**Chapter 5: Adaptive Automation and Testing Design Patterns**

**Keywords: Adaptive, Automation, Development, Testing**

**Introduction:**

**In this Chapter we will discuss about the Adaptive Software Development, Testing and Automation Process.**

Adaptive software development focuses on the rapid creation and evolution of software. This method grew out of the rapid application development method.  
Adaptive Software Development replaces the traditional models with a repeating series of speculate, *collaborate*, and *learn* cycles. 

**Adaptive software Design** (**ASD**) is a [software development process](https://en.wikipedia.org/wiki/Software_development_process) that grew out of [rapid application development](https://en.wikipedia.org/wiki/Rapid_application_development) (RAD). It includes the principle that continuous adaptation of the process to the work at hand is the normal state of affairs.

Adaptive software replaces the traditional models with a repeating series of *speculate*, *collaborate*, and *learn* cycles.

This provides for continuous learning and adaptation to the emergent state of the project. The characteristics of an ASD life cycle are following goal focused, clear picture of features, [iterative](https://en.wikipedia.org/wiki/Iterative_and_incremental_development) in nature, [time Bo](https://en.wikipedia.org/wiki/Timeboxing)und, risk driven, and change tolerant. ASD is also a type of [agile software development](https://en.wikipedia.org/wiki/Agile_software_development) methodology.

ASD is made of following steps (*speculate*, *collaborate*, and *learn*). So let us see those.

The word ***speculate*** refers to the concept that all stakeholders may be comparably wrong for certain aspects of the project’s when discussing the definition of same. During design, the project is initiated and adaptive cycle planning is conducted. Adaptive cycle planning uses project initiation information— mission statement, project constraints, delivery dates, and basic requirements—to define the set of release cycles (software increments)

***Collaboration*** refers to the efforts for managing the work based on predictable parts of the environment (planning and guiding them) and adapting to the uncertain surrounding mix of changes caused by technology, requirements, stakeholders, software vendors.

The ***learning*** cycles, are based on the short iterations with design, build and testing. During these the knowledge is structured by making small mistakes based on false assumptions and correcting those mistakes, leading to more experience and eventually complete understanding problem domain.



An adaptive software development approach is based on observation" and that means both the design of product and the process to create the solution are continuously adjusted with each other based on observation throughout the project an adaptive approach encourages changes throughout the life cycle of project to optimize the design of the overall solution.

An adaptive process is best-suited for projects with uncertainty. It is somewhat ridiculous to try to develop a detailed plan for the effort upfront for uncertain problems. That’s an ideal situation for an adaptive process in this stakeholders need to start with general direction based on whatever is upfront and continue to refine the direction based on observations throughout the duration of the project.

Focuses is on the rapid creation and evolution of software systems. There is never a stoppage where the software is finished; there are just temporary stable periods between releases. Rapid application development gives Time after project completed while adaptive software development is continuous learning curve.  
  
Dynamic cycle in ASD provides for continuous learning and adaptation to the emergent state of the project.

The focus of adaptive software development is in the programming code. Instead of planning the software out well in advance, Programmers have a basic idea in their mind and those kicks off. When parts need changing or adapting to a new system, the programmers simply do it. If the program needs a patch, just makes it.  
  
Overall, removing the pre-planning steps allows the developers/designers to make the software very quickly. Sometime it may result in software that doesn’t perform the precise functions required, but it’s not a problem. The developmental cycle in this process is made short such that a new version with added features can come out very quickly. Rapid prototyping is the related to both adaptive software development and rapid application development.

The other SDLC models are more oriented to the practices of stability, predictability and decreasing returns. The industry, moving to increase environments, those are unpredictable, nonlinear, and fast approaches.

Adaptive Software Development (ASD) has evolved to address these issues. It focuses on the important factor from the management’s perspective, to enhance the ability to manage product development.

In Jim Highsmith’s words, “Adaptive Software Development framework is based on years of experience with traditional Software Development methodologies, consulting on, practicing, and writing about Rapid Application Development (RAD) techniques and working with high-technology software companies on managing their product development practices”.

