International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Hybrid Automation Framework: A Core Strategy to Decrease Testing Effort in Automation

Srikanth Kavuri

Srikanth.kavuri@ieee.org
Independent Researcher

Abstract

Automation testing is a key element of contemporary software development, which serves as the backbone for testing efficiency and product quality improvements besides being able to cut down the costs. This thesis presents a very convenient strategy towards the Hybrid Automation Framework, using various templates, thus, significantly decreasing the effort for practiced automation testing. Through the Hybrid Framework, this research could reduce the amount of hard and exhausting test case scripting, as well as the highly reusable scripts and the efficient modularization of scripts.

The outline of the Hybrid Automation Framework is made of the main principles of test automation, which are test independence, idempotency, and clarity that are very fundamental when building a robust testing suite. Moreover, the framework has a modular structure and a Page Object Model (POM) design pattern to facilitate flexibility and maintainability across disparate testing environments. The basic elements like the Object Repository, Name Mapping, Functions Library, and Data Manager are the nucleus of the framework, which allows smooth test management and proper execution.

The framework's architecture is designed to allow for automation that can be more dynamically handled across different interfaces, such as the Command Line Interface (CLI), the Application Programming Interface (API), and the Graphical User Interface (GUI). Logging and reporting are the basic functions that help failure analysis become effective and the reporting system can be adjusted to determine how the test report is distributed to the stakeholders. This Hybrid Automation Framework, in the end, adds the aspect of full automation testing that permits the testers to carry out the critically important points, thus making it consistent while at the same time cutting down on the manual intervention. The proposed framework is expected to deliver the software test automation system that is scalable, flexible, and performance-driven, which is the methodology suitable for every organization.

Keywords: Hybrid Automation Framework; Automation Testing; Page Object Model (POM); Modular Structure; Application Programming Interface (API); Graphical User Interface (GUI); Logging and Reporting; Failure Analysis; Scalable Automation; Flexible Automation; Performance-Driven Testing; Software Quality.

1. Introduction

A combination of two or more frameworks approaches makes up a Hybrid Framework. In this thesis we are adopting same technique, pulling the strengths of different frameworks and trying to mitigate their

weaknesses. In the below section hybrid approach is described. This document intends to explain the details and specifications of the Hybrid framework to be used for any Product Suite. A Test Automation Framework is a set of assumptions, concepts and best practices that provide support for automated software testing. The Framework is an attempt to simplify and speed-up scripting by reusing functionalities. The goal is to provide a complete end-to-end automation solution for testing the AUT.

The document has the framework architectural diagram and its required components / layers. The framework will be designed specifically to cover the Product components and will contain the features to automate product faster. The details of components are covered below. Automated Testing can simply be considered as a solution to decrease costs in software testing and for achieving high quality in software products. Just like any other matter of life, in test automation also, an initial effort is required in order to gain advantages from it. These goals can be achieved if concentration is put on the implementation of automation project through best practices. The first and second phase of automation is quite sensitive and requires more human input for developing a mature process or we can say that automation testing requires care in beginning. The most important things which initially require attention are identification of test cases to be automated, selection/development of automation tool and modes of automated testing. If adequate care is applied on the initial phases of test automation and best practices are followed then consequences will be nothing less than time saving, high efficiency flexibility and usability. Automation Testing is not an alternative to manual testing.

2. Automation Testing Principles

Before writing the automation scripts we need to consider the following:

a. Test cases are pre-written and not ambiguous.

Means that the test cases clearly show the exact functionality of test. And test cases did not give several meanings, before start writing automation scripts we need to consider the test case. If test case is not written we need to write test cases before starting automation also the test cases should be crystal clear and should not give several meanings.

b. Test cases are autonomous

Means that each test case should not depend on the other test cases. Every test case can be run autonomously.

c. Each test case can run independently.

Each test case is independent means that prerequisites should also cover in all the test cases and test case reverse all the changes performed during testing the object or application.

d. Test cases may not cover end to end scenarios.

Means that the test case document should cover all end-to-end scenarios. Nothing should remain after completion of test cases document. It also gives the meaning of covering all scenario by opening the first door first and closing the first door at last.

e. Keep test tests short

Test cases should be written preciously and consistently. Each test case is specific and cover more details in shorter words,

f. Each test case can cover only one functionality.

Means that every test case is designed to cover only one functionality. We may have happy scenarios in covering more than one functionality but in writing test cases we should acknowledge this principle, if we have more functionalities to be cover then we need to redesign the test cases or we can add more test cases.

g. Test cases are idempotent

Means that the test case may call as many times as required. If we want to call one test case or number of test cases for example 1000 times our test should be flexible enough to maintain this.

h. Minimize incident Test coverage.

A basic rule of incident prevention and response is to plan for the worst and hope for the best. For overcoming external incident we need to have dedicated server environment to minimize the test external event with only necessary software installed.

i. Automation testing tool selected via proper research.

Several companies use different automation testing tools but you need to analyze your company requirements or your product first. A proper POC is required to for tool selection and automation. The rule of thumb is that before selecting the tool look at the coverage and help available online. Or some training material. Then go for it.

j. Coding standards should define for script writing.

Coding standards are necessary for analyzing and optimal code. You need to define coding standards it also depends on the language you have selected for writing scripts

Some practitioners believe that manual testing can be done as pre-requisite of automation testing. What we actually get through automated testing has been explored by many researchers, whereas this paper aims to present a solution of this common query ‘How to gain benefits from Automated Testing.’

3. When to automate?

A lot of work is to be done before you actually begin test automation. It must be guaranteed that following things have been verified before we jump into test automation:

Architecture and flow

Automation framework should be designed to maintain test scripts and minimize efforts. Each framework should have following 3 properties. These are basically properties of framework.

1. Reusability
2. Maintainability
3. Flexibility

2.1 Architecture Diagram

Hybrid automation testing framework can be shown as follows:

