The Automated Testing Life-cycle Methodology (ATLM)[i]

Elfriede Dustin

Software project managers and software developers building today's applications face the challenge of doing so within an ever-shrinking schedule and with minimal resources. As part of their attempt to do more with less, organizations want to test software adequately, but as quickly and thoroughly as possible. To accomplish this goal, organizations are turning to automated testing.

Faced with this reality and realizing that many tests cannot be executed manually, such as simulating 1,000 virtual users for volume testing, software professionals are introducing automated testing to their projects. While needing to introduce automated testing, software professionals may not know what's involved in introducing an automated test tool to a software project, and they may be unfamiliar with the breadth of application that automated test tools have today. The Automated Testing Life-cycle Methodology (ATLM), depicted in Figure 1, provides guidance in these areas.

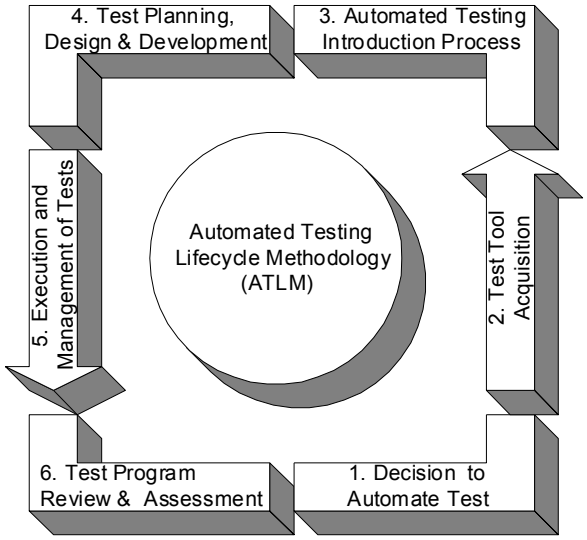

Figure 1 - Automated Test Lifecycle Methodology (ATLM)

By using the systematic approach outlined within the ATLM, organizations are able to organize and execute test activities in such a way as to maximize test coverage within the limits of testing resources. This structured test methodology involves a multi-stage process, supporting the detailed, and inter-related activities that are required to introduce and utilize an automated test tool; develop test design; develop and execute test cases; develop and manage test data and the test environment; as well as document, track and obtain closure on issue/trouble reports.

Clearly, the emphasis on automated testing represents a paradigm change for the software industry. This change does not simply involve the application of tools and the performance of test automation. Rather, it is pervasive across the entire test life cycle and the system development life cycle. The ATLM implementation takes place in parallel with the system development lifecycle. For software professionals to successfully make the leap to automated testing, structured approaches to testing must be embraced. The ATLM is revolutionary in the fact that it promulgates a new structured, building-block approach to the entire test life cycle, which enables software test professionals to approach software testing in a methodical and repeatable fashion.

The growth of automated test capability has stemmed in large part from the growing popularity of the iterative and incremental development life cycle, a software development methodology that focuses on minimizing the development schedule while providing frequent, incremental software builds. The objective of this incremental and iterative development is to engage the user and the test team early throughout design and development of each build so as to refine the software, thereby ensuring that it more closely reflects the needs and preferences of the user and thus addressing the riskiest aspects of development in early builds

In this environment of continual changes and additions to the software through each software build, software testing takes on an iterative nature itself. Each new build is accompanied by a considerable number of new tests as well as rework to existing test scripts, just as there is rework on previously released software modules. Given the continual changes and additions to software applications, especially web applications, automated software testing becomes an important control mechanism to ensure accuracy and stability of the software through each build.

The ATLM, invoked to support test efforts involving automated test tools, incorporates a multi-stage process. The methodology supports the detailed and inter-related activities that are required to decide whether to acquire an automated testing tool. The methodology includes the process of how to introduce and utilize an automated test tool, covers test development and test design, and addresses test execution and management. The methodology also supports the development and management of test data and the test environment, and addresses test documentation to include problem reports.

The ATLM methodology represents a structured approach, which depicts a process with which to approach and execute test. This structured approach is necessary to help steer the test team away from the common test program mistakes below.

- Implementing the use of an automated test tool without a testing process in place resulting in an ad-hoc, non-repeatable, non-measurable test program.
- Implementing a test design without following any design standards, resulting in the creation of test scripts that are not repeatable and therefore not reusable for incremental software builds.
- Efforts attempting to automate 100% of test requirements, when tools or in-house developed automated test harnesses being applied do not support automation of all tests required.
- Using the wrong tool or developing a too elaborate in house test harness.
- Test tool implementation initiated too late in the application development life cycle, not allowing sufficient time for tool setup and test tool introduction process (i.e. learning curve).
- Test engineer involvement initiated too late in the application development life cycle resulting in poor understanding of the application and system design, which results in incomplete testing.