Waterfall model is found to be characterized by linearity and predictability, with meagre feedback. It can be viewed as a sequence of **Plan → Build → Implement**.



The Evolutionary Lifecycle models such as the Spiral model moved the Deterministic approach to the Adaptive one, with **Plan → Build → Revise Cycles**.



However, the practitioners’ mindset remained Deterministic with long-term predictability turning to short-term predictability. The practices of Evolutionary Lifecycle models such as RAD are found to be less Deterministic.

The Adaptive Life Cycle

The Adaptive model is built from a different point of view. Though cyclical like the Evolutionary model, the names of the phase reflect the unpredictable nature of increasingly complex systems.

Adaptive Development goes further than its evolutionary heritage in two key ways −

* It explicitly replaces Determinism with Emergence.
* It goes beyond a change in life cycle to a deeper change in management style.



The three phases in Adaptive Software Development Lifecycle are −

* **Speculate** − Speculate replaces the deterministic word planning, planning of product specifications or planning of project management tasks.
* **Collaborate** − Collaborate represents drawing a balance between
  + Managing in the traditional project management sense, and
  + Creating and maintaining the collaborative environment needed for emergence.

Collaborative Activities build products, keeping up the pace of changes in the environment.

* **Learn** − Learn aims both, the developers and the customers, to use the results of each development cycle to learn the direction of the next.

In this chapter, we will understand the various concepts of Adaptive Software Development.

## Complex Adaptive Systems (CAS) Theory

Brian Arthur and his colleagues, at the Santa Fe institute, used the Complex Adaptive Systems (CAS) theory to revolutionize the understanding of Physics, Biology, Evolution, and Economics.

Brian Arthur culminated his more than two decades of trying to convince mainstream economists that their view, dominated by fundamental assumptions of decreasing returns, equilibrium, and deterministic dynamics, was no longer sufficient to understand reality. The new world is one of increasing returns, instability, and inability to determine cause and effect.

The two worlds differ in behaviour, style, and culture. They call for −

* Different Management Techniques
* Different Strategies
* Different Understanding

## Complex Software Development

With the scope of Software Applications being exploded, even the software development organizations are accruing similar contradictions as mentioned above.

* One World is represented by the Deterministic development, derived from management practices that are rooted with the basics of stability and predictability (which in Arthur's terms means decreasing returns)
* Second World is represented by the industries moving from decreasing to increasing return environments that are unpredictable, nonlinear and fast.

To address the issues of this second world, Jig Highsmith offered a framework, Adaptive Software Development that is different from the Deterministic Software Development.

The Adaptive Software Development focuses on addressing the complex systems −

* Adaptive Software Development for the development life cycle.
* Adaptive Management Techniques calling for a different mind-set from that of traditional project management practices.

In this tutorial, you can understand both these implementations.

Adaptive Software Development (ASD) is based on two perspectives −

* Conceptual perspective based on the Complex Adaptive Systems (CAS) theory, as given in the first section of this chapter.
* Practical Perspective based on
  + Years of experience with Deterministic software development methodologies.
  + Consulting, practicing, and writing about Rapid Application Development (RAD) techniques; and working with high-technology software companies on managing their product development.

In this chapter, you will understand the conceptual perspective of Adaptive Software Development.

## Complex Adaptive Systems (CAS) Concepts

Complex Adaptive Systems (CAS) theory has many concepts. Adaptive Software Development is based on two of these concepts −

* Emergence
* Complexity

### **Emergence**

In complex software product-development projects, the outcomes are inherently unpredictable. However, successful products emerge from such environments all the time.

This can happen by Emergence, as illustrated in the Complex Adaptive Systems (CAS) theory. It can be understood by a simple example, flocking behavior of birds.

When you observe a flock of birds, you notice that −

* Each bird tries to
  + Maintain a minimum distance from other objects in the environment, including other birds.
  + Match velocities with birds in its neighbourhood.
  + Move towards the perceived center of mass of birds in its neighbourhood.
* There are no rules of behaviour for the group. The only rules are about the behaviour of individual birds.
* However, there exists an emergent behaviour, the flocking of birds. When errant birds rush to catch up, the flock splits around obstacles and reforms on the other side.

