**Chapter II: Introduction to Automaton**

**Introducing the Adaptive (Automated) Life Cycle Management (ALM)**

Today's applications face the challenge of doing required tasks and perform within an ever-shrinking schedule and with minimal resources. Hence Project stake holders attempt to do more with less, because organizations want to test software as quickly and thoroughly as possible. Hence organizations are turning to automated testing for Applications, Systems and Other Hardware resources.

This is reality that many (most of the) tests cannot be executed manually all the time, for example simulating 1, 0000 virtual users for volume testing, automated testing is introduced for such projects. But stakeholders may not know what's involved in introducing an automated tool to a software project, and also unfamiliar with the breadth of application that automated tools have.

But by defining systematic approach within the ALM, test activities can be organized and executed in such a way as to maximize test coverage within the limits of testing resources. Structured test methodology involves a multi-stage process, supporting the detailed and interconnected activities that are required to introduce and utilize an automated test tools and Techniques:

1. Develop test design and test cases.
2. Develop and execute test cases.
3. Develop, manage and arrange test data and the test environment.
4. Document, track, and obtain closure on issue/trouble reports.

Automated testing represents a paradigm change not only in the software industry but also in manufacturing Industry. This change doesn't simply involve the application of test automation tools. Rather, it have the entire test lifecycle and the system development lifecycle. The ALM implementation takes place in parallel with the system (software or any product) development lifecycle. Software professionals (engineers) to make a successful leap to automated adaptive testing, they must embrace structured approaches to testing. The ALM is revolutionary as it promulgates a new structured, building-block approach to the entire test lifecycle, which enables software professionals and stakeholders to approach software testing and Automation in a methodical and repeatable fashion.

The growth of adaptive capability has stemmed in large part from the growing popularity of the iterative and incremental development lifecycle, a software development methodology that focuses on minimizing the development schedule while providing frequent, incremental software builds. The objective of incremental and iterative development is to engage the user early throughout the design and development of each build in order to refine the software, and ensuring that it more closely reflects the needs and preferences of the user and thus addressing the required aspects of development in early builds.

In the environment of continuous changes and updates to the software through each software build, software testing itself takes on an iterative nature. Next build is accompanied by a considerable number of additional tests as well as rework to existing test scripts, or in software modules. Given the continual changes and additions to software applications, especially Web applications, adaptive software testing becomes an important control mechanism to ensure accuracy and stability of the software through each build of application.

The ALM, invoked to support automation efforts involving automated test tools, incorporates a multi-stage process. This supports the detailed and interrelated activities that are required to decide about Tools and Methods required for project. The methodology includes the process of how to utilize an automated test tool, covers test development and test design, and addresses test execution and management. It also supports the development and management of test data and the test environment, and addresses test documentation to include problem reports.

It represents a structured approach to execute automated testing. It is necessary to help the team away from these common test (White Box, Black Box) program mistakes:

1. Starts the use of an automated test tool without a testing process in place, resulting in an ad hoc, non-repeatable, non-measurable test program goes away from adaptive approach
2. Starts Implementation of 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.
3. Doing Attempt to automate 100% of testing requirements, when tools or in-house–developed automated test harnesses do not support automation of all tests required.
4. Using the wrong tool or developing elaborated in-house test harness.
5. Including test tool implementation too late in the Application-Development Lifecycle, therefore not allowing sufficient time for tool setup and test tool introduction process (learning curve) which should be of Adaptive nature.
6. Initiating Automation Test engineer involvement too late in the application-development lifecycle, resulting in poor understanding of the application and system design, which results in incomplete automation testing.

The Adaptive (Automated) Lifecycle Methodology (ALM) comprises following components:

1. Take Decision to Automate in Quality Assurance, Quality Engineering and Build & Release.
2. Getting requited Tools Acquisition for Analysis.
3. Doing Automated Introduction Process.
4. Working with Planning, Design, and Development.
5. Defining and Execution and Management of Process.
6. Finalize Program Review and Assessment.

**Following sections describe process, and subordinate processes contained within each primary process**

## Phase 1: Decision to Automation

The decision to automation represents the first phase of the ALM. This phase covers the entire process that goes into the automation decision. During this phase, it's important for the team to manage automation expectations and to outline the potential benefits of automation when implemented correctly. A tool proposal needs to be outlined, which will be helpful in acquiring management support.