The Automated Test Life-cycle Methodology (ATLM) is comprised of six primary processes or components. Each primary process is further composed of subordinate processes as described below.

1 Decision to Automate Test

The *Decision to Automate Test* represents the first phase of the ATLM. This phase covers the entire process that goes into the automated testing decision. During this phase it is important for the test team to manage automated testing expectations and to outline the potential benefits of automated testing when implemented correctly. A test tool proposal needs to be outlined, which will be helpful in acquiring management support.

1.1 Overcoming False Expectations for Automated Testing

While it has been proven successfully that automated testing is valuable and can produce a return on investment, there isn't always an immediate payback on investment. It is important that some of the misconceptions that persist in the software industry are addressed and that the automated testing utopia is managed. What follows is a list of just a few of the misconceptions that need to be addressed. It is important to note that people often see test automation as a silver bullet and, when they find that test automation requires a significant short-term investment of time and energy to achieve a long-term return on investment (ROI) of faster and cheaper regression testing (for example), the testing tool often becomes "shelf-ware" the tool. This is why it is important that in order to introduce automated testing correctly into a project, expectations are managed.

Automatic Test Plan Generation

Currently, there is no commercially available tool that can automatically create a comprehensive test plan, while also supporting test design and execution.

Throughout a software test career, the test engineer can expect to witness test tool demonstrations and review an abundant amount of test tool literature. Often the test engineer will be asked to stand before a senior manager or a small number of managers to give a test tool functionality overview. As always the presenter must bear in mind the audience. In this case, the audience may represent individuals with just enough technical knowledge to make them enthusiastic about automated testing, while not aware of the complexity involved with an automated test effort. Specifically, the managers may have obtained information about automated test tools third hand, and may have reached the wrong interpretation of the actual capability of automated test tools.

What the audience at the management presentation may be waiting to hear, is that the tool that you are proposing automatically develops the test plan, designs and creates the test procedures, executes all the test procedures and analyzes the results automatically. You meanwhile start out the presentation by informing the group that automated test tools should be viewed as *enhancements to manual testing*, and that automated test tools will not automatically develop the test plan, design and create the test procedures and execute the test procedures.

Soon into the presentation and after several management questions, it becomes very apparent just how much of a divide exists between the reality of the test tool capabilities and the perceptions of the individuals in the audience. The term *automated test tool* seems to bring with it a great deal of wishful thinking that is not closely aligned with reality. An automated test tool will not replace the human factor necessary for testing a product. The proficiencies of test engineers and other quality assurance experts will still be needed to keep the testing machinery running. A test tool can be viewed as an additional part of the machinery that supports the release of a good product.

Test Tool Fits All

Currently not one single test tool exists that can be used to support all operating system environments.

Generally, a single test tool will not fulfill all the testing requirements for an organization. Consider the experience of one test engineer encountering such a situation. The test engineer, Dave, was asked by a manager to find a test tool that could be used to automate the testing of all of the department's applications. The department was using various technologies to include mainframe computers and Sun workstations; operating systems such as Windows 3.1, Windows 95 and Windows NT, Windows 2000, programming languages such as VC++, and Visual Basic, other client server technologies and web technologies, such as DHTML, XML, ASP, etc.

After conducting a tool evaluation, the test engineer determined that the tool of choice was not compatible with the VC++ third party add-ons (in this case, Stingray grids). Another tool had to be brought in, that was compatible with this specific application.

It is important to note that expectations have to be managed and it has to be made clear that currently there does not exist one single tool on the market that is compatible with all of the operating systems and programming languages simultaneously. More than one tool is required to test the various technologies.

Immediate Reduction in Schedule – An automated test tool will not immediately minimize the testing schedule.
Another automated test misconception is the expectation that the use of an automated testing tool on a new project will immediately minimize the test schedule. Since the testing effort actually increases, as previously described, the testing schedule will not experience the anticipated decrease at first, and an allowance for schedule increase is required, when initially introducing an automated test tool. This is due to the fact that when rolling out an automated test tool, the current testing process has to be augmented or an entirely new testing process has to be developed and implemented. The entire test team and possibly the development team needs to become familiar with this new automated testing process (i.e. ATLM) and needs to follow it. Once an automatic testing process has been established and effectively implemented, the project can expect to experience gains in productivity and turn around time that have a positive effect on schedule and cost.

1.2 Benefits of Automated Testing

The previous discussion points out and clarifies some of the false automated testing expectations that exist. The test engineer will also need to be able to elaborate on the benefits of automated testing, when automated testing is implemented correctly and a process is followed. The test engineer must evaluate whether potential benefits fit required improvement criteria and whether the pursuit of automated testing on the project is still a logical fit, given the organizational needs. There are three significant automated test benefits (in combination with manual test) which include a) producing a reliable system, b) improving the quality of the test effort, and c) reducing test effort and minimizing schedule.