This shows the requirement of the most difficult mental model changes in Adaptive Development − from ways of managing and organizing that individual freedom to the notion that a creative new order emerges unpredictably from spontaneous self-organization.

In addition to the development, emergence is the most important concept from the management perspective also.

### **Complexity**

In the Software Development context, Complexity is about −

* The individuals of a team such as the developers, customers, vendors, competitors, and stockholders, their numbers and their speed.
* Size and technological complexity.

## Adaptive Software Development Practices

Adaptive Software Development offers a different perspective on software management practices. In the sections below, you can understand the two important practices − Quality and RAD, both of which have ramifications for gathering requirements.

You can find the details of all the practices in the chapter, Adaptive Software Development Practices in this tutorial.

### **Quality**

In a complex environment, the age-old practice of "Do it right the first time" does not work as you cannot predict what is right at the beginning. You need to have an aim to produce the right value. However, in complex environment, the combinations and permutations of value components like scope (features, performance, defect levels), schedule, and resources is so vast that there can never be an optimum value. Hence, the focus is to shift to deliver the best value in the competitive market.

### **RAD Practices**

RAD Practices generally involve a combination of the following −

* Evolutionary Lifecycle
* Customer Focus Groups, JAD Sessions, Technical Reviews
* Time-boxed Project Management
* Continuous Software Engineering
* Dedicated Teams with war rooms

The RAD projects have an inherent adaptive, emergent flavour. Many IT organizations are against RAD. However, Microsoft and others have produced incredibly large and complex software using techniques comparable to RAD because it raises questions about their fundamental world view.

RAD practices and Microsoft process are both examples of Adaptive Development in action. Giving them a label (i.e., Adaptive Development) and realizing that there is a growing body of scientific knowledge (i.e., CAS theory) explains why they work. This should provide a basis for more extensive use of these practices.

Adaptive Software Development Lifecycle has six basic characteristics −

* Mission focused
* Feature based
* Iterative
* Time-boxed
* Risk driven
* Change tolerant

In this chapter, you will understand these six characteristics of Adaptive Software Development.

Mission-focused

For many projects, the overall mission that guides the team is well articulated, though the requirements may be uncertain at the beginning of the project. Mission statements act as guides that encourage exploration in the beginning but have a narrow focus over the course of a project. A mission provides boundaries rather than a fixed destination. Mission statements and the discussions that result in those statements provide direction and criteria for making critical project trade-off decisions.

Without a clear mission and a constant mission refinement practice, iterative lifecycles become oscillating lifecycles, swinging back and forth with no progress in the development.

Feature-based

The Adaptive Software Development Lifecycle is based on application features and not on tasks. Features are the functionality that are developed during an iteration based on the customer’s priorities.

Features can evolve over several iterations when the customers provide feedback.

The application features that provide direct results to the customer after implementation are primary. A customer-oriented document such as a user manual is also considered as a feature. The other documents such as the data model, even if defined as deliverables are always secondary.

Iterative

The Adaptive Software Development Lifecycle is iterative and focuses on frequent releases in order to obtain feedback, assimilate the resulting learning and setting the right direction for further development.

Time-boxed

In Adaptive Software Development Lifecycle, the iterations are time-boxed. However, one should remember that time-boxing in Adaptive Software Development is not about time deadlines. It should not be used to make the team work for long hours challenging a collaborative environment or for compromising on the quality of the deliverables.

In Adaptive Software Development, time-boxing is considered as a direction for focusing and forcing hard trade-off decisions as and when required. In an uncertain environment, in which change rates are high, there needs to be a periodic forcing function such as a time box to get the work finished.

Risk-driven

In Adaptive Software Development, the iterations are driven by identifying and evaluating the critical risks.

Change-tolerant

Adaptive Software Development is change-tolerant, viewing change as the ability to incorporate competitive advantage, but not as a problem for development.

# **Adaptive Software Development - Practices**

The Adaptive Software Development practices are driven by a belief in continuous adaptation, with the lifecycle equipped to accepting continuous change as the norm.