### **Overcoming False Expectations for Automated Testing**

While it has been proven that automation is valuable and can produce a successful return on investment, there isn't always an immediate payback on investment. It's important to address some of the misconceptions that persist in the industry and to manage the automation utopia. There is a misconceptions.

Automation needs a significant short-term investment of time and energy to achieve a long-term return on investment (ROI) of faster and cheaper regression.

#### **Automatic Test Plan Generation**

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

Throughout a software career, the engineer can expect to witness tool demonstrations and review an abundant amount of tool literature. Often the engineer will be asked to stand before one or more senior managers to give a tool functionality overview. In this case, the audience may represent individuals with just enough technical knowledge to make them enthusiastic about automation, while unaware of the complexity involved with an automation effort. Specifically, the managers may have obtained secondhand information about automated tools, and may have reached the wrong interpretation of the actual capability of automated tools.

Management may expect that the tool being proposed automatically develops the plan, designs and creates the procedures, executes all the procedures, and analyzes the results automatically. Meanwhile, informing the group that automated tools should be viewed as *enhancements to manual process*, and that automated tools will not develop the plan, design and create the procedures, or execute the procedures.

Shortly into the presentation and after several management questions, it becomes very apparent just how much of a divide exists between the reality of the tool capabilities and the perceptions of the individuals in the audience. The *automated tool* seems to bring with it a great deal of wishful thinking that's not closely aligned with reality.

An automated tool will not replace the human factor necessary for testing a product. The proficiencies of engineers and other QA experts will still be needed to keep the test machinery running. An automation tool can be viewed as an additional part of the machinery that supports the release of a good product.

#### One Test Tool Fits All

**A single tool cannot be used to support all operating system environments.**

Generally, a single tool will not fulfill all the requirements for an organization/team. Consider the experience of one engineer encountering such a situation. The engineer is asked by a manager to find a tool that could be used to automate the all the department's applications. The department using various technologies including mainframe computers and Sun workstations; operating systems such as; programming languages such as C,C++, Java, Python other client/server technologies; and Web technologies such as DHTML, XML, ASP, and so on.

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

#### **Immediate Reduction in Schedule**

**An automated tool will always not minimize the testing schedule.**

Another automation misconception is the expectation that the use of an automated tool on a new project will immediately minimize the schedule. The schedule will not experience the anticipated decrease at first, and an allowance for schedule increase is required when initially introducing an automated tool. This is due to the fact that when rolling out an automated tool, the current process has to be augmented or an entirely new process has to be developed and implemented.

The entire team and possibly the development team, needs to become familiar with this new automated process (such as ALM) and needs to follow it. Once an automatic process has been established and effectively implemented, the project can expect to experience gains in productivity and turnaround time that have a positive effect on schedule and cost.

### **Benefits of Automated Testing, QA, and Continues Integration/Delivery**.

The previous discussion points out and clarifies some of the false automated expectations that exist. The engineer will also need to be able to elaborate on the true benefits of automation, when automated testing is implemented correctly and a process is followed. The engineer must evaluate whether potential benefits fit required improvement criteria and whether the pursuit of automation on the project is still a logical fit, given the organizational needs. There are three significant automation benefits:

* Producing a reliable system.
* Improving the quality of the effort.
* Reducing effort and minimizing schedule.

Many return on investment case studies have been done with regard to the implementation of automation.

### **Acquiring Management Support**

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

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

The first step in moving toward a decision to automation on a project requires that the team should be able to show to management understanding of the appropriate application of automation for the specific need at hand. For example, the team needs to check early on whether management is cost-averse and would be unwilling to accept the estimated cost of automated tools for a particular effort. If so, executive need to convince management about the potential return on investment by conducting cost/benefit analysis.

If management is willing to invest in an automated tool, but is unable or unwilling to staff a team with individuals having the proper software skill level or to provide for adequate tool training, the team needs to point out the risks involved and/or may need to reconsider a recommendation to automation.

Management 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 process or implementing a new process.