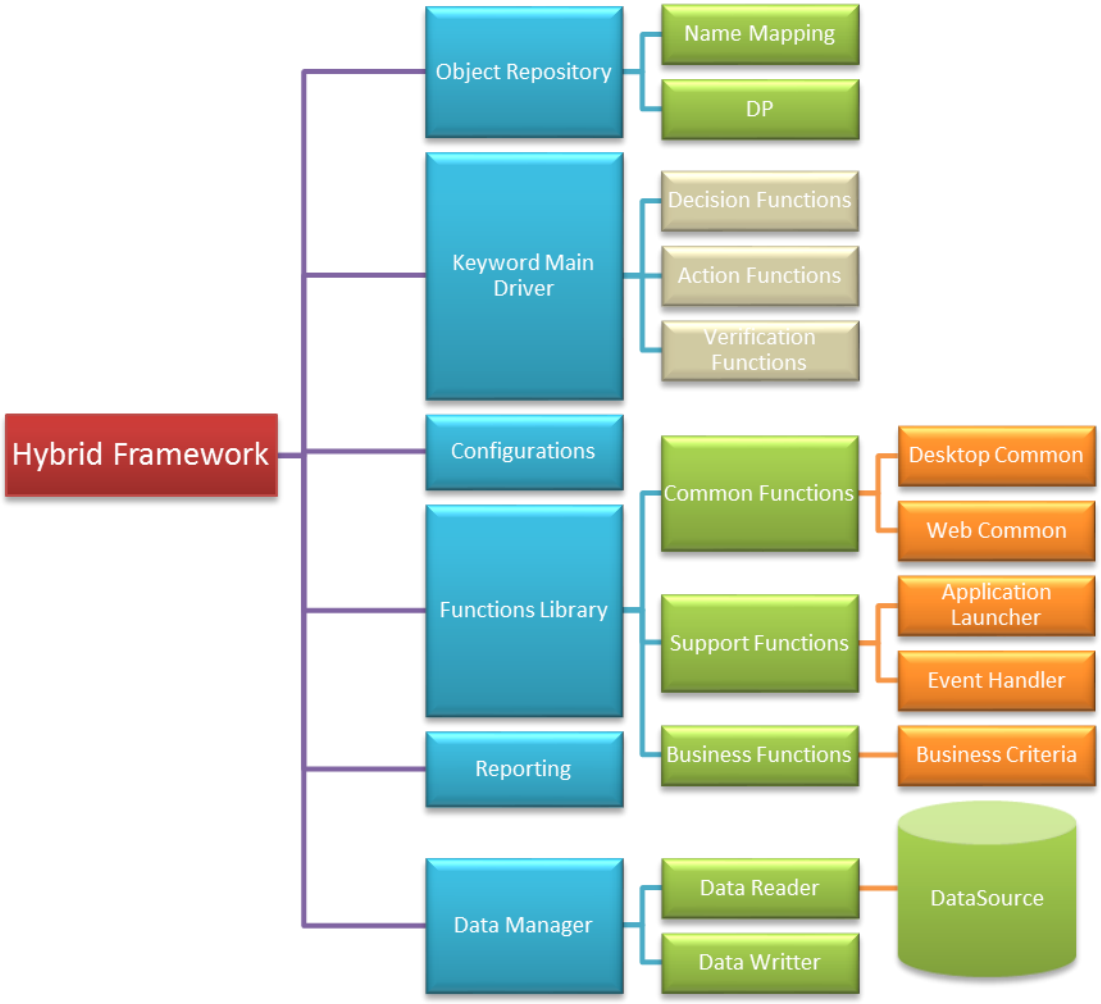

Figure 1: **Architecture diagram of Hybrid Automation Testing Framework**

1. Control Flow of Hybrid Automation Testing Framework

The below diagram describes the flow of control among different components. Execution starts with batch, events are triggered with execution of each test where Application Launcher checks the status of AUT. The configuration setting are loaded for the test and the execution of steps are enquired which use different components like Data Manager, Functions Lib, Object Repository to perform action on AUT and report the logs.

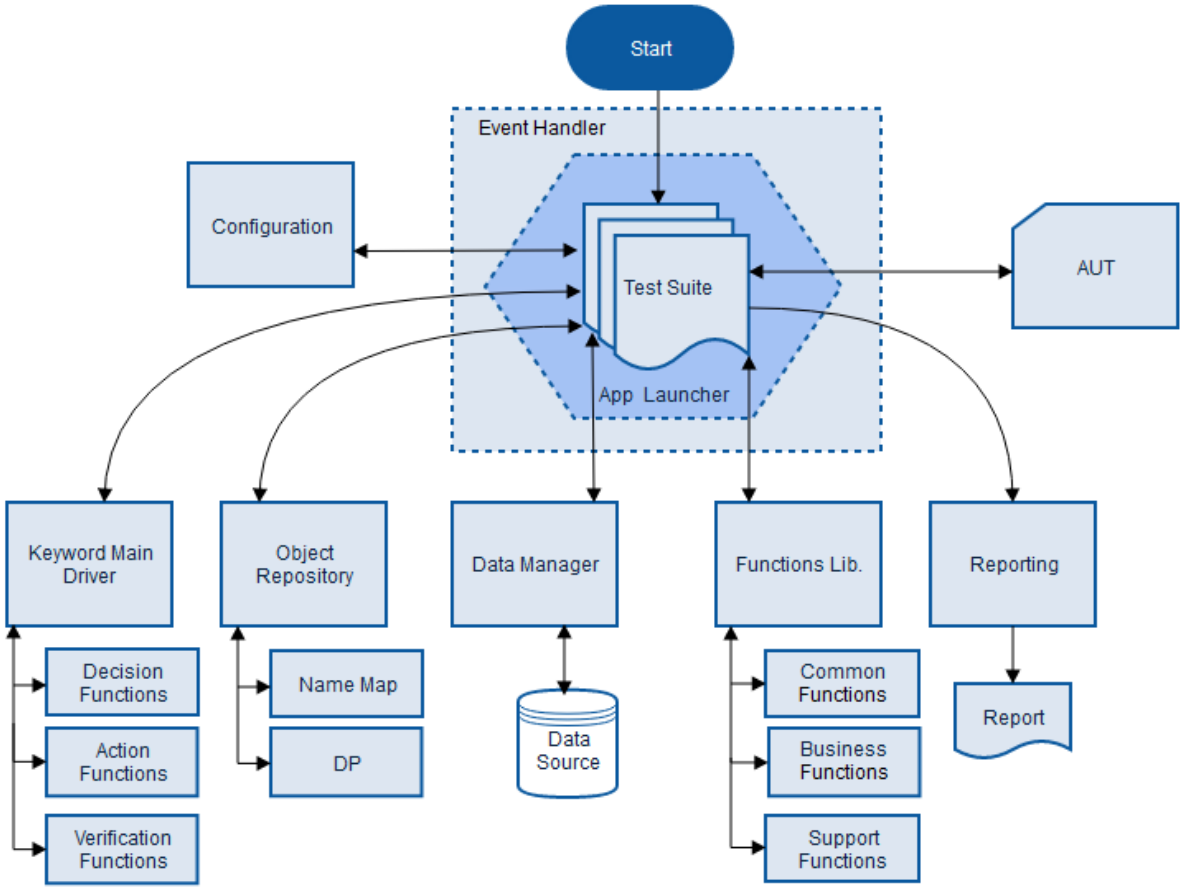

Figure 2: Control Flow of Hybrid Automation Framework

2. Object Repository

Object Repository is a collection of objects. OR saves objects with their properties in a container. OR can be maintain inside the testing application, or via some external resource like in excel or in DB

3. Name Mapping

In testing application, Name Mapping objects repository is a tree style container of AUT objects. It has two inner containers. It will return the complete hierarchy of the object i-e from the root (parent) to the object. This is relatively faster approach but sometimes this approach will not work if the hierarchy is changed this approach require remapping of all the objects for example if parent object changed the hierarchy of all the objects also changed. To avoid this we will use DP.

4. Mapped objects

Mapped objects allow us to map/store the AUT objects with their original hierarchy.

5. Aliases

Aliases allows us to reduce the original complex hierarchy. Objects in the Mapped Object repository has also a user defined names (aliases) that are used in the tests to make them more readable. We will map

the complete hierarchy in some shorter name for example the hierarchy Browser("Chrome"). Panal(0).abc.axc.xyz.username can be mapped into Aliases.username
In our framework only static objects will be in Name mapping

6. Descriptive Programming

Application objects can be accessed (even if they are not present in object repository) by describing properties directly in a script. This technique is called 'Descriptive Programming'. DP normally comes into play when objects in the application are very dynamic in nature, and so will be in our framework or we are working on waterfall model and continuous integration testing should be the main requirement.