Adaptive Software Development Lifecycle is dedicated to −

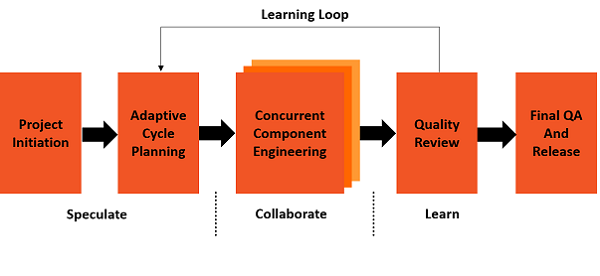
* Continuous learning
* Change orientation
* Re-evaluation
* Peering into an uncertain future
* Intense collaboration among developers, management, and customers

## Adaptive SDLC

Adaptive Software Development combines RAD with Software Engineering Best Practices, such as −

* Project initiation.
* Adaptive cycle planning.
* Concurrent component engineering.
* Quality review.
* Final QA and release.

Adaptive Software Development practices can be illustrated as follows −



As illustrated above, Adaptive Software Development practices are spread across the three phases as follows −

* **Speculate** − Initiation and planning
  + Project Initiation
  + Establishing time-box for the entire project
  + Decide on the number of iterations and assign a time-box to each one
  + Develop a theme or objective for each of the iterations
  + Assign features to each iteration
* **Collaborate** − Concurrent feature development
  + Collaboration for distributed teams
  + Collaboration for smaller projects
  + Collaboration for larger projects
* **Learn** − Quality Review
  + Result quality from the customer's perspective
  + Result quality from a technical perspective
  + The functioning of the delivery team and the practices team members are utilizing
  + The project status

## Speculate - Initiation and Planning

In Adaptive Software Development, the speculate phase has two activities −

* Initiation
* Planning

Speculate has five practices that can be executed repetitively during the initiation and planning phase. They are −

* Project initiation
* Establishing time-box for the entire project
* Decide on the number of iterations and assign a time-box to each one
* Develop a theme or objective for each of the iterations
* Assign features to each iteration

### **Project Initiation**

Project Initiation involves −

* Setting the project's mission and objectives
* Understanding constraints
* Establishing the project organization
* Identifying and outlining requirements
* Making initial size and scope estimates
* Identifying key project risks

The project initiation data should be gathered in a preliminary JAD session, considering speed as the major aspect. Initiation can be completed in a concentrated two to five day effort for a small to medium sized projects, or two to three weeks effort for larger projects.

During the JAD sessions, requirements are gathered in enough detail to identify features and establish an overview of the object, data, or other architectural model.

### **Establishing Time-box for the Entire Project**

The time-box for the entire project should be established, based on the scope, feature-set requirements, estimates, and resource availability that result from project initiation work.

As you know, Speculating does not abandon estimating, but it just means accepting that estimates can go wrong.

### **Iterations and Time-box**

Decide on the number of iterations and the individual iteration lengths based on the overall project scope and the degree of uncertainty.

For a small to medium sized application −

* Iterations usually vary from four to eight weeks.
* Some projects work best with two-week iterations.
* Some projects might require more than eight weeks.

Choose the time, based on what works for you. Once you decide on the number of iterations and the lengths of each of the iterations, assign a schedule to each of the iterations.

### **Develop a Theme or Objective**

The team members should develop a theme or objective for each iteration. This is something similar to the Sprint Goal in Scrum. Each iteration should deliver a set of features that can demonstrate the product functionality making the product visible to the customer to enable review and feedback.

Within the iterations, the builds should deliver working features on a preferably daily basis enabling integration process and making the product visible to the development team. Testing should be an ongoing, integral part of the feature development. It should not be delayed until the end of the project.

### **Assign Features**

Developers and customers should together assign features to each iteration. The most important criteria for this feature assignment is that every iteration must deliver a visible set of features with considerable functionality to the customer.

During the assignment of features to the iterations −

* Development team should come up with the feature estimates, risks, and dependencies and provide them to the customer.
* Customers should decide on feature prioritization, using the information provided by the development team.

Thus iteration planning is feature-based and done as a team with developers and customers. Experience has shown that this type of planning provides better understanding of the project than a task-based planning by the project manager. Further, feature-based planning reflects the uniqueness of each project.