Automation represents 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 tool implementation. The issues that organizations face when adopting automated systems include those outlined below:

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

## Phase 2: Automation Tool Acquisition

Tool acquisition represents the second phase of the ALM. This phase guides the engineer through the entire 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 engineer will need to review the system's engineering environment and other organizational needs and come up with a list of tool evaluation criteria enabling the reader to make an informed decision with regard to the types of tests to be performed on a particular project. The engineer then needs to define an evaluation domain to pilot the tool. Finally, after all those steps have been completed, engineer can make vendor contact to bring in the selected tool(s). Then tool is evaluated, with sample criteria provided.

## Phase 3: Automation (Adaptive) Introduction Process

The process of introducing automation to a new project team constitutes the third phase of the ALM. This phase outlines the steps necessary to successfully introduce automation to a new project, which are summarized in the following sections.

### Process Analysis

Process analysis ensures that an overall process and strategy are in place and are modified, if necessary, to allow automation to be introduced in a successful fashion. The 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 team. In this phase, the kinds of process applicable for the technical environment will be defined, that can be supported by automated tools.

During the process analysis, techniques are defined. Plans for user involvement are assessed, and team personnel skills are analyzed against requirements and planned activities. Early team participation is emphasized, supporting refinement of requirement specifications into terms that can be adequately analyzed while also supporting team understanding of application requirements and design.

### **Tool Consideration**

The tool consideration process includes steps that investigate whether incorporation of automated 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 requirements, available environment, personnel resources, user environment, platform, and product features of the application under execution. Schedule is reviewed to ensure sufficient time for tool setup and development of requirements hierarchy; potential tools and utilities are mapped to requirements, tool compatibility with the application and environment is checked, and workaround solutions are found for incompatibility problems.

## Phase 4: Automation Planning, Design, and Development

Planning, design, and development is the fourth phase of the ALM. These subjects are summarized in the following sections.

### **Automation Planning**

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

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

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

### **Automation Design**

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

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

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

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

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

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

The next step in the procedure design process (see Table 1) is to identify those procedures that stand out as being more sophisticated, and as a result are required to be defined further as part of detailed design. These procedures are flagged and a detailed design document is prepared in support of the more sophisticated procedures.

Following detailed design, data requirements are mapped against the defined procedures. To create a repeatable, reusable process for producing procedures, the team needs to create a document that outlines procedure design standards. Only when these standards are followed can the automated program achieve real efficiency and success, by being repeatable and maintainable.

**Table 1 Automation Procedure Design Process**

|  |  |
| --- | --- |
| Step | Description |
| 1 | **Architecture Review**. The team reviews the architecture in order to identify the techniques that apply. |
| 2 | **Procedure Definition (Development Level)**. A procedure definition is constructed at the development level, identifying the procedure series that applies for the various design components and techniques. |
| 3 | **Procedure Definition (System Level)**. A procedure definition is constructed at the system level, identifying the procedure series that applies for the various techniques. |
| 4 | **Procedure Design Standards**. Design standards are adopted and a unique naming convention is adopted that distinguishes the procedures on the project from procedures developed in the past or on other projects. |
| 5 | **Manual Versus Automated Tests**. Procedures is depicted as being either performed manually or as an automated test. |
| 6 | **Test Procedures Flagged for Detailed Design**. Test procedures which stand out as more sophisticated are flagged. These test procedures then defined as part of detailed design. |
| 7 | **Detailed Design**. Those procedures flagged as part of step 7 are designed in further detail within a detailed design file or document. Detailed design consist of pseudo-code of algorithms, preliminary test step definition, or pseudo-code of test automation programs. |
| 8 | **Data Mapping**. Procedure matrix is changed to reflect test data requirements for each test procedure. |

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

At the system level, it may be useful to develop a detailed design for sophisticated tests. This might involve 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 is to review the procedure definition at the system test level. This review is conducted for the purpose of identifying those procedures that stand out as being more sophisticated and that, as a result, are required to be defined further as part of detailed test design.

Detailed design may take the form of program pseudo-code, when programming is required. The detailed design may be represented simply as a sequence of steps that need to be performed in support of an execution. 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 execution involving different values, together with a list or table identifying the kinds of data or ranges of data required for the execution.

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

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

An understanding of execution techniques is necessary when developing design and the program design models. Personnel performing need to be familiar with the techniques associated with the white box and black box test-approach methods.

*White box* test techniques are focused at exercising software program internals; *black box* techniques compare the application under test behavior against requirements that address testing via established public interfaces such as the UI or the published API.

### **Automation Development**

For automation be reusable, repeatable, and maintainable, development standards need to be defined and followed.

After performing analysis and design, the team is now ready to perform development.

Keep in mind that the 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 phases to the process phases. The automation 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 procedures at the same time wen software developers are creating the software modules. Automated and/or manual test procedures are defined during the integration phase with the intention of reusing them during the system test phase.

**Table 2 Development/Test Relationship**

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

Many preparation activities need to take place before development can begin. A development architecture is developed (described in the next section), which provides the team with a clear picture of the development preparation activities or building blocks necessary for the efficient creation of test procedures.

The team will need to tailor the sample development architecture to reflect the priorities of their particular project. Part of these setup and preparation activities involves the need to track and manage environment set up activities, where material procurements may have long lead times. Prior to the commencement of development, the team also needs to perform analysis to identify the potential for reuse of existing procedures and scripts within the automation infrastructure (reuse library).

The team needs to develop procedures according to development/execution schedule. And schedule needs to allocate resources and reflect development due dates, among other factors. The team needs to monitor development progress and produce progress status reports. Prior to the creation of a complete suite of procedures, the team performs a modularity relationship analysis.

The results of this analysis help to incorporate data dependencies, plan for workflow dependencies between executions, and identify common scripts that can be applied repeatedly to the effort. As procedures are being developed, the team needs to ensure that configuration control is performed for the entire bed to include design, scripts, and data, as well as for each individual procedure. The bed needs to be baselined using a configuration management tool.

Test, QA, Integration, Deployment development involves the development of procedures that are maintainable, reusable, simple, and robust, which in itself can be as challenging as the development of the application under execution. Development standards should be in place supporting structured and consistent development of automated execution. Development standards can be based on the scripting language standards of a particular tool.

The adoption of existing system 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's important for the team to develop script development guidelines. Such guidelines can include directions on *context independence*, which addresses the particular place where a procedure should start and where it should end. By developing procedures based on development guidelines, the team creates the initial building blocks for an automation infrastructure.

The automation infrastructure will store a library of common, reusable scripts. Throughout the effort and in future releases, the engineer can make use of the automation infrastructure to support reuse of archived procedures, minimize duplication, and thus enhance the entire automation effort.

### **Automation Development Architecture**

Team members responsible for product development need to be prepared with the proper materials. Team member need to follow an architecture that includes, for example, a listing of the procedures assigned to them and a listing of the outcome of automated versus manual analysis. Also, team personnel need to decide when to automate. At times a test team might want to avoid automating using a GUI tool before the interface, whether API, character UI, or GUI, is stabilized, to avoid having to reengineer the automation in response to non–bug-related changes. At other times, the team might find workaround solutions when automating an unstable GUI, such as focusing automation on the known stable parts only.

The engineer needs to adhere to the procedure development and execution schedule, test design information, automated tool user manuals, and procedure development guidelines. Occupied 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. Repeating a process and repeatedly demonstrating a strong program depends on the availability of documented processes and standard guidelines such as the test development architecture.

### **Technical Environment**

This activity needs to be supported by a technical environment, which facilitates the development of procedures. As a result, the environment needs to be set up and ready to go. This environment includes the technical environment, which may include facility resources as well as the hardware and software necessary to support test, development and execution. It needs to be ensured that there are enough workstations to support the entire team.

Environment setup activities can also include the use of an environment setup script to load data or restore a drive image, and to calibrate the tool to the environment. When tool compatibility problems arise with the application under execution, workaround solutions have to be identified. When developing procedures, it's important that the schedule for developing procedures is consistent with the execution schedule. It's also important that the team follow procedure development guidelines.

The team must ensure that the proper test room or laboratory facilities are reserved and set up. Once the physical environment is established, the team ensures that all necessary equipment is installed and operational. The plan defined the required technical environment and addressed environment planning. Within the environment section of the plan, the team has already identified operational support required to install and check out the operational readiness of the technical environment. The 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 plan should now have been addressed and resolved. Automated tools that apply should have been scheduled for installation and checkout. These tools now should be configured to support the team and be operational within the test environment.

