White Paper

Cognizant Testing Services Automation Framework – Data Handling

Contents

Introduction

In today's world of Business, "Change" is the constant factor that has been driven by heavy and healthy competition. As more countries are opening their markets, global competition is growing at a rapid phase. Anybody can start business from any part of the world and base them in any number of locations around the globe and serve their clients who are present at millions of miles away. This concept of "Global Village" or "Globalization" has emerged out as a striking solution to the needs of many to make life easier on the earth. And thus the new approaches of doing business are constantly created, developed and put in place to serve the clients in a better way.

As the business grows, the information management becomes more and more complex, leading to higher degree of difficulty in managing it. Information Technology is the key enabler to any business. Businesses are aware of the benefits derived by effective usage of Information Technology and they believe that IT would help them to become more competitive, cost effective and productive in their business.

This competitive and demanding scenario in business drives IT. Information Technology is constantly evolving and redefining the technologies and processes to provide better solutions to meet the increasingly complicated and complex business requirements. As the business activities become more mission-critical, the need for verification and validation methods that support business objectivities has dramatically increased. It is necessary to ensure that these systems are reliable, built according to the specifications, and have the ability to support business processes. Many Internal and external factors are driving IT organizations to ensure a high level of quality and reliability.

Quality of a system can be assured only through an extensive testing process. This conscious understanding among the business and IT community has created a great demand for software testing services. Testing starts with innovations, creativity and redefining with new dimensions. Many new testing methodologies, processes and quality road maps are created to exceed the expectations of the demands. Automation is one such dimension that helps us to migrate up the value-add chain with our customers. This paper is an attempt to identify the potential areas of automation and define a framework for efficient test automation.

Types of Automation

- Functional Automation
- Performance Automation
- Data Handling Automation

Functional Automation – An Overview

Automated testing is recognized as a cost-efficient way to increase application reliability, while reducing the time and cost of software quality programs. In the past, most software tests were performed using manual methods. Owing to the size and complexity of today's advanced software applications, manual testing is no longer a viable option for most testing situations. A lot of effort goes into developing and maintaining test automation, and even once it's built you may or may not recover your investments. It's very important to perform a good cost/benefit analysis on how much of manual testing can be automated. The success stories of automation have mostly been on focused areas of the application where it makes sense to automate, rather than complete automation efforts. Also, skilled people are involved in these efforts and they are allowed the time to do it right.

Test automation can add a lot of complexity and cost to a test team's effort, but it can provide a lot of valuable assistance if it is done by the right people, with the right environment and in right situations. This document will detail the efficient way of approaching automation to save time, money and less frustration in your efforts to implement test automation back on the job.

Data handling Automation – An overview

To make it in simple words, "Automated Testing" is the process of automating manual testing process currently in use. There are lots of functional testing tools available in market, each proclaiming them as the best tool to use. If the word "Automated Testing" were referred in a forum, the general mindset would be to record and playback the script for GUI testing. The usage of automated testing is not only in GUI testing, Web based testing, but it has a huge potential in data validation testing also.

By this, what do we mean by data validation testing? Where is this kind of validation done? Is it a run-time data validation? Which platform does the data belong to? Which tools are good in data validation testing? These might be some of the possible questions that rise in your mind.

