Chapter 5

ADAPTIVE AUTOMATION TESTING DESIGN PATTERNS

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

# 5.1 Introduction

In this Chapter we will discuss about the Adaptive Software Development, Testing and Automation Design Patterns 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 that grew out of 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 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 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.



**Figure 5.1**: Learn Speculate Collaborate

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. 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.

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) evolved to address above mentioned issues. It focuses on the important factor from the management’s perspective, to enhance the ability to manage product development.

In Jim Highsmith’s definition, “Adaptive Software Development framework is being matured 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 characterized by linearity and predictability. It is viewed as a sequence of **Plan → Build → Implement**.



**Figure 5.2**: Plan Build Implement

The Evolutionary Lifecycle models like Spiral model moved the Deterministic approach to the Adaptive approach, with **Plan → build → Revise Cycles**.



**Figure 5.3**: Plan Build Revise

However, the mind set remained Deterministic with long-term predictability turns into short-term predictability. Hence the practices of Evolutionary Lifecycle models are found to be less Deterministic.

# 5.2 The Adaptive Life Cycle

The Adaptive model is built cyclical like the Evolutionary model, and the names of the phase reflect the unpredictable nature of increasingly complex systems.

Adaptive Development goes further as following −

* It explicitly replaces Determinism with Emergence.
* It traverse from change in life cycle then in management style.



**Figure 5.4**: Speculate Learn Collaborate

The phases in ASD Lifecycle are as following −

* **Speculate** − the deterministic word planning is replaced by speculate, planning of product specifications or project management etc...
* **Collaborate** − Collaborate represents balance between
  + Managing the traditional project management style and
  + Starting and maintaining the collaborative environment for emergence.
  + Collaborative Activities, build products, keeping up the pace of changes in the development environment.
* **Learn** − Learn aims stakeholders, to use the results of each development cycle to learn the direction of the next iteration cycle.

**Concepts of Adaptive Software Development as following**.

# 5.3 Complex Adaptive Systems (CAS) Theory

Brian Arthur with his team, at the Santa Fe institute, used the CAS theory to revolutionize the understanding of Sciences, Evolution, and Economic Systems.

He culminated that the new world is one of increasing returns, instability, and inability to determine cause and effect.

So there is differences of behavior, style, and culture with Management Techniques, Strategies and Understanding.

# 5.4 Complex Software Development

So as per above observations even the software development organizations are accruing similar challenges.

1. Second World is represented by the Deterministic development, derived from management practices that are rooted with the basics of stability and predictability (decreasing returns)
2. Second World is represented by the industries moving from decreasing to increasing return environments those are unpredictable, nonlinear and fast.

To address the issues of this second world, a framework is being designed called as Adaptive Software Development which is different from the Deterministic Software Development.

The Adaptive Software Development focuses on addressing the complex problems of engineering environment and Management of organizations.

* ASD for the development life cycle.
* Adaptive Management Techniques with different mind-set from then traditional project management practices and techniques.

There are two perspective of Adaptive Software Development (ASD): −

1. **Conceptual perspective** determined on the Complex Adaptive Systems (CAS) theory.
2. **Practical Perspective** requires following
   * Long Work experience with Deterministic software development methodologies.
   * Consulting, and practicing, about Rapid Application Development (RAD) working with Hi-tech software organizations on managing product development.

Now let’s focus on conceptual perspective of Adaptive Software Development.

# 5.5 Complex Adaptive Systems (CAS) Concepts

On the basis of CAS concepts Adaptive Software Development is based on following concepts:

* **Emergence**
* **Complexity**
* **Quality**

**5.5.1 Emergence**

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

This can happen by Emergence, as illustrated in CAS theory. It can be understood by a simple example, flocking behavior of flying birds.

When it’s been observed, that

* Each bird tries to
  + Maintain a minimum distance from other objects in the environment, including other birds, Match velocities with birds in its neighborhood bird, Move towards the perceived center of mass of birds in its neighborhood.
* Hence it’s been observed that there is no rules of behavior of whole group but observation is about the behavior of individual birds. However, there exists an emergent behavior, the flocking of birds. When diverted birds rush to manage path, the flock splits around obstacles and reforms on the other side.
* Hence it can be understood that the most difficult mental model changes in Adaptive Development from ways of managing and organizing that individual freedom to creative new order emerges unpredictably from spontaneous self-organization.

Development, and emergence are the most important concept from the management perspective.