7. Functions Library

Functions Library facilitates users by providing different functions. This library is basically divided into three parts:

8. Common Functions

Common functions provide facility to the user to get the common functionality in one place and it increases reusability and maintainability of the code. Basically we are will make our frame work maintainable by using modular approach. In this approach we will divide main flow in several modules dependent on AUT. For example we can divide Common functionalities to base common, business common and several other dependent of the AUT.

Name	Description
SelectDropDownItem	It's a function which can be used to select item from drop down List. **Usage:** SelectDropDownItem(Object,Val)
GetSelectedItemValue	It's a function which can be used to get selected item from drop down list. **Usage:** GetSelectedItemValue (Object)
SetText	It's a function which can be used to set text into required object. **Usage:** SetText(Object,Val)
GetText	It's a function which can be used to get text from required object. **Usage:** GetText(Object)
FindRow	It's a function which can be used to find row number from Grid/Table. **Usage:** FindRow(Object,ColName,Val)
ClickCell	It's a function which can be used to click on specific grid cell. **Usage:** ClickCell(Object,RowID,ColName,Val)
DoubleClickCell	It's a function which can be used to double click on grid cell. **Usage:**

	DoubleClickCell(Object,RowID,ColName,Val)
ClickTab	It's a function which can be used to click on tab control. **Usage:** ClickTab(Object,Item)
ClickRibbonButton	It's a function which can be used to click on ribbon. **Usage:** ClickRibbonButton(ButtonName)
ClickRibbonMenu	It's a function which can be used to click on ribbon menu item. **Usage:** ClickRibbonMenu(Object,MenuItem)
ClickToolBarButton	It's a function which can be used to click on tool bar button. **Usage:** ClickToolBarButton(Object)
clickMsgPopups	It's a function which can be used to click on MsgPopups button. **Usage:** ClickMsgPopups(Object,buttonname)
getRowCount	It's a function which can be used to get page row count from grid. **Usage:** GetRowCount (Object)
getCellValue	It's a function which can be used to get cell value from grid. **Usage:** getCellValue(Grid,RowIndex,ColIndex)
getColIndexByName	It's a function which can be used to get column index by its name from grid **Usage:** getColIndexByName(Grid,colName)
getColumnCount	It's a function which can be used to get column count from grid **Usage:** getColumnCount (Grid)
getDropDownValueInGrid	It's a function which can be used to get drop down value from grid **Usage:** getDropDownValueInGrid (Grid,rowIndex,ColName)
selectDropDownValueInGrid	It's a function which can be used to select drop down value from grid **Usage:** selectDropDownValueInGrid (Grid,rowIndex,ColName,Value)

Table 1: Base Common Functions

9. Support Functions

These libs facilitates other framework libs by providing common functionalities. Like events required for initialization of different libs. Support functions are divided into sub libs

Events

An event is a specific action that occurs during the test run. There are different events that can facilitate user to handle different situations.

Name	Description
OnLogMessage	It occurs when an informative message is posted to the test log. **Usage:** GeneralEvents_OnLogMessage(Sender, LogParams)
OnLogCheckpoint	It occurs when a checkpoint message is posted to the test log. **Usage:** GeneralEvents_OnLogCheckpoint(Sender, LogParams)
OnLogError	It occurs when an error message is posted to the test log. **Usage:** GeneralEvents_OnLogError(Sender, LogParams)
OnLogEvent	It occurs when an event notification is posted to the test log. **Usage**: GeneralEvents_OnLogEvent(Sender, LogParams)
OnLogWarning	It occurs when a warning message is posted to the test log. **Usage:** GeneralEvents_OnLogWarning(Sender, LogParams)
OnStartTest	It occurs when TestComplete starts a test run. **Usage**:GeneralEvents_OnStartTest(Sender)
OnStopTest	It occurs when a test run is over. **Usage**:GeneralEvents_OnStopTest(Sender)

Table 2: Event Handler Functions

10. Application Launcher

Application launcher starts the application (AUT) based on the status of the application. It checks if the application is already running or not, also checks the network and database availability.

Name	Description
Application Launcher	It's a function that starts the application, checks the availability of network and the Database. **Usage:** ApplicationLauncher(Process Name)
Network Availability	It's a function which checks the Network Availability. **Usage:** NetworkAvailability()

Database Availability	This function is use to check Database Availability **Usage:** Function DatabaseAvailability()

Table 3: Application Launcher Common Functions

11. Business Functions

Business Functions will be used to facilitate the users by providing core application functionalities in the form of functions. These business functions will increase reusability and provide ease to perform complex business operations like calculations. This functions will grow with the passage of time.

12. Data Manager

Data manager is used to manipulate the data. Different operations will be performed through data manager like read and write.

Name	Description
GetDriver	This function will be used to create a driver **Usage:** GetDriverInstance(DataSourceType,DataSourceName)
SetSourceData	This function will set a data source and filter can also be applied on that data source. **Usage:** SetSourceData(DriverObj,DataFilter,DataFilterColumn)
GetItemValue	This function will get data from a particular cell of Recordset. **Usage:** GetItemValue(RecordSet, ColumnName, RowId)
GetRowCount	This function will return the number of rows of RecordSet. **Usage:** GetRowCount(RecordSet)
GetColumnCount	This function will return the number of columns of RecordSet. **Usage:** GetColumnCount(RecordSet)
GetItemRowIndex	This function will return the row number of RecordSet. **Usage:** GetItemRowIndex(RecordSet,ColumnName,Value)
GetColumnIndex	This function will return the column number of RecordSet. **Usage:** GetColumnIndex(RecordSet,ColumnName)
DataWriter	This function will write the data in data source. **Usage:** DataWriter (DataSourceType, DataSourceName, RowId, ColumnName, Value, Appendmode)

Table 4: Data Manager Common Functions

13. Decision Functions

Decision functions are used to decide which part of code should execute and what are the bases of execution, following decision functions are implemented in automation framework

Name	Description
EvaluateObjectType (Decision)	It is used to decide the execution based on the ObjectType Keywords. **Usage:** fnSet (object, Action, Data) Note: Few functions will be developed such as fnClick, fnVerifyValue, fnVerifyExists etc
EvaluateAction (Decision)	It is used to decide the execution based on the Action Keywords. **Usage:** EvaluateAction (object, Action, Data)

Table 5: Decision Common Functions

14. Action Functions

Action Functions is a set of functions/methods providing the ability to perform the required action/activity

Name	Description
FnTextBox_Set	It provides the ability to perform the required action/activity e.g. Textbox Set etc. It is the actual implementation of performing the actions. **Usage:** FnTextBox_Set (object, Data)
FnComboBox_Set	It provides the ability to perform the required action/activity e.g, Combobox Set etc. It is the actual implementation of performing the actions. **Usage:** FnComboBox_Set (object, Data)

Table 6: Action Common Functions

15. Verification Functions

Verification functions is a set of functions providing the ability to perform the required verifications.