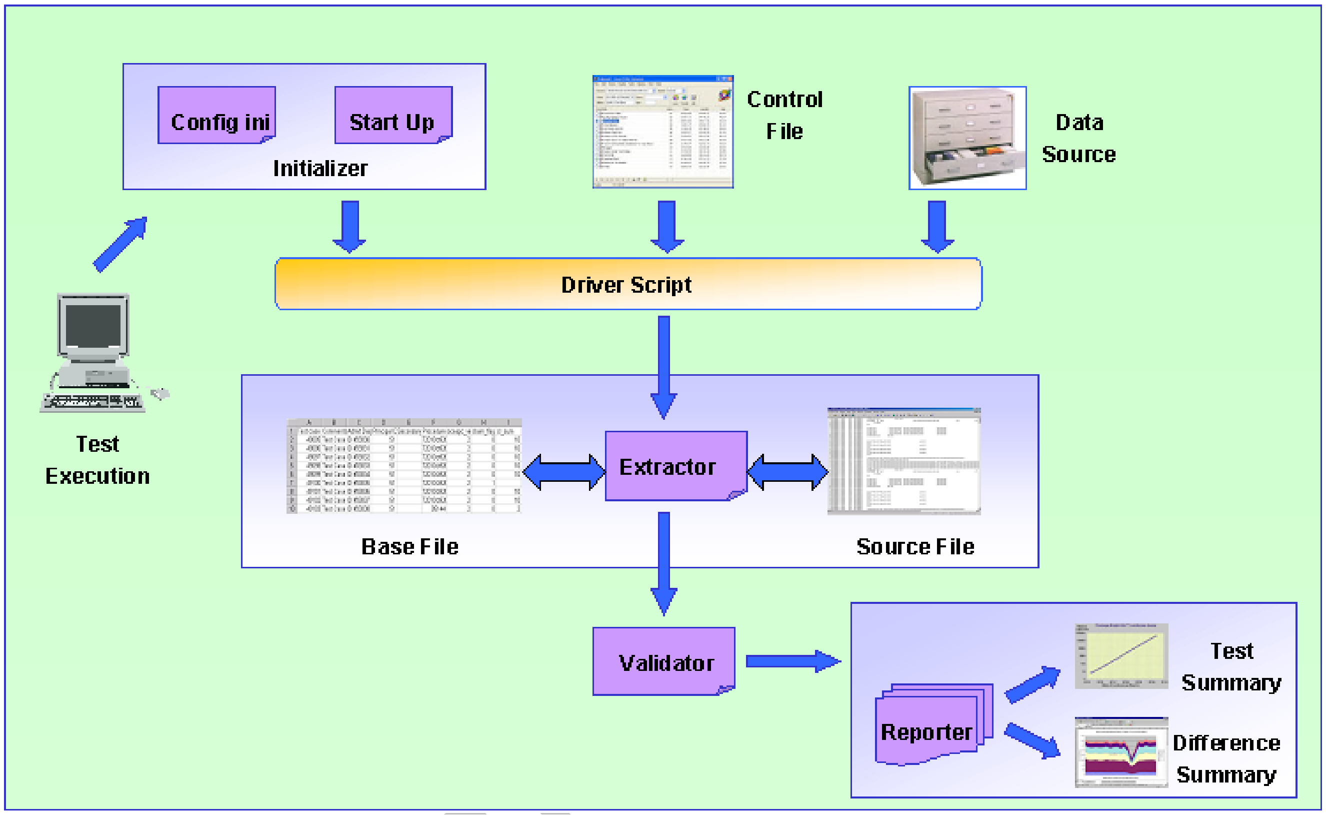
Data validation is the process of extracting data from two different resources, one being the base and the other as source, evaluating them for correctness and reporting the differences. The resources can be generally obtained from any platform, Windows, AS400 or Mainframe in the form of flat files or Excel files. A file that has no structured interrelationship between its data records is known as a flat file. A flat file's data can be identified only with the help of a data source that contains the layout information of the flat file. The data should be generated in batch mode and not in run-time mode.

Data validation testing is more commonly done in domains like Healthcare Management Systems where the content is the king. The patient health data is captured, classified, and manipulated in order to create error free reimbursement claims. The health data classification is regulated and standardized periodically by government bodies. So the data needs to be validated with the base data to ensure that the product meets the standards. The system can be in any platform, windows, AS400 or mainframe; the source file should be generated in batch mode and compared with the base file.

Automation Framework

As the name describes, this framework concentrates on modularizing the functionalities to the unit level and mapping them one after the other as per the script design flow. Any data handling process can be split into data initialization, data extraction, data validation and verification and finally data reporting.