The environment setup activity includes the need to track and manage environment setup activities, where material procurements may have long lead times. The hardware supporting the environment must be sufficient to ensure complete functionality of the production application. Environment hardware needs to support performance analysis. In cases where the environment utilizes hardware resources that are also supporting other development or management activities, special arrangements may be necessary during actual performance testing.

During system execution, the software configuration loaded within the execution environment must be a complete, fully integrated release with no patches and no disabled sections. The hardware configuration supporting the 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 environment design also needs to consider stress testing requirements. Stress and load tests may require the use of multiple workstations that will run multiple procedures simultaneously; some automated test tools include a virtual user simulation functionality that eliminates or minimizes the need for multiple workstations.

Data will need to be obtained with enough lead time to support refinement and manipulation to support requirements. Data preparation activities include the identification of conversion requirements, the preprocessing of raw files, loading of temporary tables, possibly in a relational database management system format, and the performance of consistency checks. During planning, the team defined and scheduled the environment activities.

Now the team needs to track the environment setup activities. Resources need to be identified to install hardware, software, and network resources into the environment and integrate and installed environment resources. The environment materials and the application under system need to be baselined within a configuration management tool. Additionally, execution environment materials may include test data and test processes.

The team needs to obtain and modify databases necessary to exercise software applications, and develop environment setup scripts and test bed scripts. The team should perform product reviews and validation of all source materials. The location of the environment for each project or task should be defined within the plan for each project.

## Phase 5: Execution and Management of Automation

At this stage, the team has addressed test, design and development. Environment setup planning and implementation was addressed consistent with the requirements and guidelines provided within the test plan.

With the test plan in hand and the environment now operational, it's time to execute the automation defined for the test program. When executing procedures, the team must comply with a procedure execution schedule, as discussed earlier. The procedure execution schedule implements the strategy defined within the test plan. Plans for unit, static analysis, code coverage integration, system, and user acceptance testing are executed. Together, these execution phases make up the steps that are required to execute the system as a whole. The various steps involved during execution and management are outlined below.

* When executing automation procedures, the team needs to comply with a procedure execution schedule. Following execution, outcome evaluations are performed and execution 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 executive the system as a whole. With the unit testing, code profiling of software can be performed. “Profiling is a tuning process that determines whether an algorithm is efficient enough”. Profiling also discover instances where there is improper scaling of algorithms, instantiations, and resource utilization.
* Integration testing focuses on the application internals. In this, units are incrementally integrated and tested together based on control flow. Since units may consist of other units, also called *module testing*, may take place during unit testing.
* During system test, the engineer is testing the integration of parts that comprise the entire system. A separate team usually performs system-level executions. The team implements the procedure execution schedule and the system test plan.
* The team also performs analysis to identify specific components or functionality that are experiencing a greater relative number of problem reports. As a result of this analysis, additional test procedures and effort may need to be assigned to the components. Results analysis confirm whether executed execution procedures are proving to be worthwhile in terms of identifying errors.
* Each team needs to perform problem-reporting operations in compliance with a defined process. The documentation and tracking of system problem reports is greatly facilitated by an automated defect-tracking tools and techniques.
* The team manager is responsible for ensuring the execution according to schedule, and Team member are allocated and redirected when necessary to handle problems that arise during the execution effort. To perform this oversight function effectively, the manager needs to perform program status tracking and management reporting.
* Metrics provide the manager with key indicators of the coverage, progress, and the quality of the effort. During white box testing, the engineer measures the *depth* of testing, by collecting data relative to path coverage and test coverage. During black box functional testing, metrics collection focuses on the *breadth* of testing, to include the amount of demonstrated functionality and the amount of execution that has been performed.

## Phase 6: Automation Program Review and Assessment

Automation program review and assessment activities need to be conducted throughout the product lifecycle, to allow for continuous improvement activities. Throughout the lifecycle and following 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 program review and assessment are outlined below.

