**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: