Other Test Environment Automated Testing Requirements

This section of the test strategy document should list additional requirements, other than the hardware and software requirements, that may affect specific results of the automated testing effort. Other requirements might include, among others,

• Network constraints

• Specialized equipment

• Physical or environmental constraints

## Automating the Test Environment Management—Automated CM

Once the test environment requirements have been identified, they are ideally managed in an automated fashion.

• Only approved hardware and software are in place

• System settings and parameters are defined for the secure state (e.g., operator and user permissions, privilege settings for applications and systems code)

• No extraneous hardware or software has been introduced into the operational environment

• No required hardware or software has been removed

Automated test environment management can be set up differently depending on your particular environment and configuration. However, the general categories of data collection in the process are

• Hardware

• Software

• System

• Network

• Kernel

• Boot

• Volume management

• User

# AUTOMATING THE RTM

. This section describes how to automate the RTM, another component of our AST strategy.

The SUT and in a separate effort the test tool or ASTF requirements are captured, categorized, and prioritized. This section focuses on the automated RTM related to SUT requirements.

• Develop the RTM.

• Require standard test case templates that are usable for your automation framework.

• Hyperlink the test cases.

## Require Standard Test Case Templates That Are Usable for Your Automation Framework

Standardization of test case templates compatible with your automation framework can improve the efficiency of the overall test-case-building process.

## Hyperlink the Test Cases

Provide a mechanism to easily access the test cases within the RTM

## Update the Test Case Steps to Include Pass/Fail Results

As the test case steps are performed, develop a mechanism to automatically update the pass/fail status

## Update the RTM to Include Pass/Fail Results

To tie it all together, develop a mechanism within your ASTF to allow for automated update of the test case status in the RTM after a test run. Usually all steps in the detailed test case step need to “pass” in

1. Import/write requirements in your IDE environment (e.g., Eclipse, ANT).

2. Map the unit tests to requirements as the code and unit tests are developed.

The tool helps developers map their code to specific requirements and also helps ensure that all requirements are actually covered. The tool accomplishes requirements coverage by

• Creating or importing a requirements file or files

• Associating the requirements file or files with the project

• Updating the requirements file or files to map the unit tests to the requirements

When the unit tests are run next, the tool can generate the requirements coverage report. The report can be configured to show

• Requirements successfully covered by code

• Requirements that are not covered

• Broken requirements—associated unit test(s) failed or tests did not execute

# AUTOMATED DEFECT TRACKING

As part of our ASTF we also will have selected a defect tracking tool that will be integrated as part of the framework.

# SUMMARY

This chapter introduced and defined the test strategy document and covered the sections that should be considered as part of one. An important aspect of the test strategy is identifying the approach to satisfy the overall test objectives. The AST strategy outlined will serve as your floor plan to implement your AST efforts. Once the test objectives are understood, it is important to decide on an overall testing framework, which can include coverage and automation of the entire STL, or simply automating a subset of tests. It can include various tools and automation capabilities, whether vendor-provided, open-source, or in-house-developed, such as requirements management, test management, unit, integration, and system testing, configuration management, automated testing frameworks, and defect tracking. Understanding your AST scope, and selecting the correct tool for the task at hand, is imperative, and in this chapter we described a how-to approach.

# Bibliography

. Adapted from Dustin, *Effective Software Testing*.

2. Adapted from Dustin et al., *Automated Software Testing*.

3. Adapted from Dustin, *Effective Software Testing*.

4. Dustin et al., *Automated Software Testing*, Section 7.3.4.

5. A. Page, K. Johnston, and B. Rollison, *How We Test Software at Microsoft* (Microsoft Press, 2008).

6. FIT is a tool developed to help with the writing of automated acceptance testing.

7. See [http://safsdev.sourceforge.net/  
FRAMESDataDrivenTestAutomationFrameworks.htm](http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm).

8. See <http://staf.sourceforge.net/>.

9. [www.incose.org/productspubs/products/setools/tooltax/  
reqtrace\_tools.html](http://www.incose.org/productspubs/products/setools/tooltax/reqtrace_tools.html).

10. [www.technobuff.net/webapp/product/showProduct.do?name=jfeature](http://www.technobuff.net/webapp/product/showProduct.do?name=jfeature)