* Following execution, the team needs to review the performance of the program to determine where changes can be implemented to improve the program performance on the next project. This program review represents the final phase of the Automated Lifecycle Methodology (ALM).
* Throughout the program, the team collected various metrics. The focus of the program review includes an assessment of whether the application satisfies acceptance criteria and is ready to go into production.
* As part of its culture, the team needs to adopt an ongoing iterative process of *lessons learned* activities. Such a program encourages engineers to take the responsibility to raise corrective action proposals immediately, when such actions potentially have significant impact on program performance. Throughout the entire lifecycle, it's good practice to document and begin to evaluate lessons learned at each milestone. The metrics that are collected throughout the lifecycle and especially during the 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 process in a central repository that's easily accessible.
* After collecting lessons learned and other metrics, and defining corrective actions, engineers also need to assess the effectiveness of the program to include an evaluation of the*program return on investment*. Engineers capture measures of the benefits of automation realized throughout the lifecycle in order to support this assessment.
* Teams can perform surveys to inquire about the potential value of process and tool requirements. A survey form is used to solicit feedback on the potential use of requirement-management tools, design tools, and development tools.

## Summary and Conclusions

Adaptive Lifecycle Management is a structured methodology geared toward ensuring successful implementation of automated testing. The ALM approach mirrors the benefits of modern rapid application development efforts, where such efforts engage the user early in the development cycle. The user of the software product is actively involved throughout analysis, design, development, and test of each software build, which is delivered in an incremental fashion may be called as beta versions.

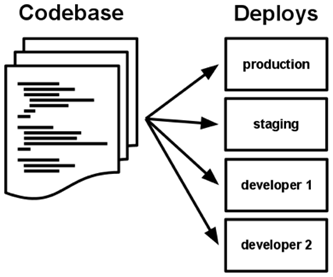
Many organizations have adopted the ALM. Many companies have adopted the book and ALM as their company standard for automated software testing. Others believe that industry automated tool vendors will soon be incorporating the book's structured methodology within their tools. Instead of performing the entire test lifecycle haphazardly, software managers will use an ALM-compliant test tool that automatically supports (and possibly enforces) the book's sound building-block approach to the test effort.

Almost all of the modern cloud or API applications are developed in the new micro-services architecture, there are certain factors or requirements the application should closely adhere to make sure it is cloud ready or can be used to easily expand and change services/behaviors as per market demand. These standards help us achieve applications to be in sync with the cloud computing definition from NIST as closely as possible to be used and deployed on a cloud infrastructure.

The 12 Factor App requirements make sure that the app can be made infrastructure agnostic, be it cloud or standalone. From an Engineering perspective, we have to make sure the below standards are maintained and the arrangements of the infrastructure/resources are as per the twelve factor requirements below:

**I. Codebase**

One codebase tracked in revision control, many deploys. Each app should have only one code base which is version tracked with proper version maintenance of the code. One App cannot have different code bases. There can be different deploys of the same app running. Deploys are running instances of the same app. Each deploy can have different version of code depending upon the code build pipeline status.



**II. Dependencies**

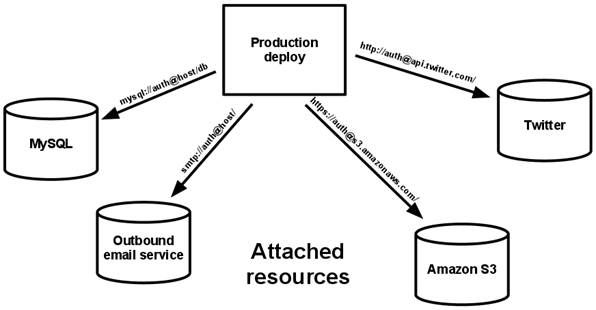
Explicitly declare and isolate dependencies. The dependencies should not be tied implicitly inside app or classes and should be declared in one place for the whole app. Any new developer looking at the app should be able to just pull in dependencies from the dependency management tool and not worry much about pulling in implicit dependencies and should be able to start building and running the app within no time.

**III. Config**

Store config in the environment. Each environment (deploys) should have its own specific variables declared in an environment specific config files. The easy test of it is that the app should be able to be made open source anytime for its config to be kept separated.