Many return on investment case studies have been done with regard to the implementation of automated testing. One example is a research effort conducted by imbus GmbH (see www.imbus.de). They conducted a test automation value study in order to collect test automation measurements with the purpose of studying the benefits of test automation versus the implementation of manual test methods. Their research determined that the break-even point of automated testing is on average at 2.03 test runs (see www.imbus.de for more detail)[ii].

1.3 Acquiring management support

Whenever an organization tries to adopt a new technology, they encounter a significant effort when determining how to apply it to their needs. Even with completed training, organizations wrestle with time-consuming false starts before they become capable with the new technology. For the test team interested in implementing automated test tools, the challenge is how to best present the case for a new test automation technology and its implementation to the management team.

Test engineers need to influence management's expectations for the use of automated testing on projects. Test engineers can help to manage expectations of others in the organization by forwarding helpful information to the management staff. Bringing up test tool issues during strategy and planning meetings can also help develop better understanding of test tool capabilities of everyone involved on a project or within the organization. A test engineer can develop training material on the subject of automated testing and can advocate to management that a seminar be scheduled to conduct the training.

The first step in moving toward a decision to automate testing on a project requires that the test team adjust management understanding of the appropriate application of automated testing for the specific need at hand. The test team, for example, needs to check early on whether management is cost adverse and

would be unwilling to accept the estimated cost of automated test tools for a particular effort. If so, test personnel need to convince management about the potential return on investment by conducting cost benefit analysis.

If management is willing to invest in an automated test tool, but is not able or willing to staff a test team with individuals having the proper software skill level or provide for adequate test tool training, the test team will need to point out the risks involved and/or may need to reconsider a recommendation to automate test.

Management also needs to be made aware of the additional cost involved when introducing a new tool, not only for the tool purchase, but for initial schedule/cost increase, additional training costs and for enhancing an existing testing process or implementing a new testing process.

Test automation represents highly flexible technology, which provides several ways to accomplish an objective. Use of this technology requires new ways of thinking, which only amplifies the problem of test tool implementation. Many organizations can readily come up with examples of their own experience of technology that failed to deliver on its potential because of the difficulty of overcoming the "Now what?" syndrome. The issues that organizations face when adopting automated test systems include those outlined below.

- Finding/hiring test tool experts
- Using the correct tool for the task at hand
- Developing and implementing an automated testing process, which includes developing automated test design and development standards
- Analyzing various applications to determine those which are best suited for automation
- Analyzing the test requirements to determine the ones suitable for automation
- Training the test team on the automated testing process, automated test design, development and execution.
- Initial increase in schedule and cost

2 Test Tool Acquisition

Test Tool Acquisition represents the 2nd phase of the ATLM. This phase guides the test engineer through the entire test tool evaluation and selection process, starting with confirmation of management support. Since a tool should support most of the organizations' testing requirements, whenever feasible, the test engineer will need to review the systems engineering environment and other organizational needs and come up with a list of tool evaluation criteria. A review of the different types of tools available to support aspects of the entire testing life-cycle is provided in the book Automated Software Testing (as part of the ATLM), enabling the reader to make an informed decision with regard to the types of tests to be performed on a particular project. The test engineer then needs to define an evaluation domain to pilot the test tool. Finally, after all those steps have been completed, the test engineer can make vendor contact to bring in the selected tool(s). Test personnel then evaluate the tool, based on sample criteria provided.

3 Automated Testing Introduction Phase

The *Process of Introducing Automated Testing* to a new project team represents the 3rd phase of the ATLM. This phase outlines the steps necessary to successfully introduce automated testing to a new project, which are summarized below.

Test Process Analysis
Test Process Analysis ensures that an overall test process and strategy are in place and are modified, if necessary, to allow automated test to be introduced in a successful fashion. The test engineers define and collect test process metrics in order to allow for process improvement. Here test goals, objectives and strategies need to be defined and test process needs to be documented and communicated to the test team. In this phase, the kinds of testing applicable for the technical environment will be defined and tests are defined that can be supported by automated tools.
During the test process analysis, techniques get defined. Best practices, such as conducting performance testing during the unit-testing phase are laid out.
Plans for user involvement are assessed, and test team personnel skills are analyzed against test requirements and planned test activities. Early test team participation is emphasized, supporting refinement of requirement specifications into terms, which can be adequately tested while also supporting test team understanding of application requirements and design.

Test Tool Consideration
The Test Tool Consideration phase includes steps that investigate whether incorporation of automated test tools that have been brought into the company without a specific project in mind now would be beneficial to a specific project, given the project testing requirements, available test environment and personnel resources, the user environment, platform, and product features of the application under test. Schedule is reviewed to ensure sufficient time for test tool setup and development of requirements hierarchy; potential test tools and utilities are mapped to test requirements; test tool compatibility with the application and environment is verified; workaround solutions are investigated to incompatibility problems surfaced during compatibility tests.