**5.5.2 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.

**5.5.3 Quality**

In a complex environment, the practice of "Do things right from start" does not work as right things cannot be predicted at the beginning. Aim should be to produce the right value as a final result. However, in complex software development environment, the combinations and permutations of value components like scope (features, performance, defect levels), schedule, and resources is vast that there can never be an optimum value. Hence, the focus should be to shift to deliver the best value in the competitive environment.

# 5.6 RAD Practices

RAD Practices generally contains following −

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

The RAD projects have an inherent adaptive, emergent flavor. Microsoft process follows the RAD practices are both examples of Adaptive Development in action. Giving a label and realizing about a growing body of scientific knowledge (i.e., CAS theory) explains why they work. This should provide a basis for extensive use of ASD practices.

Following are has six basic characteristics of Adaptive Software Development Lifecycle

## 5.6.1 Mission-focused

For many projects, requirements may be uncertain at the beginning but the goal that guides the team is well articulated manner. Mission statements act as guide that encourage exploration in the beginning but have a linear focus over the course of a project. A mission provides boundaries in place of a fixed destination. Mission statements provide direction with criteria for making critical project trade-off decisions.

Without a focused and constant mission refinement practice, iterative lifecycles become oscillating lifecycles, with no progress in the development process.

## 5.6.2 Feature-based

The Adaptive Software Development Lifecycle should be based on application features and not on process. Features are developed side by side with an iteration based on the stakeholder’s priorities.

Feature evolves over iterations when the customers provide feedback. Primary Feature may consists of that provide direct results to the customer after implementation also a customer-oriented document (user manual). Secondary features are documents i.e. the data model, may be defined as deliverables.

## 5.6.3 Iterative

The Adaptive Software Development Lifecycle is iterative which focuses on frequent releases to get feedback, learning and setting the proper direction for continuous development.

## 5.6.4 Time-boxed

In Adaptive Software Development Lifecycle, the iterations are time-bound. But time-boxing in Adaptive Software Development is not time deadlines. Team does not work for long hours and works in collaborative environment hence no compromise on the quality of the deliverables.

Time-boxing is considered as a direction for focus and forcing hard trade-off decisions whenever required. In an uncertain environment, where change rates are high, there should be a periodic forcing function to get the work finished.

## 5.6.5 Risk-driven

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

## 5.6.6 Change-tolerant

Adaptive Software Development is tolerant to change, and look at this as the ability to incorporate better advantage, but should not be a problem for development.

# 5.7 Adaptive Software Development - Practices

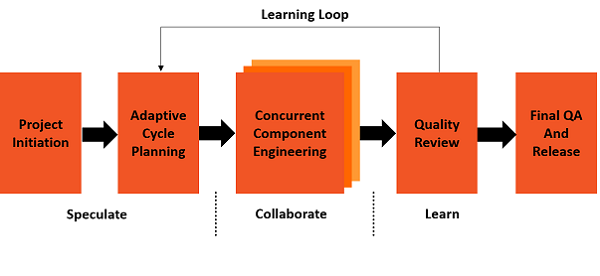
The Adaptive Software Development driven by continuous adaptation, with the lifecycle equipped to accepting continuous change.

Hence the Adaptive Software Development Lifecycle explores Continuous learning, Change Orientation, Re-evaluation, and Peering into a changing future and does intense collaboration among stakeholders.

## 5.7.1 Adaptive SDLC

Adaptive Software Development combines RAD with Best Practices, such as Project initiation, Planning, Concurrent Engineering, Quality review and Final QA and release.

Adaptive Software Development practices is illustrated as follows −



**Figure 5.5**: Adaptive Learning Loop

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

* + **Speculate** includes Initiation and planning, it does Project Initiation then Establishes time-box for the entire project, thereafter decide on the number of iterations and assign a fixed time to each one to develop a theme or objective for each of the iterations and then assign features to each iteration
  + **Collaborate** – Starts Concurrent feature development, Collaboration for distributed teams, for smaller projects, and for larger projects
  + **Learn** − Quality Review results quality from the stakeholders perspective and it result quality from a technical perspective, The functioning of the delivery team and the practices team members are utilizing The project status

Now let us explore the three points as following.

## 5.7.2 Speculate - Initiation and Planning

Speculate phase has two activities

1. Initiation
2. Planning

There are five practices need to be processed repetitively during the initiation and planning phase.

* Project initiation
* Establishing time-bound planning.
* Define the number of iterations and assign fix time for each iteration.
* Develop objective for each of the iterations
* Assign features to every ach iteration

### 5.7.2.1 Project Initiation:

Project Initiation involves, setting the project's mission and objectives then Understanding constraints, does establishing the project organization, Identifying and outlining requirements and Making initial size and scope estimates and finally Identifying key project risks

The project initiation data should be gathered in a preliminary sessions, considering speed as the major aspect. Initiation can be completed in around a week and effort for a small to medium sized projects, or two to three weeks effort for larger projects. Requirements are gathered in detail to identify features and establish an overview of the object, data, or other architectural model.