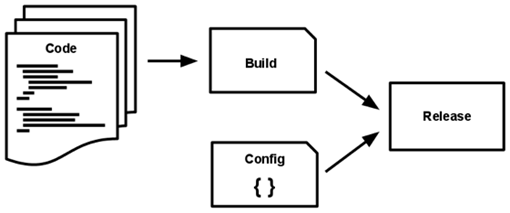
**IV. Backing services**

Treat backing services as attached resources. All the resources whether local DBs or external services should be loosely coupled and can be replaced easily in future if we want to. The backing services should be easily configurable and can be easily replaced in case of failures.



**V. Build, release, run**

Strictly separate build and run stages. The build stage for an app should be separate and be executed before the run(release) stage to make sure the build doesn't have any issue before running the app itself. This helps to isolate the problem and track it easily to fix any issues in lesser time.



**VI. Processes**

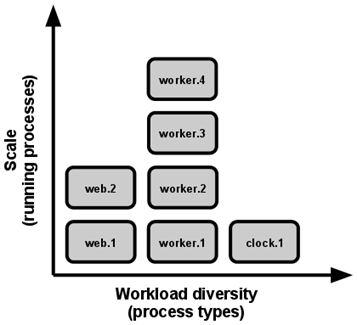
Run the application as stateless processes. Everything should be stored in a data store and any future processes should not depend on the cached data on filesystem or any place. Each process should be independent on its own and should not depend on data of any other process' state.

**VII. Port binding**

Export services via port binding. Expose apps via URL: port for easy access to other external or internal applications. Each app and its services should be easily accessible via URL endpoint whether it is used by an external or internal API or application.

**VIII. Concurrency**

Scale out via the process model. Each app should have its own process types/groups defined like below to scale out the processes as per the requirements in that particular process type. Array of process types and number of processes of particular type is known as the process formation. App should not demonize or write PID files in system, instead it should let the OS handle the processes spun out from the APP which can be distributed over different systems as per the process loads.



**IX. Disposability**

Maximize robustness with fast startup and graceful shutdown. Processes should be able to start up and shutdown in small amount of time and should be able to resume the program execution in minimal amount of time. The system should be robust enough to handle such failures and get back online in no matter of time.

**X. Dev/prod parity**

Development, staging, and production should be as similar as possible. There should be minimal gaps in processes to take app from development to production. The different gaps which need to be minimized are as below:

1. The time gap: A developer may work on code that takes days, weeks, or even months to go into production.
2. The personnel gap: Developers write code, ops engineers deploy it.
3. The tools gap: Developers may be using a stack like Nginx, SQLite, and OS X, while the production deploy uses Apache, MySQL, and Linux.

**XI. Logs**

Treat logs as event streams. A 12 factor system should not worry about storing log files, it should just be writing all its logging events to stdout and let the deployment infrastructure take care of handling logs. The log are events of things happening in the app sorted in a timely manner which need to be routed to appropriate tools to handle logs for analysis and alerting.

**XII. Admin processes**

Run admin/management tasks as one-off processes. All the admin related tasks whether it is in development or production should be executed as on-off processes which once triggered should handle all the admin related task in one go than executing multiple processes for the same.

Depending upon the organization structure and priority, the above factors should be incorporated in an application. Below are some the priorities currently in place for the Cloud Native Architecture requirements:

High = Cloud ready, Medium = Cloud friendly, Low = Cloud Resilient

|  |  |  |
| --- | --- | --- |
| Factor | Priority | Description |
| 1. Codebase | High | One code base tracked in revision control, many deploys |
| 2. Dependencies | High | Explicitly declare and isolate dependencies |
| 3. Configuration | Medium | Store config in the environment |
| 4. Backing Services | Medium | Treat backing services as attached resources |
| 5. Build, Release, Run | Low | Strictly separate build and run stages |
| 6. Processes | High | Execute the app as one or more stateless processes |
| 7. Port Binding | Low | Export services via port binding |
| 8. Concurrency | High | Scale out via the process model |
| 9. Disposability | Medium | Maximize robustness with fast startup and graceful shutdown |
| 10. Dev/Prod Parity | Medium | Keep development, staging, and production as similar as possible |
| 11. Logs | High | Treat logs as event streams |
| 12. Admin Processes | Low | Run admin/management tasks as one-off processes |