Fig: 1.1



Key Components of the framework

The **Initializer** consists of a control file that is used to vary the input parameters for the driver script and data source that contains the layout information of the flat files or general information to extract data from the files. Pre-execution information is provided to the initializer to set controls and vary parameters of the driver script prior to test execution. Initializer facilitates methods to verify the folder structure, delete previous execution reports, set new base and source files appropriately and start the driver script.

The **Driver script** is the engine of the framework. It has to be triggered first for the execution. It acts as an engine and pulls the rest of the components based on the design. It loads all modules that need to be triggered for the smooth execution. It gathers information from the control file and data source, sets control over other modules till the entire execution is completed.

The **Extractor** is the key component of this framework. Its function is to open the base and source files, and extract the required data for validation. It is the deciding component for the complexity of the script. For example, if both the files are in excel format, then extracting data from the excel files would be quite easy and would have a lighter effect on the complexity of the script. Instead, if both the files are flat files and of different formats, then it would increase the complexity of the script. The extracted data is the input parameter for the validator component.

The **Validator** compares the actual and the expected data and sends the results to the reporter. It is the smallest component of the framework with a comparison logic included. And the result as either pass or fail is sent as input parameter to the reporter component.

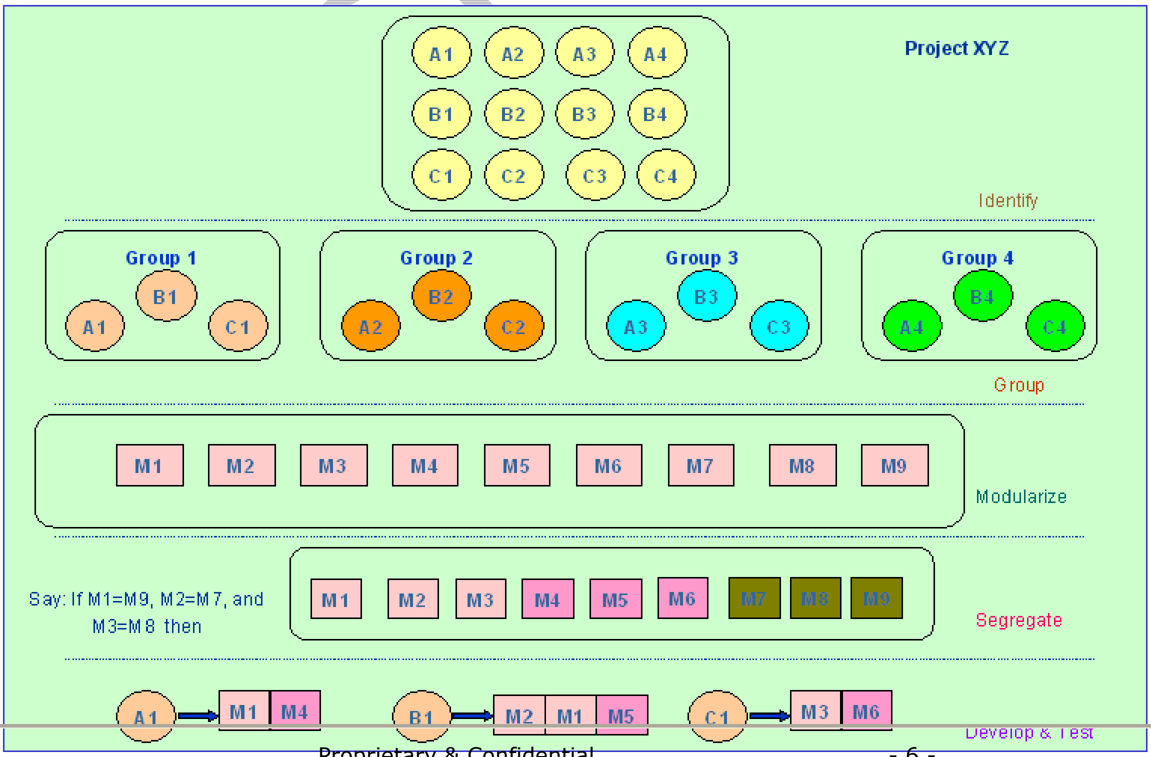
The **Reporter** presents the output to the end user. The reports include test summary and difference summary. It has options to view the number of records compared, number of records passed and number of records failed in the validation. The difference summary provides the file details, actual data and expected data for the deviated fields. It can also be customized according to the style of reports required.