## Collaborate ─ Concurrent Feature Development

During the Collaborate phase, the focus is on the development. The Collaborate phase has two activities −

* The Development team collaborate and deliver working software.
* The project managers facilitate collaboration and concurrent development activities.

Collaboration is an act of shared creation that encompasses the development team, the customers and the managers. Shared creation is fostered by trust and respect.

Teams should collaborate on −

* Technical problems
* Business requirements
* Rapid decision making

Following are the practices relevant to the Collaborate phase in Adaptive Software Development −

* Collaboration for distributed teams
* Collaboration for smaller projects
* Collaboration for larger projects

### **Collaboration for Distributed Teams**

In the projects involving distributed teams, the following should be considered −

* Varying alliance partners
* Broad-based knowledge
* The way people interact
* The way they manage interdependencies

### **Collaboration for Smaller Projects**

In the smaller projects, when team members are working in physical proximity, Collaboration with informal hallway chats and whiteboard scribbling should be encouraged, as this is found to be effective.

### **Collaboration for Larger Projects**

Larger projects require additional practices, collaboration tools, and project manager interaction and should be arranged on the contextual basis.

## Learn - Quality Review

Adaptive Software Development encourages the concept of ‘Experiment and Learn’.

Learning from the mistakes and experimentation requires that the team members share partially completed code and artifacts early, in order to −

* Find mistakes
* Learn from them
* Reduce rework by finding small problems before they become large ones

At the end of each development iteration, there are four general categories of things to learn −

* Result quality from the customer's perspective
* Result quality from a technical perspective
* The functioning of the delivery team and the practices team
* The project status

### **Result Quality from the Customer's Perspective**

In the Adaptive Software Development projects, getting feedback from the customers is the first priority. The recommended practice for this is a customer focus group. These sessions are designed to explore a working model of the application and record customer change requests.

Customer focus group sessions are facilitated sessions, but rather than generating requirements or defining project plans, they are designed to review the application itself. The customers provide feedback on the working software resulting from an iteration.

### **Result Quality from a Technical Perspective**

In the Adaptive Software Development projects, periodic review of technical artifacts should be given importance. Code Reviews should be done on a continuous basis. Reviews of other technical artifacts, such as technical architecture can be conducted weekly or at the end of an iteration.

In Adaptive Software Development projects, the team should monitor its own performance periodically. Retrospectives encourage the teams to learn about themselves and their work, together as a team.

Iteration-end retrospectives facilitate periodic team performance self-review such as −

* Determine what is not working.
* What the Team needs to do more.
* What the Team needs to do less.

### **The Project Status**

The Project status review helps in planning further work. In the adaptive software development projects, determining the project status is feature-based approach, the end of each iteration marked by completed features resulting in working software.

The Project Status review should include −

* Where is the project?
* Where is the project versus the plans?
* Where should the project be?

As the plans in the Adaptive Software Development projects are speculative, more than the question 2 above, question 3 is important. That is, the project team and the customers need to continuously ask themselves, "What have we learned so far, and does it change our perspective on where we need to go?"

# **Adaptive S/W Development - Management**

Traditional software management has been characterized by the term command-control.

Many organizations are steeped in a tradition of optimization, efficiency, predictability, control, rigor and process improvement. However, the emerging information age economy requires adaptability, speed, collaboration, improvisation, flexibility, innovation, and suppleness.

Harvard business review and management books have come up with the terms such as empowerment, participative management, learning organization, human-cantered management, etc., but none of these are being put into managing modern organizations.

In the context of Adaptive Software Development, the gap looks much wider and there is a necessity to consider the Adaptive management techniques that have been proven successful in other fields.

Adaptive Management

Adaptive management has proven successful in the environments where the resource managers worked together with stakeholders and scientists as a team, with the following goals −

* To learn how managed systems respond to human interventions.
* To improve resource policies and practices in future.

The principle behind adaptive management is that many resource management activities are experiments as their outcomes cannot be reliably predicted beforehand. These experiments are then used as learning opportunities for the improvements in the future.

