**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 BrowserStack, 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:

**Adaptive software development** (**ASD**) is a [software development process](https://en.wikipedia.org/wiki/Software_development_process) that grew out of the work by [Jim Highsmith](https://en.wikipedia.org/wiki/Jim_Highsmith) and Sam Bayer on [rapid application development](https://en.wikipedia.org/wiki/Rapid_application_development) (RAD). It embodies the principle that continuous adaptation of the process to the work at hand is the normal state of affairs.

Adaptive software development replaces the traditional [waterfall](https://en.wikipedia.org/wiki/Waterfall_model) cycle with a repeating series of *speculate*, *collaborate*, and *learn* cycles. This dynamic cycle provides for continuous learning and adaptation to the emergent state of the project. The characteristics of an ASD life cycle are that it is mission focused, feature based, [iterative](https://en.wikipedia.org/wiki/Iterative_and_incremental_development), [timeboxed](https://en.wikipedia.org/wiki/Timeboxing" \o "Timeboxing), risk driven, and change tolerant. As with RAD, ASD is also an antecedent to [agile software development](https://en.wikipedia.org/wiki/Agile_software_development).

The word *speculate* refers to the [paradox](https://en.wikipedia.org/wiki/Paradox) of planning – it is more likely to assume that all stakeholders are comparably wrong for certain aspects of the project’s mission, while trying to define it. During speculation, the project is initiated and adaptive cycle planning is conducted. Adaptive cycle planning uses project initiation information—the customer’s mission statement, project constraints (e.g., delivery dates or user descriptions), and basic requirements—to define the set of release cycles (software increments) that will be required for the project.

*Collaboration* refers to the efforts for balancing the work based on predictable parts of the environment (planning and guiding them) and adapting to the uncertain surrounding mix of changes caused by various factors, such as technology, requirements, stakeholders, software vendors. The *learning* cycles, challenging all stakeholders, are based on the short iterations with design, build and testing. During these iterations the knowledge is gathered by making small mistakes based on false assumptions and correcting those mistakes, thus leading to greater experience and eventually mastery in the problem domain.[[1]](https://en.wikipedia.org/wiki/Adaptive_software_development#cite_note-1)

Adaptive software development is a design principle for the creation of software systems. The principle focuses on the rapid creation and evolution of software systems.  
The adaptive development method grew out of the rapid application development method.  
Adaptive Software Development replaces the traditional waterfall cycle with a repeating series of *speculate*, *collaborate*, and *learn* cycles.   
  
ASD is made of three steps (*speculate*, *collaborate*, and *learn*) . So , here those steps described briefly:   
Speculate  --> Initiation and Planning  
Collaborate --> Concurrent feature development  
Learn -->  Quality Review



An adaptive software development approach is based on an empirical process control model. The word "empirical" means "based on observation" and that means both the design of the solution and the process to create the solution are continuously adjusted based on observation throughout the project.

* All Agile methodologies are adaptive in nature and an adaptive approach encourages changes throughout the project to optimize the design of the overall solution.
* The opposite of "adaptive" is "plan-driven". A plan-driven project attempts to define and stabilize the requirements for the project prior to the start of the project and control changes once the project is in progress.

An adaptive process is best-suited for projects with higher levels of uncertainty. An example that I like to use is: if you were to set out to find a cure for cancer, it would be somewhat ridiculous to try to develop a detailed plan for the effort upfront because there is so much uncertainty. That’s an ideal situation for an adaptive process - you start with general direction based on whatever you know and continue to refine the direction of the project based on observations throughout the duration of the project.

Adaptive Software Development (ASD) is a software development process that grew out of rapid application development work by Jim Highsmith and Sam Bayer. It is a design principle for the creation of software systems. The principle focuses on the rapid creation and evolution of software systems. There is never a period where the software is finished; there are just stable periods between new releases. Adaptive software development grew out of the rapid application development method. These two methods are similar in structure, but rapid application development allows for a time when the project is finished, while adaptive software development doesn't.  
  
Adaptive Software Development replaces the traditional waterfall cycle with a repeating series of speculate, collaborate, and learn cycles. This dynamic cycle provides for continuous learning and adaptation to the emergent state of the project. The characteristics of an ASD life cycle are that it is mission focused, feature based, iterative, timeboxed, risk driven, and change tolerant.  
  
The focus of adaptive software development is in the computer code. Instead of planning the software out before hand, developers have a basic idea in their heads and they go to work. When pieces need changing or adapting to a new system, the coders simply do it. If the program needs a patch, somebody just makes it.  
  
Overall, the lack of pre-planning steps allows the developers to make the software very quickly. While this will occasionally result in software that doesn’t perform the precise functions required, that is generally not a problem. The developmental cycle in this process is so short that a new version with additional features can come out very quickly. This process or rapid prototyping is the cornerstone of both adaptive software development and rapid application development.

The earlier SDLC models are more oriented to the practices of stability, predictability and decreasing returns. The industry, such as the Internet Platforms has been moving to increase return environments, unpredictable, nonlinear, and fast approaches.

Adaptive Software Development (ASD) has evolved to address these issues. It focuses on emergence as the most 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.

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.