How to make it work in your project

Till now, we discussed about the Modular Centric Framework, its various components and the core functionalities. It is also important to know how to make it work in our data handling projects. Automation is not an isolated island and requires a solid infrastructure, and a thoughtful software testing life cycle that can deliver the real value from the automation effort. It should be treated as a development project and should pass through the phases of requirements gathering, analysis, design, code, test and finally into production.

Taking into consideration the strong groundwork required, a five phase approach (IGMSD) is designed and suggested as the steps to be followed to implement the modular centric framework

Fig: 1.2 Conceptual view of IGMSD - The five phases



effectively and successfully. The five phases include Identify, Group, Modularize, Segregate and Develop & Test. Each of them has its own entry and exit criteria to make them comprehensive.

Here we find a few constraints in providing a familiar example like Login function of an application to explain the phases. It is a well known fact that familiar functionalities, as illustrations, would be easy to visualize and understand the concept. As there is no such familiar example in data validation testing, we have tried out a conceptual pattern to make you comfortable.

Phase I: Identify

This is an important phase and very project specific. Identifying the right candidates for automation is more critical than developing the scripts. In big picture, data validation testing is a better candidate for automation, but it is obligatory to spot the proper nominees under data validation testing before automation. The process of identifying the test cases for automation differs depending on projects. Yet, a standard manner should be followed to avoid defects entering into the automated test life cycle at the earlier phase itself. Few steps to identify the test cases for automation are listed below:

- Form an automation team that has good programming capabilities and tools knowledge
- Arrange for a brainstorming session which includes product test leads and automation team to discuss on various test cases considered for automation
- Discuss the complexity of the test cases and possibilities to automate them
- Select test cases that have proper expected results and avoid test cases that need more manual interpretations
- Document the discussions and finally come out with the total test cases need to be automated

Let us discuss the phases with the example in Fig 1.2. Say, you are working in a project XYZ that contains three products A, B and C. You need to automate the data validation testing part of your project and at the end of the brainstorming session your team has come out with four test cases in each product to automate with a total candidates of twelve for automation A1, A2 …C3 and C4.

Phase II: Group

The test cases across products should be grouped based on the similarities. Here similarities stand for the common type of business functionalities in data validation. It is done to create major clusters over the products so that any future functionality changes in data validation require lesser effort to update the scripts. Each group should be supported by a group logic document that explains the scope, the common functionalities and the validation logics for the group.

In the above example, Let us assume that test cases A1, B1 and C1 have similar business functions and hence they are grouped as Group 1. Likewise, other groups are formed. The rest of the phases will deal about a single group. The advantage of grouping is that the automation team can concurrently work on different groups and thereby stimulate rapid script development. It is not mandatory for a test case to be in a group, it can be separate if it has distinct business function.

Phase III: Modularize

This phase requires good functional knowledge and basically deals with modularizing the functions to the unit level. Each and every test case in a group should be analyzed, drilled down and dismantled into modules. These modules should be properly documented. The following things should be noted for all modules.

- Name each modules as per the automation standard
- Describe its functionality in two lines so that it will help others to understand it later
- Note the input and output parameters of the functions
- Continue this process for all the test cases in the group

In the above example, the group 1 is modularized into functions M1, M2 …and M9.

Phase IV: Segregate

Now that all modules of a group are in hand, the next action is to segregate and bundle them into the components of the framework as Initializer, Driver script, Extractor and Reporter. The duplicate modules across test cases should be ignored, this increases the reusability of the modules and reduces the effort spend to develop and test the duplicate functions. Based on the functionality of the modules, they are packed into any one of the framework components.

In the example, the modules are analyzed for duplication and the duplicate modules M7, M8 and M9 are ignored for development. Generally, the extractor and validator components increase in size, as these two functions would be unique for each test case. M4, M5 and M6 are distinct extraction functions of the test cases and M1, M2 and M3 belongs to different components of the framework.

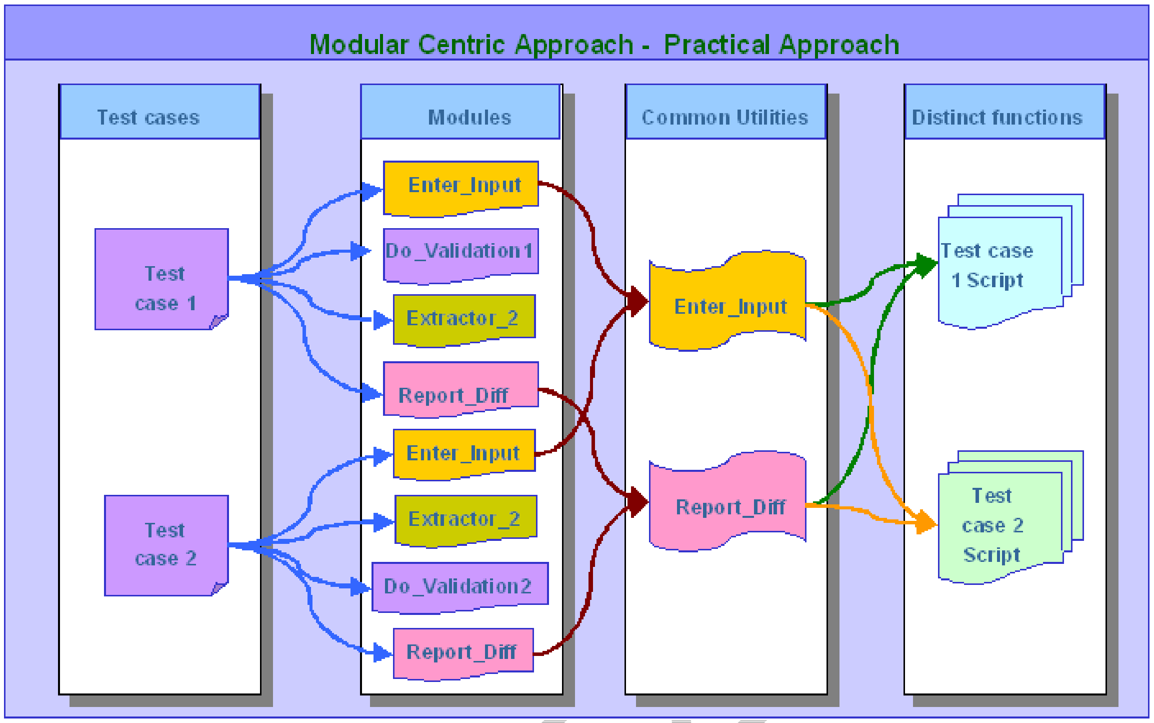
Phase V: Develop and Test

The final phase of the approach is to develop the modules, integrate them as components, call the components as per the framework design and finally validate the automated script for both pass and fail conditions. Create a checklist to ensure that all data are validated as per the requirements. The checklist can also be submitted as a review report to the project management team.

The automated script should be tested with sample base and source files. Each data in a source can have different validations, and multiple sample files covering as many scenarios as preferred. Acceptance testing should be given sufficient time in order to reduce the efforts for maintenance and rework after the test script execution. In this example, the modules are developed and integrated as per the test requirements.

The following is the diagrammatic view of the Modular Centric Approach. It shows how test cases are transformed into the framework as automated scripts by following the IGMSD phases.

The further enhancements in the test cases make minimal effect on the common components and only the distinct components require script development, furthermore, the test cases are quickly automated without requirements leakage.



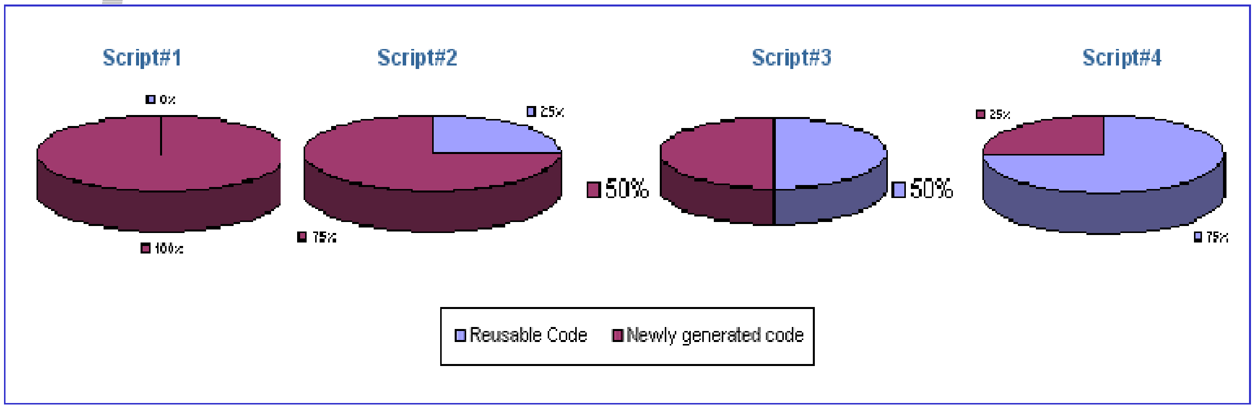
The framework provides the flexibility to use the common components across groups.

Assets and Benefits of the Framework

Reusability

The concept of 'Modularization" is adopted in data validation automation to improve the reusability of the functions. In conventional automation projects, the same functions are knowingly or unknowingly repeated across scripts. The effort spend in developing such repeated codes is known as Redundant effort.

The above chart illustrates the benefit of using modular centric framework for data validation automation. The first script of a group requires 100% effort in generating new functions and in the consecutive scripts; there is a steady rise in the reusable functions. In the later scripts, the reusable functions play major role, but



still practically looking there might be few distinct functions that can only be developed newly.

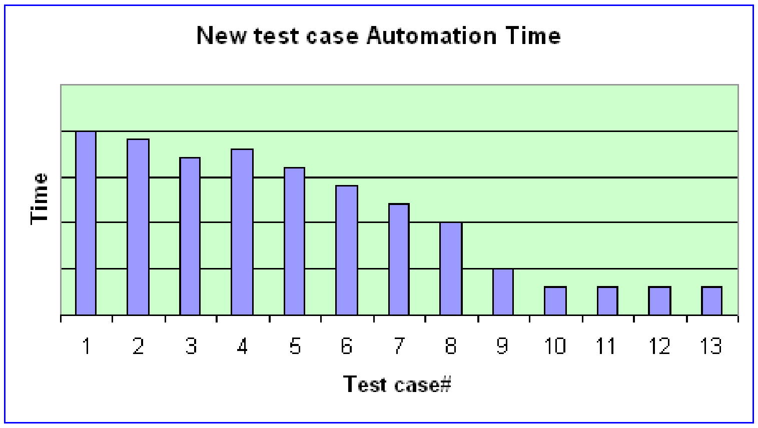
Easy Maintainability

Maintainability is the key factor for the success of any automation framework. In data validation automation, maintaining the scripts is not only updating the existing logic but also involves adding new logic and functions for validating new fields. As the conditional logic grows, the complexity of the script increases and the conventional scripts fail at their endurance limit. As the scripts are a collection of modules, pining them to the required module is easy and the changes are incorporated without impacting other functions. Addition of new logic is as easy as plug and play and it can be developed and put in the right place. As we have a well organized approach (IGMSD) to implement the framework, it is simple for any member of the project to trace the updates required in the script.