4 Test Planning, Design and Development

Test Planning, Design and Development is the 4th phase of the ATLM. These subjects are summarized below.

Test Planning
The Test Planning phase represents the need to review long lead-time test planning activities. During this phase, the test team identifies test procedure creation standards and guidelines; hardware, software and network required to support test environment; test data requirements; a preliminary test schedule; performance measure requirements; a procedure to control test configuration and environment; defect tracking procedure and associated tracking tool.

The test plan contains the results of each preliminary phase of the structured test methodology (ATLM). The test plan will define roles and responsibilities, project test schedule, test planning and design activities, test environment preparation, test risks and contingencies, and acceptable level of thoroughness (i.e. test acceptance criteria). Test plan appendices may include test procedures, naming convention description, test procedure format standards and a test procedure traceability matrix.

The Test Environment Setup is part of test planning. It represents the need to plan, track and manage test environment set up activities, where material procurements may have long lead-times. The test team needs to schedule and track environment set up activities; install test environment hardware, software and network resources; integrate and install test environment resources; obtain/refine test databases; and develop environment setup scripts and test bed scripts.

Test Design

The Test Design component addresses the need to define the number of tests to be performed, the ways that test will be approached (paths, functions), and the test conditions which need to be exercised. Test design standards need to be defined and followed.

An effective test program, incorporating the automation of software testing, involves a mini-development life cycle of its own, complete with strategy and goal planning, test requirement definition, analysis, design and coding. Similar to software application development, test requirements must be specified before test design is constructed. Test requirements need to be clearly defined and documented, so that all project personnel will understand the basis of the test effort. Test requirements are defined within requirement statements as an outcome of test requirement analysis.

After Test Requirements have been derived using the described techniques, test procedure design can begin. Test procedure definition consists of the definition of logical groups of test procedures and a naming convention for the suite of test procedures. With a test procedure definition in place, each test procedure is then identified as either an automated or a manual test. During the test planning phase the test team gets an understanding of the number of test techniques being employed and an estimate for the number of test procedures that will be required. The test team also will have an estimate of the number of test procedures that will need to be performed manually, as well as with an automated test tool.

Much like a software development effort, the test program must be mapped out and consciously designed to ensure that test activities performed represent the most efficient and effective tests for the system under test. Test program resources are limited, yet ways of testing the system are endless. Test design is developed, which portrays the test effort, in order to give project and test personnel a mental framework on the boundary and scope of the test program.

Following test analysis, the test team develops the test program design models. The first of these design models, the Test Program Model, consists of a graphic illustration that depicts the scope of the test program. This model typically depicts the test techniques required to support the dynamic test effort and also outline static test strategies.

Having defined a test program model, the test team constructs a test architecture, which depicts the structure of the test program, and defines the way that test procedures will be organized in support of the test effort.

The next step in the test procedure design process as depicted in Table 1, is to identify those test procedures which stand out as being more sophisticated, and as a result, are required to be defined further as part of detailed test design. These test procedures are flagged and a detailed design document is prepared in support of the more sophisticated test procedures. Following detailed test design, test data requirements are mapped against the defined test procedures. In order to create a repeatable, reusable process for producing test procedures, the test team needs to create a document that outlines test procedure design standards. Only when these standards are followed can the automated test program achieve real efficiency and success, by being repeatable and maintainable.

STEP	DESCRIPTION
1	**Test Architecture Review**. The test team reviews the test architecture in order to identify the test techniques which apply.
2	**Test Procedure Definition (Development Level)**. A Test Procedure Definition is constructed at the development test level, that identifies the test procedure series that applies for the different design components and test techniques.
3	**Test Procedure Definition (System Level)**. A Test Procedure Definition is constructed at the system test level, that identifies the test procedure series that

	applies for the different test techniques.
4	**Test Procedure Design Standards**. Design standards are adopted and a naming convention is adopted that uniquely identifies the test procedures on the project from test procedures developed in the past or on other projects.
5	**Manual Versus Automated Tests**. Test procedures will be depicted as being either performed manually or as part of an automated test.
6	**Test Procedures Flagged for Detailed Design**. Test procedures that stand out as more sophisticated are flagged. These test procedures are further defined as part of detailed test design.
7	**Detailed Design**. Those test procedures flagged as part of step above, are designed in further detail within a Detailed Test Design file or document. Test procedure detailed design may consist of pseudocode of algorithms, preliminary test step definition, or pseudocode of test automation programs.
8	**Test Data Mapping**. Test Procedure Matrix is modified to reflect test data requirements for each test procedure.