Name	Description
FnVerifyValue	It provides the ability to perform the required verifications e.g. VerifyValue etc. It is the actual implementation of performing the verifications. **Usage:** FnVerifyValue (object, Data)
FnVerifyExists	It provides the ability to perform the required verifications e.g. VerifyExists etc. It is the actual implementation of performing the verifications. **Usage:** FnVerifyExists (object, Data)
FnVerifyState	It provides the ability to perform the required verifications e.g. VerifyState etc. It is the actual implementation of performing the verifications. **Usage:** FnVerifyState (object, Data)

Table 7: Verification Common Functions

16. Configuration

Configurations facilitates to declare and initialize variables necessary for different setups like paths, folder names etc. Configuration file is the start-up file which automation framework loaded automatically at startup and set their environment variables and several paths i-e database path, data files path, opject repository paths etc.

Characteristics of Hybrid Automation Framework

1. Introduction to Hybrid Automation Testing Framework

Automation framework should be designed to maintain test scripts and minimize efforts. Each framework should have following 3 properties

1. Reusability
2. Maintainability
3. Flexibility

Our hybrid approach have few more:

1. Adoptability,

2. Hybrid (Adopt the good things of other frameworks and remove the limitations of other frame work)

2. Reusability

Reusability can be achieved with common functions. The formula for writing common function is simple. If you want to write code lines more than one you need to make a common functions Scripts Management "App Common functionalities" are those that are the same in whole application for example click on Title Bar button from top of the application is common function we use it by giving specific name (which may vary in different screens) in parameter to achieve more reusability. The advantage of common function is reusability. All common functions are written in desired Scripting language and it will be generic for all common buttons, forms and dropdown etc.

Following are some rules should be followed while creating common functions:

1. Base Common functions are those functions which are used throughout the application, each function should describe in separate unit/file/class. Other functionalities of that particular function for example dialogue handling etc. should also describe in the same class.

2. Module Common functions are those functions which are used for that particular module only and should never be used in outside the module or some other module each function should describe in separate unit/file/class. Other functionalities of that particular function for example dialogue handling etc. should also describe in the same class.

3. The naming conventions of common functions should be according to some pre-defined standards for example function name can be written as Pascal case like ClickButton() is a perfect example of a function which can click on desired button

3. Maintainability

We can achieve maintainability by using modular approach, every test case placed in its particular module as well as its common functions can also retain in its particular module folder, also each test case should assign a particular name which has ATS(Automation Test Script) and name should same as in test case document, but the naming convention should be change. For example test case starts with FTC (Functional test case) in test case document but in script FTC should be replaced as ATS (automation test script) in automation framework.

4. Flexibility

Flexibility refers to the change adaption. Your framework should be flexible enough to adopt change in environment hybrid means power of adoption, at some point your framework is keyword driven and some other point it is data driven means your framework is hybrid and flexible enough to adopt changes. Flexibility can be achieved by abstraction; we need to put abstract layer between actual functionalities and driver object for example function gotohomepage() lead you to the home page without asking you anything for example if your home page is 125.209.79.85 or mail.bahriatown.com.pk user should no need to know what is actual url.

5. Adoptability

Adoptability refers to the change in environment. Our hyrid automation framework is flexible enough to adopt the environment changes and modify its functionality accordingly for example same automation frame may be used in desktop and web AUT. This can also refer to the adaptation of development environment for example the AUT is designed and developed in silver light and later it will be converted to junit the same hybrid automation framework adopted the changes of development environment.

6. Hybrid

This is the main characteristic of our automation framework. Hybrid refer to the mixture of several framework depending on the environment and requirements of AUT. This approach lets you to get good things from other frameworks and discard limitations of other framework.

Design Pattern Involved

1. Design Pattern (POM)

The framework has been designed by the inspiration of Page Object Model (POM). Page object model is very common in selenium, by using the page object model we can make non-brittle test code and reduce or eliminate duplicate test code (scripts). Beside of that it improves the readability and allows us to create interactive documentation. This POM also suggests us to use Object Oriented approach and give us the concept of wrapper classes besides every implementation class. The above frame work is basically enhancement of POM. I have called the above framework is a hybrid approach which will cater the functionalities of Data Driven, Modular Driven at the same time. This framework will cater the following Objectives:

2. Page Object Model

Page Object model is already defined in this thesis below, It is basically a design pattern used to develop automation framework.

3. Object Repository

Object Repository is collection of Objects in AUT, we can create Object repository in TestComplete's built-in name mapped or we can put this repository in some external file for example excel or database. Object repository can be saving as the complete tree form of hierarchy of the desired object or it can be the properties and values for DP method. In hybrid AF we support object repository in some external resource file this will help us in migration of tool or can be easily update if the object is changed (remapped)

4. Modular Driven Technique

Our Modular driven approach enhances the maintainability of project. Application under test is divided in several modules depending upon the business of application. Each part of application is sub divided into several module and each module is further divided into several modules depending upon the application behavior. For example we have a module name common we can further divide this module into application common, business common, support common. This is how we can achieve the modularity of our framework.

5. Function Parameters

Functions parameters are defined to make functions more generic. This is another approach to minimize the coding redundancy. We can use function parameters to identify the objects resides in the same hierarchy. For example if there are several buttons in the ribbon TAB and the hierarchy from parent to child is same for all the objects we can use function parameter index or a content text to identify the several objects and create only one function for all the ribbon TAB objects.

6. Constants

Our framework also supports constants. These are the values which remains same throughout the program execution. The suggested approach is to use constants from some external resource this is the best way if you want to migrate code from one tool to other the frame work supports adoptability.

7. Data Driven Technique

Frame work also support Data Driven approach but it is based on the object oriented approach. Therefore we have written a wrapper class in which get, set functions those get set functions interact with driver object and that driver might be the excel file driver or might be the SQL driver. If the driver is changed and user wants to migrate data source it will not affect the functionality due to the wrapper class. User just call the get data function and this is the responsibility of getData to return appropriate data even user does not need to change the driver it just need to change the configuration file before running the test suite and appropriate driver automatically loaded.

8. Logging

Logging is essential part of hybrid test automation framework, there are two types of logging

1. Explicit
2. Implicit

In Explicit logging we can log exception messages, failure causes and other calculation scenarios. We also can log Expected and actual values in explicit logging in case of failure.

In Implicit logging we can log actions and may get some internal data to figure out the exact cause of failures etc.