### 5.7.2.2 Establishing Time-box for the Entire Project:

The time-box for the whole project should be established, based on the scope, requirements, estimates, and resource availability that result from project initiation work. Speculating does not overrule estimating, but it means accepting that estimates may go wrong.

### 5.7.2.3 Iterations and Time-box:

Project stakeholders should 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 vary from four to eight weeks.
* Some projects work best with around two-week iterations.
* Some projects might require around more than eight weeks.

Choose the time, based on what works. Once decision has been taken on the number of iterations and the lengths of each of the iterations, assign a schedule to each of the iterations.

### 5.7.2.4 Develop a Theme or Objective:

Stakeholders should develop a theme/objective for every iteration and same should be similar to the Sprint Goal in Scrum. Each iteration should deliver a feature set that demonstrates the product functionality and making the product visible to the customer to enable review and feedback.

Inside 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, and should be integral part of the feature development. Theme should not be delayed until the end of the project.

### 5.7.2.5 Assign Features:

Stakeholders should assign features to each iteration. The important criteria for this feature assignment is that every iteration should 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 pass them to the customer. And Customers should decide on feature prioritization, with the information provided by the development team.

Iteration planning is feature-based and done as a team with all Stakeholders. Experience has shown that Iteration planning provides better understanding of the project than a task-based planning manager. Thereafter, feature-based planning reflects the uniqueness of each project.

## 5.7.3 COLLABORATE, FEATURE, DEVELOPMENT

During the Collaborate phase, focus should always be on the development. The Collaborate phase has two activities −

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

**Collaboration** is an act of shared creation that keeps together the development team, the customers and the managers. This is fostered by trust and respect.

Teams should collaborate on Technical problems, Business requirements and Rapid decision making

Following are the rules for the Collaborate phase in ASD.

### Collaboration for Distributed Teams

In the projects involving distributed teams, the following should be considered varying alliance partners with Broad-based knowledge, the way people interact, the way they manage interdependencies

### Collaboration for Smaller Projects

In smaller projects, when team members works in physical proximity, Collaboration with informal and whiteboard scribbling should be encouraged, as this is found to be effective, All Stakeholders should be collaborated in small projects.

### Collaboration for Larger Projects

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

## 5.7.4 Learn - Quality Review

ASD encourages the concept of ‘Experiment and Learn’. Learning from the mistakes/experimentation requires that the team share partially completed artifacts early, in order to −

* Find mistakes
* Learn from them
* Find small problems before they become large.

At the end of each iteration, below 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 team
* The project status

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

Getting feedback from the customers is the first priority in Adaptive Software Development projects. These 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 process resulting from an iteration.

### 5.7.4.2 Result Quality from a Technical Perspective

In ASD projects, cyclic review of technical artifacts should be given much importance. Code Reviews should be done on regularly. Reviews of other technical artifacts, can be conducted weekly or at the end of an iteration as per the requirements.

In Adaptive Software Development projects, the team should keep an eye on its own performance periodically. It encourage the teams to learn about approach and their work, together as a group.

Iteration-end retrospectives checks periodic team performance self-review such as and it determine what is not working then what the Team needs to do more or less.

### 5.7.4.3 The Project Status

The Project review helps to plan future course of action. In the ASD 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 about of the project?
* Details of project and the plans?
* And where should the project be at this time?

Hence the project team and the customers need to continuously ask themselves, "What have been learned so far, and does it change perspective of goal?"

# 5.8 Adaptive S/W Development - Management

Traditional software management is characterized by the term command-control.

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

Business review and management books has terms such as empowerment, participative management, learning organization, human-cantered management, etc., but these are not put into managing modern organizations.

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

## 5.8.1 Adaptive Management

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

1. Learn how to manage systems respond to human interventions.
2. Improve resource policies and practices for future planning.

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

Adaptive management is put on board to increase the ability to respond timely in the face of added information and in a setting of varied stakeholder objectives/ preferences. It encourages stakeholders to show disputes and discuss in an orderly fashion while the environmental uncertainties are being investigated and better understood.

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

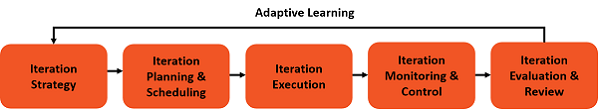
Adaptive management helps to change the decisions made by making it clear that the decisions are provisional and a management’s decision need not always be right hence Modifications are expected.

There are two types of Adaptive management Policies −

1. Passive Adaptive Management.
2. Active Adaptive Management.

## 5.8.2 Passive Adaptive Management

The aim of Adaptive management is to enhance the scientific knowledge and reducing uncertainties.



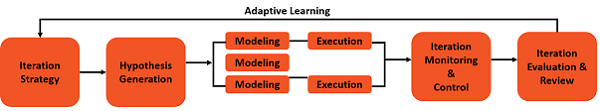
**Figure 5.6**: Adaptive Learning

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

It gives a perspective of learning and effective management. However, it is limited in its ability to enhance scientific/management capabilities for conditions that go beyond the course of action selected.

## 5.8.3 Active Adaptive Management

An Active Adaptive management style reviews the information before management actions.



**Figure 5.7**: Adaptive Learning Iteration

A range of, alternative system and related responses, rather than a single model, is developed. Management options are chosen based on the evaluations of alternative models.

## 5.8.4 Leadership-Collaboration Management

Adaptive management is a management technique which is best suited for Adaptive Software Development. The approach requires resource/people managers, those can work with people, allow human-interventions, and create an amicable environment for goals to achieve.

In software development, the leaders/managers/directors often take up these responsibilities. Leaders are not merely commanders. Leaders are collaborators and work with the team. Collaborative-Leadership (Working with the team) is the most important factor in Adaptive development.

Leader should have following qualities −

* Understand and set the direction for team.
* Should Influence people involved and provide proper guidance.
* He should Collaborate, facilitate and macro-manage the team. Here we should understand that micromanagement creates negative impact on team and should be avoided.
* Managers Provide direction with creative environments where talented people can be innovative, creative, and make effective decisions.
* Commanding should not be a predominant style but it is occasionally allowed.

**Adaptive Testing**

### ****5.8.4.1 Speculate****

A degree of certainty is indicated about the expected results by the plan. The Team Leaders ability of steering the project in innovative directions is restricted by the implicit or explicit goals of the plan conformance.

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

### ****5.8.4.2 Collaborate****

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

In order to produce accurate results, share knowledge and to make decisions, it is required to work in collaborative way.

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

### ****5.8.4.3 Learn****

For the ultimate success of the project work, Learning is an important part in the Adaptive (Dynamic Run Time) Development lifecycle. By using defined practices, the team has to enhance the knowledge by:

* Technical Reviews
* Project Retrospectives
* Customer Focus Groups

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

* Changes in the product and the assumptions of how products are developed then about product changes

# 5.9 Adaptive Automation Testing

Adaptive (Run Time, Dynamic) testing is the counterpart of adaptive control in software testing. It suggests that software testing strategy should be modified 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 (Run Time Dynamic) software testing is the counterpart of adaptive control in software testing. It means that software testing strategy should be adjusted dynamically 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 (CMC) model for software testing which produces several unrealistic assumptions.

Adaptive software testing approach in the context of an improved model that aims to eliminate threats to validity. Hence some new set of fundamental 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 an upcoming adaptive testing strategy is also developed and implemented. Thereafter Experimental data collected to demonstrate the effectiveness of the upcoming methodology.

DevOps as a trend is on the rise. Studies show that most of the organizations adopting DevOps continues to grow with every passing year. The core requirements in DevOps world is higher automation. So manual testing moving to automated testing in areas of operations. To this effect, automation teams has to create automation frameworks enabling easier and more effective automated testing to be taken up by one and all on the testing team. However one ongoing flaw in all of this, is 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 been taking shape where expectation to see measurable outcomes of around 100% automated effort (A test case when fully automated there is no manual effort involved), a tight coupling between defect management and automation execution, and considerable time savings, this is what we call the adaptive automation testing framework.

# 5.10 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.

Let’s check 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.

## 5.10.1 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**

## 5.10.2 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.

## 5.10.3 Proxy Patterns

Web Server

HTTP

Internet HTTP

Client Machine

HTTP

Internet

Server

HTTP Proxy

Cache Storage

**Figure 5.8**: Proxy Patterns

[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 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.

The proxy settings be controlled by environment by default. Adaptive Test Automation Framework recognizes the LOCAL, DevCloud environment, and Productions environments by default and has dedicated property files for each. There are many different environment definitions within organization to define property files. i.e. Default Proxy of cloud environment.

Whether or not a proxy is used is controlled by the presence or absence of two properties. 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

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

## 5.10.4 Business Patterns

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

## 5.10.5 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.

## 5.10.6 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.

## 5.10.7 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.

## 5.10.8 Singleton Pattern

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