Adaptive management is intended to increase the ability to respond timely in the face of new information and in a setting of varied stakeholder objectives and preferences. It encourages stakeholders to bound disputes and discuss them in an orderly fashion while the environmental uncertainties are being investigated and better understood.

Adaptive management helps the stakeholders, the managers and other decision makers recognize the limits of knowledge and the need to act on imperfect information.

Adaptive management helps to change the decisions made by making it clear that −

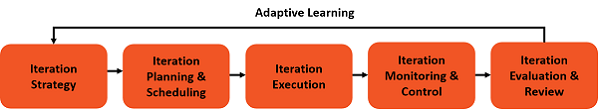
* The decisions are provisional.
* A management’s decision need not always be right.
* Modifications are expected.

There are two types of Adaptive management approaches −

* Passive Adaptive Management.
* Active Adaptive Management.

Passive Adaptive Management

Adaptive management aims to enhance the scientific knowledge and thereby reduce uncertainties.

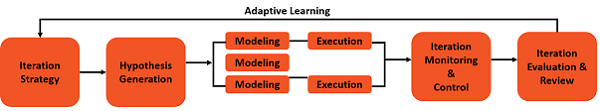


Within Passive Adaptive management, a single preferred course of action, based on existing information and understanding, is selected. The outcomes of management actions are monitored, and subsequent decisions are adjusted based on the outcomes.

This approach contributes to the learning and effective management. However, it is limited in its ability to enhance scientific and management capabilities for conditions that go beyond the course of action selected.

Active Adaptive Management

An Active Adaptive management approach reviews the information before management actions are taken.



A range of competing, alternative system models of ecosystem and related responses (e.g. demographic changes; recreational uses), rather than a single model, is then developed. Management options are chosen based on the evaluations of these alternative models.

Leadership-Collaboration Management

Adaptive management is what is best suited for Adaptive Software Development. The approach requires resource managers, i.e. the managers who can work with people, allow human-interventions, and create an amicable environment.

In software development, the leaders often take up these responsibilities. We need leaders more than the commanders. The leaders are collaborators and work alongside with the team. Collaborative-Leadership is the most sought after practice in Adaptive development.

The leaders have the following qualities −

* Grasp and set the direction.
* Influence people involved and provide guidance.
* Collaborate, facilitate and macro-manage the team.
* Provide direction.
* Create environments where talented people can be innovative, creative, and make effective decisions.
* Understand that occasionally they need to command, but that is not their predominant style.

Adaptive Testing:

Adaptive testing is the counterpart of adaptive control in software testing. It means that software testing strategy should be adjusted on-line by using the testing data collected during software testing as our understanding of the software under test is improved. Previous studies on adaptive testing involved a simplified Controlled Markov Chain (CMC) model for software testing which employs several unrealistic assumptions. In this paper we propose a new adaptive software testing approach in the context of an improved and namely, general CMC model which aims to eliminate such threats to validity. A set of more realistic basic assumptions on the software testing process is proposed and several unrealistic assumptions are replaced by less unrealistic assumptions. A new adaptive testing strategy based on the general CMC is developed and implemented. Mathematical simulations and experiments on real life software are conducted to demonstrate the effectiveness of the new strategy.

Abstract Adaptive software testing is the counterpart of adaptive control in software testing. It means that software testing strategy should be adjusted online by using the testing data collected during software testing as our understanding of the software under test improves. Previous studies on adaptive testing rely on a simplified Controlled Markov Chain (CMC) model for software testing which employs several unrealistic assumptions. In this paper we propose a new adaptive software testing approach in the context of an improved CMC model which aims to eliminate such threats to validity. A new set of basic assumptions on the software testing process is proposed and several unrealistic assumptions are replaced by more common situations in real life software testing. The methodology of a new adaptive testing strategy is also developed and implemented. Experimental data are collected to demonstrate the effectiveness of the new methodology.