9. Reporting

Reporting is the part in which client is interested. Report shows the output of test cases, logs failure and warnings. There are two kinds of reports:

1. Auto Generated Report
2. Custom Report

Auto generated reports are more detailed report in which each and everything is described briefly and along with screenshots. Size of auto generated report is very big and sometimes client is not interested in details he only wanted to see the count of test cases at that time we use custom reports

In Custom reports we give abstract overview of report. Following is an example of custom report

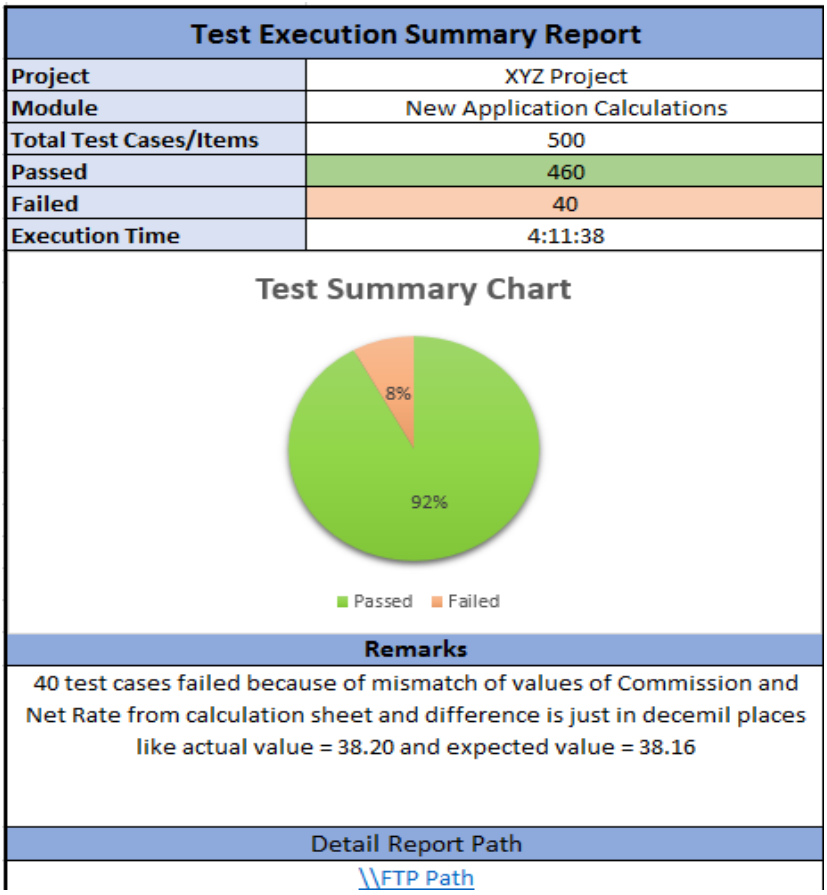

Figure 3: Custom Report

10. User Defined Functions

User Defined functions are those which are provided to the program by user to perform specific tasks. In normal execution environment those functions accept parameters, perform operations and return results. They might be designed to perform complex calculation tasks

11. Exception Handling

Exception handling is the process of responding to the occurrence, during computation, of exceptions – anomalous or exceptional conditions requiring special processing – often changing the normal flow of program execution. It is provided by specialized programming language constructs or computer hardware mechanisms.

Scripting

1. Scripting

Test case scripting will be done using 'scripts Test, User use 'Common functions' by adding unit references and will change the value of parameter if require to make an automated test case.

2. Check points

Check points are used for verification and validation. Check points should use where necessary but inside the code and specific. Check points should not be generic.

3. Exclusive Functions

Exclusive functions are those when any user wants to use some unique function in some other screen or forms that will be not present in common function script unit then user must create a script unit in the folder Exclusive Functions with name script name Function_Name_Test_Case_Name (Test Case in which exclusive function will be use) folder for call exclusive function in his keyword test script.

4. Script Validation Checklist

Validation check list should be design according to the standard. Following is some example of validation check list.

Script Validation CheckList

Areas	CheckList	Pass/Fail
	Does the Script Executable?	
Inputs		
	Does the script implement each step of the test case?	
	Does the script implement each step in the correct order?	
	Does the script implement each validation given in the test case?	
	Does the script have each validation after the correct step?	
	Does the script have executable values?	
	Does the script have Error Logging messages?	
Execution		
	Does the Script have Screen Names/ 'Values' in capital letters?	
	Does the script logic call the correct functions? Are these functions called with the correct arguments?	
	Does the script give the application sufficient time to transition to the correct test state e.g. the correct screen?	
	Does the script validate application output values against the correct output test data file(s)?	
	Does the script validate the state of the correct objects?	
	Is the script free from unnecessary delays?	
	Does the script handle application errors as designed?	
Reusability		
	Are the functions that can be commoly used defined in 'Common File'?	
	Are there proper comments included for the modifications done in a 'Common File'?	
Miscellaneous		
	Does the script have a unique name (As suggested)?	
	Is the script commented at each appropriate place in it? i.e. Test case mapping is done with Test Script	
	Is the script free from unnecessary or extra code?	
	Are there variables that are declared but never used in the Script?	

Figure 4: Check List

5. Performance of application under testing:

Automated testing is never going to pay back good if the application under testing is subjected to change every now and then or various functionalities are unstable. Automated testing is considered best for those applications which have already gone through some early cycles of testing and are quite mature. The changing behaviors cannot be handled by automated testing, even if there is a way to handle them, it will not be worthy. Dealing with changing behaviors does not refer to the context of inability of automated testing. In many cases automated testing is done to see change in error messages, change in GUIs or some modifications in calculation based fields. The applications which are facing structural changes are not considered good to be automated.

6. Identification of interface layers:

A software product may have different interfaces like command line interface (CLI), application programming interface (API) and graphical user interface. Some applications are developed with three

of these interfaces and some have just two. In testing sense if we compare these interfaces, CLIs and APIs are quite easy to automate as compared to GUIs. It has to be decided earlier that which interface is to be automated first.

GUI automation is difficult due to following reasons:

GUI's automation requires manual script writing.
GUI's behavior is dynamic in nature.
GUI changes more frequently.
Execution is slow on GUI.
GUI's test scripts require more maintenance.

Coding Standards

The main reason for using a consistent set of coding conventions is to standardize the structure and coding style of a script or set of scripts so that you and others can easily read and understand the code. Using good coding conventions results in precise, readable, and unambiguous source code that is consistent with other language conventions and as intuitive as possible.

1. **Coding Standards**
    a. Indentation
1. Indentation of a code is a basic rule to improve the readability of code
2. Code in all routines/ functions will be indented with single tab. Code inside programming constructs such as If...Else, Switch, For, and Do...Loop will be indented one additional tab. Nested constructs will be indented an additional single tab for each level of nesting e.g.

```
function whileloop(){
  i=5
    while(i<10){
      Log["Message"]("while loop" +i)
      i++
     }
}

function dowhileloop(){
   i=5
    do
     {
      Log["Message"]("while loop" +i)
      i++
     }
    while(i<10)

}
```

b. Common Functions

Now we will have more than one function in one Unit, however in one unit all the related functions (e.g. same module functions) will be written

c. Function Names PascalCase

1. Functions name should be meaningful
2. Functions name should be written in Pascal case i.e. first letter must be in upper case e.g. "AddFlatCancellation"

```
function AddFlatCancellation()
{
ShowMessage("Hello")
Log["Error"]("error message/ Test Case Fails")
}

function UpdateFlatCancellation()
{
ShowMessage("Hi")
}

function DeleteFlatCancellation()
{
ShowMessage("World")
}
```

3. For longer Function names we should be using meaningful abbreviations. An abbreviation should not be greater than 5 characters. Moreover, for two characters abbreviations name should be in all caps and for greater the name should be in Pascal case.

d. Use of Prefixes

4. Use "Can", "Is" and "Has" prefixes with Boolean functions e.g. IsEnable()
5. Append computational qualifiers to function names like Average, Count, Sum, Min, and Max where appropriate e.g. SumTotalAmount()

e. Hungarian Notation

6. Do not use Hungarian Notation! E.g. strName or iCount

f. Class/Unit Name in Pascal Case

Unit names should be meaningful in Pascal cases. Also the proper hierarchy of the units should be established according to the pattern they are shown in the application.

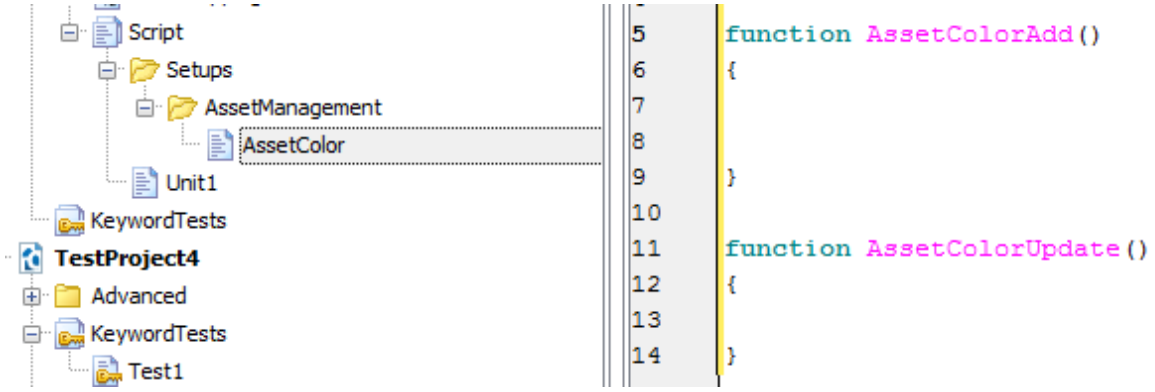

Figure 5: Common Function

User will create common functions that will be commonly used in different modules also any code that is repetitive even within the unit a separate function will be written for that.

Moreover, there should be specific functions written for all same type of controls in the application.

g. Code Organization

1. Keep modules to a manageable size. If module become larger split into functional criteria.
2. Small, cohesive modules are more manageable and easier to understand
3. Any if or else block should not be empty.
4. Apply spaces liberally. Spaces break up the code, making it easier to read.
5. There should not be any unreachable code in the script
6. Add comments for anything ambiguous in script like use of abbreviations.
7. In case of multiple if else statements, use switch statement.

h. Variable Names

Following rules will be applicable for both local and Global variable

1. Variable Name should be Descriptive and Meaningful.
2. Array should be declared using “Arr ” prefix
   Example: “arrName(Camels case) for local variables” and “ArrName(Pascal case) for global variables”
3. Local Variables should be written in camel case i.e. addButton
4. Global Variables should be in Pascal case i.e. AddButton.
5. For longer variable names we should be using meaningful abbreviations. An abbreviation should not be greater than 5 characters. Moreover, for two characters abbreviations name should be in all caps and for greater the name should be in Pascal case. Also comments should be provided where abbreviations will be used
6. Use “Can”, “Is” and “Has” prefixes with Boolean variables.
7. Append computational qualifiers to variable names like Average, Count, Sum, Min, and Max where appropriate.
8. Do not use Hungarian Notation! Example: strName or iCount

i. Comments and Headers

1. Add proper meaningful comments between coding statements.
2. Use Proper English While Adding Comments. Don't use Abbreviations while adding comments.
3. On top of each script or function there should be a General information header. It should contains information such as description, summary, author name, reviewer name etc.
4. This is very important that you explicitly mention the mapped FTCs ids in Script information header, since one ATS can mapped to multiple FTCs so use will mention all the FTCs ids that are covering in that particular ATS.

```
/*
SUMMARY:
How to USE:

AUTHOR NAME:
REVIEWER NAME:

UPDATED BY:
*/
```

Figure 6 Function Information Header

```
/*
DATASHEET NAME:

AUTHOR NAME:
REVIEWER NAME:

UPDATED BY:

Mapped FTC's:
*/
```

Figure 7: Script Information Header

j. Debugging Statements

All the debugging statements should be removed from the final code.

Example:

```
ShowMessage("Test variable" + tmp)
```

k. Error Logging

1. Possible Errors should be handled using If Else blocks. Errors should be printed using log.error.

   Example.

   ```
   Log["Error"]("error message/Test Case Fails")
   ```
2. Make sure there should not be any "If" that has no "else"

Automation Scope/ Coverage

1. Defining scope of automation

Before getting on the road of automation testing, define the scope of automation testing. It needs to be decided what we actually want to achieve from this type of testing. The scope may include selection of testing type like functional testing, regression testing or acceptance testing. Sometimes particular features or certain test cases can be automated.

2. Identification of test cases to be automated

Test cases serve as the base for automation testing. We know it is quite obvious that we cannot achieve 100% automation testing. Therefore test cases which we want to automated should be identified. Automation certainly reduces manual testing after some time. It should be understood that automation testing is not the replacement of manual testing. As every test case cannot be automated, therefore following test cases should be dropped immediately from scope of automated testing.

1. Test cases that are lengthy and complex and require human involvement.
2. Test case which can take long time to be automated.
3. Test cases which cannot be reused.
4. Test cases for usability testing. Usability testing is supposed to be done in real time end user environment.

It's very important to consider those test cases which yield to high reusability, and are loosely coupled, less complex in nature, involves minimum human intervention and can be automated in small time.

3. Test Cases have been adjusted:

The test cases need to be adjusted for automation. Many testers write test cases for manual testing purpose. However the way human tests an application is definitely different than how a robot is going to test it. Test cases written for manual testing purposed should be fine-tuned so that original functionalities can be automated.

4. Selection of automation tool

There are many automation tools available in market. Every tool is not capable of performing similar testing tasks. Majorly automation tools have been categorized as Functional/Regression Testing Tools, Load Testing Tools and API Testing Tools. There are many articles available on Internet which guide about selection of automation tool. A very well organized approach is to conduct a Proof of Concept to evaluate different tool. The evaluation criteria is mainly based on factors like Cost, Language Support, Compatibility with other technologies, offered features, complexity to use, result log details and integration options. As there are different mediums of applications like Web, Desktop and Mobile, mostly tools are designed to support one medium or are available in single flavors. However there are some expensive tools which capable of testing on every platform. Companies also have an option to develop their own automation tool if they can't find anything in market which can add value to their work.

5. Choosing the right mode of automation

As there are many ways of automating your tests, choosing the right method is really important. Modes like Record and Play, Script Writing, Key Word generation are some common modes which are used for automation. It depend on automation tool which mode it is offering. The maintenance and strength of automated test cases depends on how we have scripted them. For GUI test cases record and play is a good option, as it is quite easier to script test scripts for GUI test cases. When it is about complex scenarios and backend testing script writing is probably the best option. It gives the liberty to create custom functions and custom frameworks.

6. **HOW to automate?**

After going through above mentioned process, the right approach has been identified, and now it is the time to get started with automation testing. But in order to reach the target a lot more needs to done.