Table 1 - Test Procedure Design Process

The exercise of developing the test procedure definition not only aids in test development, but this definition also helps to quantify or bound the test effort. The development of the test procedure definition involves the identification of the suite of test procedures that will need to be developed and executed in support of the test effort. The design exercise involves the organization of test procedures into logical groups and the definition of a naming convention for the suite of test procedures.

At the system level, it may be worthwhile to develop a detailed test design for sophisticated tests. These tests might involve test procedures that perform complex algorithms, consist of both manual and automated steps, and test programming scripts that are modified for use in multiple test procedures. The first step in the detailed design process is to review the test procedure definition at the system test level. This review is conducted for the purpose of identifying those test procedures that stand out as being more sophisticated, which as a result, are required to be defined further as part of detailed test design.

Detailed test design may take the form of test program pseudocode, when test programming is required. The detailed design may be represented simply as a sequence of steps that need to be performed in support of a test. When programming variables and multiple data values are involved, the detailed design may reflect the programming construct of a loop supporting an iterative series of tests involving different values together with a list or table identifying the kinds of data or ranges of data required for the test.

Following the performance of detailed test design, test data requirements need to be mapped against the defined test procedures. Once test data requirements are outlined, the test team needs to plan the means for obtaining, generating or developing the test data.

The structure of the test program (test architecture) is commonly portrayed in two different ways. One test procedure organization method involves the logical grouping of test procedures with the system application design components, and is referred to as a design-based test architecture. Another method represents a test technique perspective and associates test procedures with the various kinds of test techniques represented within the test program model, and is referred to as a technique-based test architecture.

An understanding of test techniques is necessary when developing test design and the test program design models. Personnel performing test need to be familiar with the test techniques associated with the white box and black box test approach methods. White box test techniques are aimed at exercising software program internals, while black box techniques generally compare the application under test behavior

against requirements that address testing via established, public interfaces such as the user interface or the published application programming interface (API).

Test Development

In order for automated tests to be reusable, repeatable and maintainable, test development standards need to be defined and followed.

After performing test analysis and design the test team is now ready to perform test development. Keep in mind that the test design and development activities follow an iterative and incremental approach, in order to address the highest risk functionality up front. Table 2 correlates the development process phases to the test process phases. The testing processes and steps outlined in the table are strategically aligned with the development process and the execution of these steps results in the refinement of test procedures at the same time as developers are creating the software modules. Automated and/or manual test procedures are developed during the integration test phase with the intention of reusing them during the system test phase.

Many preparation activities need to take place, before test development can begin. A test development architecture is developed (Figure 2), which provides the test team with a clear picture of the test development preparation activities or building blocks necessary for the efficient creation of test procedures. The test team will need to modify and tailor the sample test development architecture in order to reflect the priorities of their particular project. Part of these setup and preparation activities include the need to track and manage test environment set up activities, where material procurements may have long lead-times. Prior to the commencement of test development, the test team also needs to perform analysis to identify the potential for reuse of already existing test procedures and scripts within the Automation Infrastructure (reuse library).

The test team needs to develop test procedures according to a test procedure development/execution schedule. This schedule needs to allocate personnel resources and reflect development due dates, among other factors. The test team needs to monitor development progress and produce progress status reports. Prior to the creation of a complete suite of test procedures, the test team performs a modularity relationship analysis. The results of this analysis help to incorporate data dependencies, plan for workflow dependencies between tests, and identify common scripts that can be repeatedly applied to the test effort. As test procedures are being developed, the test team needs to ensure that configuration control is performed for the entire test bed to include test design, test scripts, and test data, as well as for each individual test procedure. The test bed needs to be baselined using a configuration management tool.

Test development involves the development of test procedures that are maintainable, reusable, simple and robust, which in itself can be as challenging as the development of the application-under-test. Test procedure development standards need to be in place supporting structured and consistent development of automated tests. Test development standards can be based on the scripting language standards of a particular test tool. For example, Rational's Robot uses SQABasic, a Visual Basic like scripting language and therefore the script development standards could be based on the Visual Basic development standards, outlined in a number of books on the subject.

Usually internal development standards exist that can be followed if the organization is developing in a language similar to the tool's scripting language. The adoption or slight modification of existing development standards is generally a better approach than creating a standard from scratch. If no development standards exist within the organization for the particular tool scripting language, it is important for the test team to develop script development guidelines. Such guidelines can include directions on *context independence*, which addresses the particular place where a test procedure should start and where it should end. Additionally, modularity and reusability guidelines need to be addressed.

PHASE	DEVELOPMENT PROCESS	TEST PROCESS
Module (Unit) Development	Design module from requirements	Perform test planning and test environment set up.
	Code module	Create test design and develop test data.
	Debug module	Write test scripts or record test scenario using module.
	Unit test module	Debug automated test script by running against module. Use tools that support unit testing,
	Correct defects	Rerun automated test script to regression test as defects are corrected.
	Conduct Performance Testing	Verify system is scaleable and will meet performance requirements
Integration	Build system by connecting modules. Integration test connected modules. Review trouble reports.	Combine unit test scripts and add new scripts that demonstrate module inter-connectivity. Use test tool to support automated integration testing.
	Correct defects and update defect status.	Rerun automated test script as part of regression test, as defects are corrected.
	Continued Performance Testing Activities	Verify system is scaleable and will meet performance requirements
System Test	Review trouble reports.	Integrate automated test scripts into system level test procedures, where possible, and develop additional system level test procedures. Execute system test and record test results.
	Correct defects and update defect status.	Rerun automated test script as part of regression test as defects are corrected.
Acceptance Test	Review incident reports.	Perform subset of system test as part of demonstration of user acceptance test.
	Correct defects	Rerun automated test script as part of regression test as defects are corrected.

Table 2 - Development / Test Relationship

By developing test procedures based on development guidelines, the test team creates the initial building blocks for an Automation Infrastructure. The Automation Infrastructure will eventually contain a library of common, reusable scripts. Throughout the test effort and in future releases, the test engineer can make use of the Automation Infrastructure in order to support reuse of archived test procedures, minimize duplication, and thus enhance the entire automation effort.

Test Development Architecture

Test team members responsible for test development need to be prepared with the proper materials. Test team personnel need to follow a test development architecture that includes, for example, a listing of the test procedures assigned to them and a listing of the outcome of automated vs. manual test analysis. Also, test team personnel will need to decide when to automate. At times a test team might want to avoid

automating using a GUI testing tool before the interface –whether API, character UI, or GUI is stabilized to avoid re-engineering the automated tests in response to non-bug-related changes. Other times the test team can find workaround solutions when automating an unstable GUI, such as focusing automation on the known stable parts only.

The test engineer needs to adhere to the test procedure development and execution schedule, test design information, automated test tool user manuals, and test procedure development guidelines. Armed with the proper instructions, documentation and guidelines, test engineers will have the foundation that allows them to develop a more cohesive and structured set of test procedures. It is important to note that the ability for the test team to repeat a process and repeatedly demonstrate a strong test program, depends upon the availability of documented processes and standard guidelines such as the test development architecture.

An example of the graphical illustration, containing the major activities to be performed as part of the test development architecture, is depicted in Figure 2. Test development starts with test environment setup and preparation activities, discussed previously. Once they are concluded, the test team needs to make sure that pertinent information, necessary to support development has been documented or gathered. The test team will need to modify and tailor the sample test development architecture, depicted in Figure 2, in order to reflect the priorities of their particular project. Note that Figure 2 should be read from bottom to top.

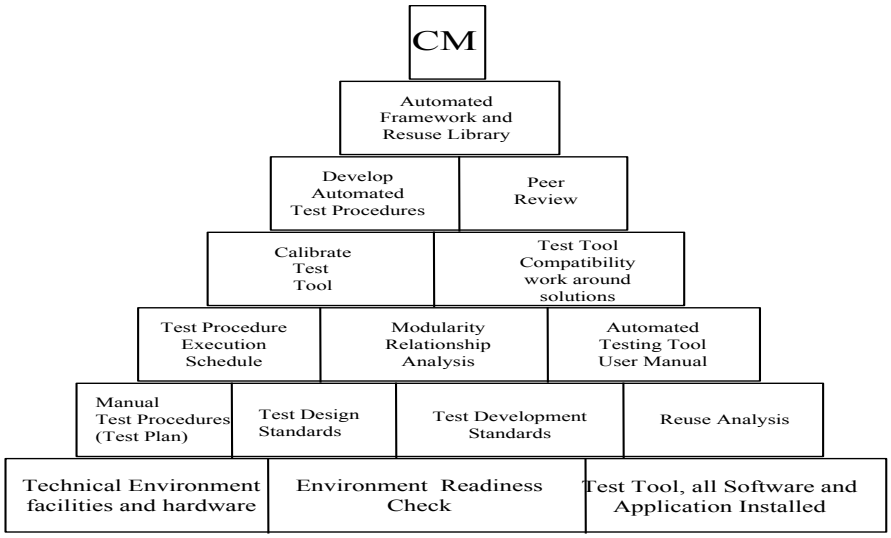

Figure 2 - Building Blocks of the Test Development Architecture

Technical Environment

Test procedure development needs to be preceded by several setup activities. The test development activity needs to be supported by a technical environment, which facilitates the development of test procedures. As a result, the test environment needs to be set up and ready to go. The test environment includes the technical environment, which may include facility resources as well as the hardware and software necessary to support test development and execution. The test team needs to ensure that there are enough workstations to support the entire team. The various elements of the test environment need to be outlined within the test plan, as discussed previously.

Environment setup activities can also include the use of an environment setup script to load test data restore a drive image, and to calibrate the test tool to the environment. When test tool compatibility

problems arise with the application under test, workaround solutions have to be identified. When developing test procedures it is important that the schedule for developing test procedures is consistent with the test execution schedule. It is also important that the test team follow test procedure development guidelines.

The test team will need to ensure that the proper test room or laboratory facilities are reserved and setup. Once the physical environment is established, the test team will need to ensure that all necessary equipment is installed and operational. The test plan defined the required technical environment and addressed test environment planning. Also within the test environment section of the test plan, the test team has already identified operational support required to install and checkout the operational readiness of the technical environment. The test team needs to ensure that operational support activities have been properly scheduled and must monitor progress of these tasks.

Specific tasks and potential issues outlined in the test plan should have now been addressed and resolved. Such issues could include network installation, network server configuration and allocated disk space, network access privileges, required desktop computer processing speed and memory, number and types of desktop computers (clients), video resolution requirements, and any additional software required to support the application such as browser software. Automated test tools that apply should have been scheduled for installation and checkout. These tools now should be configured to support the test team and be operational within the test environment.

The test environment setup activity includes the need to track and manage test environment set up activities, where material procurements may have long lead-times. These activities include the need to schedule and track environment set up activities; install test environment hardware, software and network resources; integrate and test install test environment resources; obtain/refine test databases; and develop environment setup scripts and test bed scripts.

The hardware supporting the test environment must be sufficient to ensure complete functionality of the production application. Test environment hardware needs to be sufficient to support performance analysis. In cases where the test environment utilizes hardware resources, which are also supporting other development or management activities, special arrangements may be necessary during actual performance testing. During system test, the software configuration loaded within the test environment must be a complete, fully integrated release with no patches and no disabled sections. The hardware configuration supporting the test environment needs to be designed to support processing, storage, and retrieval activities, which may be performed across a local or wide area network, reflecting the target environment.

The test environment design will also need to consider stress testing requirements. Stress and load tests may require the use of multiple workstations that will run multiple test procedures simultaneously, while some automated test tools include a virtual user simulation functionality that greatly eliminates or minimizes the need for multiple workstations.

Test data will need to be obtained with enough lead-time to support refinement and manipulation to support test requirements. Data preparation activities include the identification of conversion data requirements, the preprocessing of raw data files, loading of temporary tables possibly in a relational database management system format, and the performance of consistency checks. Identifying conversion data requirements involves performing in-depth analysis on data elements, which includes defining data mapping criteria, clarifying data element definitions, confirming primary keys, and defining data-acceptable parameters.

During test planning, the test team defined and scheduled the test environment activities. Now the team will need to track the test environment set up activities. Resources need to be identified to install

hardware, software and network resources into test environment and integrate and test installed test environment resources. The test environment materials and the application under test need to be baselined within a configuration management tool. Additionally, test environment materials may include test data and test processes.

The test team will need to obtain and modify test databases necessary to exercise software applications, and develop environment setup scripts and test bed scripts. The test team should perform product reviews and validation of all test source materials. The location of the test environment for each project or task should be defined within the test plan for each project. Early identification of the test site is critical to cost effective test environment planning and development.

5 Execution and Management of Tests

The test team at this stage, has addressed test design and test development. Test procedures are now ready to be executed in support of exercising the application-under-test. Also test environment setup planning and implementation was addressed consistent with the test requirements and guidelines provided within the test plan.

With the test plan in hand and the test environment now operational, it is time to execute the tests defined for the test program. When executing test procedures, the test team will need to comply with a test procedure execution schedule, as discussed previously. The test procedure execution schedule implements the strategy defined within the test plan. Plans for unit, integration, system and user acceptance testing are executed. Together, these testing phases make up the steps that are required to test the system as a whole.
The various steps involved during execution and management of tests are outlined below.

When executing test procedures, the test team will need to comply with a test procedure execution schedule. Following test execution, test outcome evaluations are performed and test result documentation is prepared.

Plans for unit, integration, system and user acceptance testing are executed, which together make up the steps that are required to test the system as a whole. During the unit test phase, code profiling can be performed. Traditionally, profiling is a tuning process that determines whether an algorithm is inefficient or a function is being called too frequently. Profiling can discover instances where there is improper scaling of algorithms, instantiations and resource utilization.

Integration testing is performed which focuses on the application internals. During integration testing, units are incrementally integrated and tested together based upon control flow. Since units may consist of other units, some integration testing, also called module testing, may take place during unit testing.

During system test, the test engineer is testing the integration of parts, which comprise the entire system. A separate test team usually performs system-level tests. The test team implements the test procedure execution schedule and the system test plan.

The test team also performs analysis to identify particular components or functionality that are experiencing a greater relative number of problem reports. As a result of this analysis, additional test procedures and test effort may need to be assigned to the components. Test results analysis can also confirm whether executed test procedures are proving to be worthwhile in terms of identifying errors.

Each test team needs to perform problem-reporting operations in compliance with a defined process. The documentation and tracking of software problem reports is greatly facilitated by an automated defect-tracking tool.

The test team manager is responsible for ensuring that tests are executed according to schedule, and test personnel are allocated and redirected when necessary to handle problems that arise during the test effort.

In order to perform this oversight function effectively, the test manager needs to perform test program status tracking and management reporting.

Test metrics provide the test manager with key indicators of the test coverage, progress, and the quality of the test effort. During white box testing the test engineer measures the *depth of testing*, by collecting data relative to path coverage and test coverage. During black box testing, metrics collection focuses on the breadth of testing to include the amount of demonstrated functionality and the amount of testing that has been performed.

6 Test Program Review and Assessment

Test Program review and assessment activities need to be conducted throughout the testing life-cycle, in order to allow for continuous improvement activities. Throughout the testing life-cycle and following test execution activities, metrics need to be evaluated and final review and assessment activities need to be conducted to allow for process improvement.

The various steps necessary for test program review and assessment are outlined below.

Following test execution, the test team needs to review the performance of the test program in order to determine where improvements can be implemented to improve the test program performance on the next project. This test program review represents the final phase of the Automated Test Life-cycle Methodology (ATLM).

Throughout the test program, the test team collected various test metrics. The focus of the test program review includes an assessment of whether the application satisfies acceptance criteria and is ready to go into production. The review also includes an evaluation of earned value progress measurements and other metrics collected.

The test team needs to adopt, as part of its culture, an ongoing iterative process of lessons learned activities. Such a program encourages test engineers to take the responsibility to raise corrective action proposals immediately, when such actions potentially have significant impact on test program performance.

Throughout the entire test life cycle, it is good practice to document and begin to evaluate *lessons learned* at each milestone. The metrics that are collected throughout the test life-cycle and especially during the test execution phase help pinpoint problems that need to be addressed.

Lessons learned, metrics evaluations and corresponding improvement activity or corrective action need to be documented throughout the entire test process in a central repository that is easily accessible.

After collecting lessons learned and other metrics, and defining the corrective actions, test engineers also need to assess the effectiveness of the test program to include an evaluation of the *Test Program Return on Investment*. Test engineers capture measures of the benefits of automation realized throughout the test life-cycle in order to support this assessment.

Test teams can perform their own surveys to inquire about the potential value of process and tool changes. A sample survey form is provided that can be used to solicit feedback on the potential use of requirement management tools, design tools, and development tools. Surveys are helpful to identify potential misconceptions, and gather positive feedback.

Summary

ATLM is a structured methodology that is geared toward ensuring successful implementation of automated testing. The ATLM approach mirrors the benefits of modern rapid application development efforts, where such efforts engage the user early in the development cycle. The end-user of the software

product is actively involved throughout analysis, design, development and test of each software build, which is augmented in an incremental fashion.

The ATLM incorporates a multi-stage process consisting of six components. This methodology supports the detailed and inter-related activities that are required to decide whether to acquire an automated testing tool. The methodology includes the process of how to introduce and best utilize an automated test tool, and addresses test planning, analysis, design, development, execution and management. The ATLM requires that the scope of the test program be outlined within the test plan, as a top-level description of test approach and implementation. The scope is further formulated through the definition of test goals, objectives and strategies, and with the definition of test requirements.

Similar to software application development, test requirements are specified before test design is constructed. Likewise, the test program must be mapped out and consciously designed to ensure that test activities performed represent the most efficient and effective tests for the application under test. Test design is developed, graphically portraying the test effort, in order to give project and test personnel a mental framework on the boundary and scope of the test program.

Rational Corporation lists the Automated Software Testing book and the (ATLM) as recommended reading as part of the Rational Unified Process. Many universities and professional software test training organizations have adopted the book for their classroom. Many companies (such as Imbus GMBH, Moehrendorf Germany) have adopted the book and ATLM as their company standard for automated software testing. Others believe that industry automated test tool vendors will soon be incorporating the book's structured methodology within their tools. Instead of performing the entire test life cycle haphazardly, software test managers will use an ATLM-compliant test tool that automatically supports (and possibly enforces) the book's sound building block approach to the test effort.

Elfriede Dustin may be contacted via her website at www.autotestco.com

---

[i] Adapted from "Automated Software Testing", Elfriede Dustin, et all, Addison Wesley Longman, Inc. July 1999

[ii] Linz, T., Daigl, M. GUI Testing Made Painless. Implementation and Results of the ESSI Project Number 24306. 1998. www.imbus.de.