The problem studied in this paper is also closely related to the test case prioritization problem [9]. Elbaum et.al reported that in regression testing feedback may play positive or negative role in test case prioritization [5]. Do et.al presented an empirical study on assessing the ability of prioritization techniques to improve the rate of fault detection of test case prioritization techniques, measured relative to mutation faults in [4]. The major difference between this work and other test case prioritization techniques is the prioritization is carried out on-the-fly as testing proceeds, which means testing history information is collected and used for future decision making. Other related studies include defect removal and its impact on software testing. Okamura proposed a new reliability estimation method that considers defect removal [8]. This study presents a rigorous model for the defect removal process and its impact to the software under test, and developed the according methodology for testing and parameter estimation.

### **Speculate**

A high degree of certainty is indicated about the desired results by the term plan. The manager’s ability of steering the project in innovative directions is restricted by the implicit and explicit goals of the plan conformance.

The term is replaced by the term speculate in Adaptive Software Development. The reality of the uncertainty in complex problems is acknowledged without the planning being abandoned. Exploration and experimentation is encouraged by Speculate. Iterations with short cycles are encouraged.

### **Collaborate**

Complex applications which are evolved, require a large volume of information for collecting, analyzing and applied to the problem. Turbulent environments have high rates of information flow and hence complex applications require huge volume of information for collecting, analyzing of the information. This results in diverse Knowledge requirements that can only be handled by team collaboration.

In order to produce results, share knowledge and make decisions, it is required to work jointly which is done by Collaborate.

A balance between managing with the traditional management techniques and creation and maintenance of the collaborative environment is portrayed by Collaboration.

### **Learn**

For the success of the project, Learn is an important part in the Adaptive Development Software lifecycle. By using some of the practices, the team has to enhance the knowledge by:

* Technical Reviews
* Project Retrospectives
* Customer Focus Groups

After each of the iteration, reviews are done. The assumptions are examined and results of each of the development cycle for learning the next direction by the developers and customers. The team learns about the

* Changes in the product
* Changes regarding the assumptions of how products are developed.
* About product changes

Adaptive Automation Testing:

DevOps as a trend is on the rise. Studies show that number of organizations adopting devops continues to grow with every passing year. One of the core requirements in the devops world is higher automated software testing. Even manual testers would soon have to take on automated testing in possible areas of their operations. To this effect, automation teams have been thinking out of the box to create automation frameworks that do much of the heavy lifting upfront enabling easier and more effective automated testing to be taken up by one and all on the test team. However one ongoing flaw in all of this, is the disconnect between the test automation and the varied test effort management systems including the test case and defect management systems.

This has been an area of research at QA InfoTech in the recent months to help draw a complete connect in the automation effort E2E and enable automation in the truest essence. This has finally been taking shape where we expect to see measurable outcomes of around 100% automated effort (meaning a test case when automated is fully automated – there is no manual effort involved), a tight coupling between defect management and automation execution, and considerable time savings on the tester’s plate – this is what we call the adaptive [automation testing framework](https://qainfotech.com/automation-testing-services-and-tools.html).

We have our annual technical symposium, Qualloquium coming up on Sept 19th. This is one of the topics the automation team would be presenting on, explaining how the framework operates and what benefits it can bring to the table. We are very excited for our audience, to encourage more of our manual testers also take on automation and importantly also enable our automation engineers to take on more productive automation, as more teams move into the DevOps world.

**Test Design Patterns with Respect to Adaptive Automation**:

When we find a specific set of problem and a specific set of corresponding solution then the combined set is defined as a pattern.

With respect to Test Pattern it specifies the way in which any product/service or application is been tested for a specific set of problem with respect to get know set of solution.

Design patterns are created to solve common problems in software design. They are not reserved only for software development but useful for software automation. Yes, there are really sophisticated design patterns used to solve complex issues in software automation. And there are ways easy to understand and adopt design patterns that can significantly improve readability and maintainability of our test automation code.

We will see the design patterns used in Adaptive and Dynamic Test Automation framework.

The same design patterns may be useful in other software activities but we will see how they can be used with Adaptive Test Automation.

->Data Patterns:

This patterns separate the Data Management from Test Logic, hence logic is clearer and there are no mixes with data. Data is managed separately whether in memory or in Data Base.

For example data may be kept in file and may be accessed with Data Provider Module as following.