Rapid Script Development

The theme of frameworks is originated in order to speed up the automation scripting process without impacting the quality. Some of the benefits of reusability we have discussed earlier, and one of it is the script development time that is greatly reduced.

Initially there would be additional efforts spent in implementing the framework and creating the modules. But this forms the base and the benefits of it are realized as the script development continues. At one point of time, building the new components gets saturated; any further script development is just calling the required functions as per the framework. The above chart substantiates the benefit of the framework on rapid script development.



Test Coverage

Test coverage is one of the vital benefits of the automation framework. It is impossible to perform manual data validation testing, as the data is huge and turn out to be a monotonous task to complete it. The dependency on automation is inevitable. Conventional automation scripts cannot handle the requirement of 100% test coverage and they are vulnerable when the complexity increases. Our automation framework is designed to manage this and the scripts are extensible on increasing complexity and thereby provide 100% test coverage. The checklist of functions can be used as a trace ability document to ensure that all expected validations are incorporated on the data and performed in the projected manner.

Tool Independent

The way we carry out the automation process is more important than the tool we are using to generate scripts. The module centric framework is strongly dependent on the automation process perspective rather than the tool perspective. The framework is tool independent, and it does not rely on any tool-specific functions. Any tool that has file handling functions and flexible programming capabilities can be used to proceed with the framework. We have experimented this framework with Mercury WinRunner and Rational Robot and achieved excellent results.

Robust Exception-handling

The data validation automation needs a lot of exception handling capability. For example, a file might not be present in the directory when the file open function tries to open the file and fails. This exception has to be handled properly. All the loop functions in a script should be checked for infinite looping. All these exceptions can be handled in an exclusive manner in this framework. The exceptions are written once and are reused in all the scripts. Each function is developed with possible exceptions incorporated in it. A safety layer is created over the functions to prevent catastrophic failures of the scripts.

Team Involvement

The important benefit of the IGMSD approach is the great deal of the entire team involved in automation process. In traditional method, the automation team receives the lit of test cases from the product tester for automation. And then a feasibility study is conducted for each test case. Here proper communication between the product team and automation team is lacking and thereby some situations lead to conflicts. To avoid such unwanted conflicts, the IGMSD phases are designed to involve the entire team in the automation process. Brain storming sessions and formal review meeting can be used as a forum to raise issues and solve them through discussions. Each team can understand the limitations of other sides and sort out optimized solutions for all concerns.

Readability

As the framework is more segmented and the functions are clearly documented, it is quite easy to understand the logic flow of the script. As per the automation standards, the functions are in user readable forms and all the navigations are clearly defined as comments. High readability assists in smooth knowledge transition to other automation resources.

Limitations in the framework

At the end let us discuss the limitations of this framework.

Entail Core Automation Team

The data validation automation is not about record and playback. Its usage is tool specific and built-in functions are very minimal, hence greater programming knowledge and skills are required. The framework constitutes of segmented structures, which has to be built on user-defined logical functions. A data validation project without a core automation team may not be successful for developing the framework components and may end up as a failure. The core automation team is indispensable, and the team should possess good analytical skills, logical thinking abilities and excellent programming expertise to create, integrate and test the logical functions.

Going Forward - Conclusion

The major benefit of this framework is to identify the capability of data handling automation using functional automation tools that provide the reusability, speed, data accuracy etc. After reading this white paper you should ask, is automation framework for you? If you are like many others out there to do record and playback or have the assumption that data handling is handled by functional tools then the answer will be an emphatic no. The speed of script development attracts everyone. The maintainability makes it popular among the developers. The concept of this framework is new and needs better visibility and selling to make it reusable. This white paper raises some additional questions, concerns, and thoughts. The dialogue is important as the automation world decides the future of Data handling automation. Now is the opportunity to take this dialogue beyond the Cognizant community to the Testing World. Future generations will benefit from today's activism.