7. Following scripting standards:

Automation testing can be called as mini development. Therefore some development standards specifically to automation project must be developed like we do in real software construction. It includes development of framework, development of scripting standards and development of procedures.

8. Identifying common actions:

There exists many common functionalities like record save, record addition, screen closure, deletion etc. which can be developed as a common function utilities. But before developing these common functions, all common actions need to be identified throughout whole applications. It will ultimately result in less scripting and more reusability.

9. Development of objects repository

Automation frameworks involves the development of object repositories through which testers are going to use objects of application under testing. This object repository can help in time saving for using these objects recursively throughout testing cycle. Test developers will be getting a readymade set of objects which just need to be called in current routine.

10. **Extensibility**

The automation suite should be developed in such a way that later on more and more test cases can be added in it. A time comes when there is a need for adding test cases of different platform. For example the testing team may begin automating the desktop application first, and then the web version of it. The architecture of automation suite should be designed to support addition of as many functions and test scripts.

11. **Custom Logs**

A common issue is what should we do when test fails? Failure analysis is mostly difficulty. A good automation suite should be capable of checking setup mistakes, setting up its environment and generate logs of its own. Logs should be developed as user friendly guides.

12. Test Batch for Execution

Automation suite should be capable of handling different type of test executions. In some case we have to execute smoke tests, sometimes regression test and sometimes functionality simulations. Therefore configuration of test batch should support multi way testing types.

13. Cleaning-Up

Once automation script has been executed, it has to be made sure that application is dragged back to its original state. Where execution entered a lot of data for testing purposes, at the same time automation should be capable of deleting or erasing all that data which it has added. Otherwise cleaning is an expensive job if it is to be done manually.

CONCLUSIONS & RECOMMENDATIONS

Conclusion

Test Automation is a great solution for making testing easier and valuable. Sufficient time and effort must be spent for the development of right automation framework and suite. The secret behind it is just to follow the right process. The process proposed in this paper will definitely lead towards gain of benefits which test automation is being implemented. And the only reason behind failure of test automation projects is the deviation from the right process.

Is it all beneficial?

Finally when we have done above mentioned things, finally it's the time to see what we are going to get out of whole effort. There reside real benefits which can be achieved if we follow this proposed process for beginning any automation activity.

1. There will be clarity in scope of automation. Testers will be confident about what they are going to automate.
2. *Testers will be sure about the level of testing they need to focus on.*
3. *Application under testing will not be vulnerable in behavior, which will ultimately help the testers to use scripts for longer time period.*
4. *Because application will be stable, scripts maintenance cost will reduce ultimately.*
5. *Testers focus will be more on inner functionalities rather than GUIs. This will save a lot of time in developing scripts for core functionalities.*
6. *The effort which was to be put in for development of complex and lengthy test cases can be put in development of smaller and easier test cases. It means more test cases will be covered in similar time period.*
7. *Selected tool will definitely adhere to all testing requirements*
8. *More reliable test cases will produce through which testers will ultimately target more sophisticated areas to be tested.*
9. *If test cases are fine-tuned then test developers can also develop fine test scripts.*
10. *It will be easier to extend the automation suite for test cases related to other versions of applications.*
11. *More consistent test cases and test scripts will be developed throughout whole testing cycle.*

12. *Re-Usability factors will definitely increase. Test Developers can access more objects and data, quickly and use easily.*
13. *A lot of manual object creation activity will be decreased.*
14. *Faults analysis will be easier with help of customized logs.*
15. *15) The manual effort in script cleaning will be decreased eventually with development of clean-up scripts.*
16. *There will be clarity in scope of automation. Testers will be confident about what they are going to automate.*

References

1. Fernandes, J., & Di Fonzo, A. (2010). When to automate your testing (and when not to).
2. *A Literature Review on Automation Testing Using Selenium+Sikuli*. (2022, January 1). SciSpace - Paper; IGI Global eBooks. https://doi.org/10.4018/978-1-6684-3694-3.ch030
3. ADNOC OFFSHORE, & Mitra, D. (2022). Organization Need Based Specific NDE Development Application Research on Key Performance Areas (KPA). *E-Journal of Nondestructive Testing*, *27*(11). https://doi.org/10.58286/27473
4. Alavi, S. A., & Noël, M. (2024). Challenges for the Development of Artificial Intelligence Models to Predict the Compressive Strength of Concrete Using Non-destructive Tests: A Review. In R. Gupta, M. Sun, S. Brzev, M. S. Alam, K. T. W. Ng, J. Li, A. El Damatty, & C. Lim (Eds.), *Proceedings of the Canadian Society of Civil Engineering Annual Conference 2022* (Vol. 367, pp. 839–857). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-35471-7_59
5. Bedick, C., Tulgestke, A., Nie, K., & Ferguson, D. H. (2022, January 3). Comparisons of RDE inlet dynamics in a linear testing platform and models of varying complexity. *AIAA SCITECH 2022 Forum*. AIAA SCITECH 2022 Forum, San Diego, CA & Virtual. https://doi.org/10.2514/6.2022-1455
6. Bierig, R., Brown, S., Galván, E., & Timoney, J. (2021). *Essentials of Software Testing* (1st ed.). Cambridge University Press. https://doi.org/10.1017/9781108974073
7. Bishop, N., Harris, R., & Marsh, D. (2022, January 3). Repurposing Spent Upper Stages into Platforms for In-Space Testing. *AIAA SCITECH 2022 Forum*. AIAA SCITECH 2022 Forum, San Diego, CA & Virtual. https://doi.org/10.2514/6.2022-2466
8. Braun, D., Schwaiger, F., Holzapfel, F., Diepolder, J., & Ben-Asher, J. Z. (2022, January 3). Continuous Integration of Optimal Control Based Flight Control Law Clearance. *AIAA SCITECH 2022 Forum*. AIAA SCITECH 2022 Forum, San Diego, CA & Virtual. https://doi.org/10.2514/6.2022-1895
9. Ciptaningtyas, H. T., Husni, M., Rosyadi, F. D., & Qudus, R. (2018). Web based Application Quality from End User Perspective: Case Study - Assignment Letter LPPM ITS: *Proceedings of the 7th Engineering International Conference on Education, Concept and Application on Green Technology*, 365–373. https://doi.org/10.5220/0009011303650373
10. Cunningham, M., Nigam, N., Mukhopadhaya, J., Alonso, J. J., & Ayyalasomayajula, S. (2023, January 23). Multi-Fidelity Probabilistic Aerodynamic Database Generation with the ProForMA Tool. *AIAA SCITECH 2023 Forum*. AIAA SCITECH 2023 Forum, National Harbor, MD & Online. https://doi.org/10.2514/6.2023-0653

11. Fatima, S., Mansoor, B., Ovais, L., Sadruddin, S. A., & Hashmi, S. A. (2022). Automated Testing with Machine Learning Frameworks: A Critical Analysis. *The 7th International Electrical Engineering Conference*, 12. https://doi.org/10.3390/engproc2022020012
12. García, B., Munoz-Organero, M., Alario-Hoyos, C., & Kloos, C. D. (2021). Automated driver management for Selenium WebDriver. *Empirical Software Engineering*, *26*(5), 107. https://doi.org/10.1007/s10664-021-09975-3
13. Halani, K. R., Kavita, & Saxena, R. (2021). Critical Analysis of Manual Versus Automation Testing. *2021 International Conference on Computational Performance Evaluation (ComPE)*, 132–135. https://doi.org/10.1109/ComPE53109.2021.9752388
14. Jain, J. (2022). Tools, Frameworks, and Libraries. In J. Jain, *Learn API Testing* (pp. 41–73). Apress. https://doi.org/10.1007/978-1-4842-8142-0_4
15. Ji, P., Feng, Y., Liu, J., Zhao, Z., & Xu, B. (2021). Automated Testing for Machine Translation via Constituency Invariance. *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 468–479. https://doi.org/10.1109/ASE51524.2021.9678715
16. Jones, H. W. (2022, October 24). Fault Tolerance Should No Longer Be Used. *ASCEND 2022*. ASCEND 2022, Las Vegas, Nevada & Online. https://doi.org/10.2514/6.2022-4368
17. Kedward, L., & Allen, C. B. (2021, January 11). Optimisation of a Finite-Volume Test-bench Code for Highly Parallel Architectures. *AIAA Scitech 2021 Forum*. AIAA Scitech 2021 Forum, VIRTUAL EVENT. https://doi.org/10.2514/6.2021-0143
18. Kim, H., & Swan, C. A. (2023, January 23). Development of a Web-based Test Procedure Authoring and Execution Environment for Space Systems. *AIAA SCITECH 2023 Forum*. AIAA SCITECH 2023 Forum, National Harbor, MD & Online. https://doi.org/10.2514/6.2023-2585
19. Manukonda, K. R. R. (2023). A COMPREHENSIVE EVALUATION OF SELENIUM WEBDRIVER FOR CROSS-BROWSER TEST AUTOMATION: PERFORMANCE, RELIABILITY, AND USABILITY. *Journal of Artificial Intelligence, Machine Learning and Data Science*, *1*(3), 580–584. https://doi.org/10.51219/JAIMLD/kodanda-rami-reddy/152
20. Mitchell, T., Hartman, M., Johnson, D., Allamraju, R., Jacob, J. D., & Epperson, K. (2020, June 15). Testing and Evaluation of UTM Systems in a BVLOS Environment. *AIAA AVIATION 2020 FORUM*. AIAA AVIATION 2020 FORUM, VIRTUAL EVENT. https://doi.org/10.2514/6.2020-2888
21. Mr. Sundaramohan S. (2023). Enhancing Performance of Software Testing Automation using Selenium Web Grid. *International Research Journal on Advanced Engineering Hub (IRJAEH)*, *1*(01), 1–5. https://doi.org/10.47392/IRJAEH.2024.001
22. Murakami, D. D., Shaw-Lecerf, M., Lash, E. L., Lyons, K., & Roozeboom, N. (2023, January 23). Implementation of the Lifetime Method in Unsteady Pressure-Sensitive Paint Measurements. *AIAA SCITECH 2023 Forum*. AIAA SCITECH 2023 Forum, National Harbor, MD & Online. https://doi.org/10.2514/6.2023-0635
23. Prof. B K Srinivas4, R. S. N. (2023). Dynamic Resource Allocation in Cloud Environments. *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, *11*(6 Jun 2023), 45–98.
24. Ricca, F., Marchetto, A., & Stocco, A. (2021). AI-based Test Automation: A Grey Literature Analysis. *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 263–270. https://doi.org/10.1109/ICSTW52544.2021.00051

25. Schmiechen, K., Schwaiger, F., Wechner, M. A., & Holzapfel, F. (2022, June 27). Combining ALM and MBD Tools for Continuous Requirement Validation Tests with Multi-Dimensional Test Parameters. *AIAA AVIATION 2022 Forum*. AIAA AVIATION 2022 Forum, Chicago, IL & Virtual. https://doi.org/10.2514/6.2022-3233
26. Schwaiger, F., Schmiechen, K., & Holzapfel, F. (2023, January 23). Tico – a Toolbox to Author and Execute Large Parametrizable Test Suites in MATLAB. *AIAA SCITECH 2023 Forum*. AIAA SCITECH 2023 Forum, National Harbor, MD & Online. https://doi.org/10.2514/6.2023-1123
27. Silva, J. R., Silva, J. M., & Vaquero, T. S. (2020). Formal Knowledge Engineering for Planning: Pre and Post-Design Analysis. In M. Vallati & D. Kitchin (Eds.), *Knowledge Engineering Tools and Techniques for AI Planning* (pp. 47–65). Springer International Publishing. https://doi.org/10.1007/978-3-030-38561-3_3
28. Singh Dhaliwal, A. (2022). Managing Artifacts and Binaries in Continuous Integration / Continuous Deployment (CI/CD) Pipelines. *International Journal of Science and Research (IJSR)*, *11*(6), 2003–2005. https://doi.org/10.21275/SR24506190429
29. Thorpe, A. (2023). *selenium: Low-Level Browser Automation Interface* (p. 0.1.4) [Dataset]. https://doi.org/10.32614/CRAN.package.selenium
30. Uittenhove, K., Jeanneret, S., & Vergauwe, E. (2022). *From Lab-Testing to Web-Testing in Cognitive Research: Who You Test is More Important Than How You Test*. PsyArXiv. https://doi.org/10.31234/osf.io/uy4kb
31. Van Merode, H. (2023a). *Continuous Integration (CI) and Continuous Delivery (CD): A Practical Guide to Designing and Developing Pipelines*. Apress. https://doi.org/10.1007/978-1-4842-9228-0
32. Van Merode, H. (2023b). Pipeline Development. In H. Van Merode, *Continuous Integration (CI) and Continuous Delivery (CD)* (pp. 207–284). Apress. https://doi.org/10.1007/978-1-4842-9228-0_5
33. Van Merode, H. (2023c). Use case. Continuous Integration (CI) and Continuous Delivery (CD),. In H. Van Merode, *Continuous Integration (CI) and Continuous Delivery (CD)* (pp. 359–406). Apress. https://doi.org/10.1007/978-1-4842-9228-0_9
34. Wechner, M. A., Marb, M. M., & Holzapfel, F. (2023, June 12). A Strategy for Efficient and Automated Validation and Verification of Maneuverability Requirements. *AIAA AVIATION 2023 Forum*. AIAA AVIATION 2023 Forum, San Diego, CA and Online. https://doi.org/10.2514/6.2023-3595
35. White, J. T. (2023, June 12). Modular Open Architecture UAS Test Platform System of Systems. *AIAA AVIATION 2023 Forum*. AIAA AVIATION 2023 Forum, San Diego, CA and Online. https://doi.org/10.2514/6.2023-3899
36. Yadav, V., Botchway, R. K., Senkerik, R., & Kominkova Oplatkova, Z. (2021). Robotic Automation of Software Testing From a Machine Learning Viewpoint. *MENDEL*, *27*(2), 68–73. https://doi.org/10.13164/mendel.2021.2.068
37. Yin, Y., & Jiang, B. (2023). *Embedded Software System Testing: Automatic Testing Solution Based on Formal Method* (1st ed.). CRC Press. https://doi.org/10.1201/9781003390923