@DataProvider

def adaptiveTestDataProvider():

With open (“C:/adaptiveTestData.xls”, r+) as f:

For line in f.readlies():

Print line

return

->Technical Patterns:

In this pattern, product technology or environment complexities are been kept separately from the Test Steps being executed. It reduces test complexity and improves test maintainability.

->Proxy Patterns:

Web Server

HTTP

Internet HTTP

Client Machine

HTTP

Internet

Server

HTTP Proxy

Cache Storage

[Executing Automation against External Vendors via Proxy](https://fusion.mastercard.int/confluence/pages/viewpage.action?pageId=228101121) pattern

Execution of automation against an external vendor such as Browser Stack, Perfecto or AppliTools Eyes is now possible via a proxy.

This is critical as most of organizations environments control outbound connectivity via a proxy server and running tests against external vendors has been impossible from these environments until now. This post will demo the new Test Framework capabilities and show how to make use of them.

We have the proxy settings be controlled by environment by default. ATAF recognizes the LOCAL, DEVCloud environment, and Productions environments by default and has dedicated property files for each. We recognize there are many different environment definitions within organization and you are always free to define your own property files. For this we can take example of **Default Proxy of cloud environment.**

Whether or not a proxy is used is controlled by the presence or absence of two properties (${VENDOR}.proxy.host & ${VENDOR}.proxy.port). The vendors currently supported are browser stack, perfecto & applitools.

At the moment in STAGE the proxy host & port are the same for all of these vendors but we defined them individually in case that changes in the future. By default ATAF defines proxy host/port in the STAGE property file of mtaf-driver-factory as host=[outboundproxy.mclocal.int](http://outboundproxy.mclocal.int/) & port=15768.

To run a test using these proxy settings a user simply needs to pass in the param -Denv=STAGE and everything else as normal.

The Driver Factory reads the STAGE property file, picks up these default settings, validates their form and adds a proxy.

**Custom Proxy Settings**  
if you need to define custom proxy settings in STAGE or any other environment the process is fairly straightforward and simple. There are two methods to do so:

1) Pass the host & port as -Dparameters, for example:

-Ddefault.web.execution.platform=browserstack\_chrome -Dbrowserstack.user=xxx -Dbrowserstack.key=xxx -Dbrowserstack.web.os=Windows -Dbrowserstack.web.os.version=10 -Dbrowserstack.chrome.version=59 -Dbrowserstack.proxy.host=some.host.url -Dbrowserstack.proxy.port=123456

2) Add a property file under the appropriate directory in your testing project (src/main/resources/config/${ENV} with the appropriate properties then pass -Denv=${ENV}

DEV.properties

browserstack.proxy.host=some.host.url

browserstack.proxy.port=123456

When the test is run the DriverFactory will pick up these properties, validate that the host is well formed and the port is between 1-65535 and create a proxy for the driver if the validation passes. If either the host or the port are invalid the DriverFactory will simply ignore them and create a RemoteWebDriver without a proxy.

Uses of HTTP Proxy:

1. It blacklists external resources
2. Cache Non Functional Resources
3. Collects HTTP Traffic for analysis (Redirects, Loading Time etc.)
4. Speedup Page Loading

Business Patterns:

This gives possibility to get actual business requirements and design Adaptive Test Automation accordingly. This makes Dynamic approach more valuable

Page Objects Pattern: in adaptive test automation it allows to create object repository with User Interface elements and these repository is separated from actual Test Automation logic gives freedom of plugging as and when required.

Façade pattern:

This pattern is useful when we need to make simple interfaces with more complex system/code need to be tested. In adaptive automation strategy as per user and system experience easy to use and maintainable API’s need to be designed either externally or internally, hence will give more control dynamically at run time.

Factory Pattern:

In adaptive automation scenarios, at run time decision can be taken by the factory for the kind of objects needs to be created for factory as it might not be suitable or we might not know or we are not bothered about the same. Hence specific rules are defined for every factory for creating specific objects.

Singleton Pattern: This pattern is used when in our adaptive Automation Testing we need to deal with exactly one object.

Null Object Pattern:

Test Design Patterns for Run time Dynamic Cross platform Testing:

Test Design Pattern based on Agile Principle: