ENHANCING SOFTWARE QUALITY

INDIUM

Design Guide – Automation Framework for Data Intensive Applications

Prepared by:

Sandeep M
&
Nivedita D

Table of Contents

1. Abstract

Testing of all the software applications is mandatory to ensure that they are bug free before they go live on production. A growing majority of business applications today come with intense data and testing them in an effective manner is not an easy task! Data Intensive software applications come up with many challenges, such as difficulties to create, manipulate, store and retrieve large amounts of data, and to analyze and apply complex business rules to such data. For Banking applications that are data intensive, additional challenges would be to deal with data confidentiality, data accuracy, data security and data consistency.

Many data-intensive software systems use data management systems for data storage, such as Relational Database Management Systems (RDBMS). However, this method does not always prove to be beneficial, considering the drawbacks such as reduced speed, performance, turnaround time; increased costs, management complexity and frequent upgrade/replacement cycles.

This paper is aimed at demonstrating how and what could be followed to overcome these challenges!

Every test environment will require a framework that provides support for automated software testing. To handle applications that are data intensive, the need for a robust, scalable, reusable, reliable, maintainable and modular “Framework and Data Sheet” are mandatory. Framework suite settings file configured using macros should be designed in such a way that it can deal with huge data with promising performance.

Based on a new concept, we will also be discussing on an exceptional approach on maintaining the data that is used for carrying out the execution on various inter-dependent days.

2. Introduction

Healthcare, banking, media and education are some domains that use huge data. With intensive Data, it is also required to ensure that the factors like security, scalability, reliability and re-usability of those data are managed.

From the point of view of 'Testing Data-Intensive Applications', one of the most relevant aspects is to ensure accuracy of the business logic in terms of data consistency.

To test your code, you need data that at least passably resembles data that the application would work with in reality. If the amount of data you're managing and the dynamic changes in applications and workflows keep you in constant maintenance mode, a new and efficient design for the framework would be required.

3. Key Terminologies

3.1 Data Intensity

Data Intensity has drawn huge attention from researchers in information sciences, policy and decision makers in governments and enterprises. There are highly useful values hidden in the huge volume of data. A new scientific model is born as data-intensive scientific discovery (DISD), referred to as Data Intensity problems. Many numbers of fields, ranging from economic and business activities to public administration, from national security, Banking, Health Care to scientific researches in many areas, involve with Intensive Data problems.

Intensive Data is valuable to produce productivity in businesses and evolutionary breakthroughs in scientific disciplines. There is no doubt that the future competitions in business productivity and technologies will surely meet the Data Intensive explorations. It also comes up with many challenges, such as difficulties in data capture, storage, analysis and visualization.

3.2 Database

Database is an organized collection of data. The data are typically organized to model relevant aspects of reality in a way that supports processes requiring this information.

RDBMSs are used to store information in a structured manner, and to define several types of constraints on the data, to maintain basic consistency. The RDBMSs are well tested, software products that one can trust to reliably store data and keep it consistent within the defined constraints.

However, when it comes to handling Data Intensive Applications, passing the responsibility of consistency enforcement to the RDBMS is not convenient, or simply not possible considering the limitations. In such cases, the alternative is to have that responsibility at the business logic level of the system.

4. Advantages and Disadvantages of using Database

Many software applications use one or more databases to store huge amounts of data. They also provide various interfaces to inspect and modify the data. Some application interfaces may be accessed from a web interface; others may be used from a desktop application. Different users have different access rights, and therefore are presented with different interfaces to the data. The database normally constrains certain relations on the data that cannot be violated; provided the database management system is implemented properly.

Apart from the built-in constraints, there are other rules that are imposed by the application. These constraints involve both data format or relationships, and non-trivial calculations. In general, these constraints are verified by the application, that is, the system validates that input data is of the correct format, or if a process is followed before the database is queried. If the information is not verified, it will result in the data storage in the database which does not fulfil the constraint. There are other reasons for the requirement of constraints which are unenforceable by the database. For example, it is required when the constraint is time-dependent, or has performance and efficiency demands that cannot be guaranteed by the database itself. Commonly, many constraints are too business-specific, and therefore there is no need to connect them in the database, in particular not if the database is to be shared by other systems. For these reasons, it is the business logic of the application which needs to cope with these business-specific constraints, also known as business rules.

When testing a data-intensive application, it is crucial to provide enough assurance to ensure that the system does not reach a situation where a business rule is violated and data is left in an inconsistent state. This should be tested independently whether such is a consequence of one or several invocations of interface functions with unexpected input or scenarios. Previous attempts have faced the challenge of business rules definition and modelling, either by proposing new development methodologies, or by formulating modifications to existing ones.

A test fails if the application crashes during consecutive calls, when the result of one of the interface functions is unexpected or when the database is left in a state that violates the business rules.

5. Advantages of using Excel in managing and designing Framework for Data Intensive Applications

Test data needs to be prepared in advance for testing many data intensive applications. Many large data intensive applications depend on the data that is stored. This storage might be in any format such as SQL, ORACLE, RDBMS, EXCEL, CSV, etc. These applications might connect to the database(s) and read or write data from or to the database using queries.

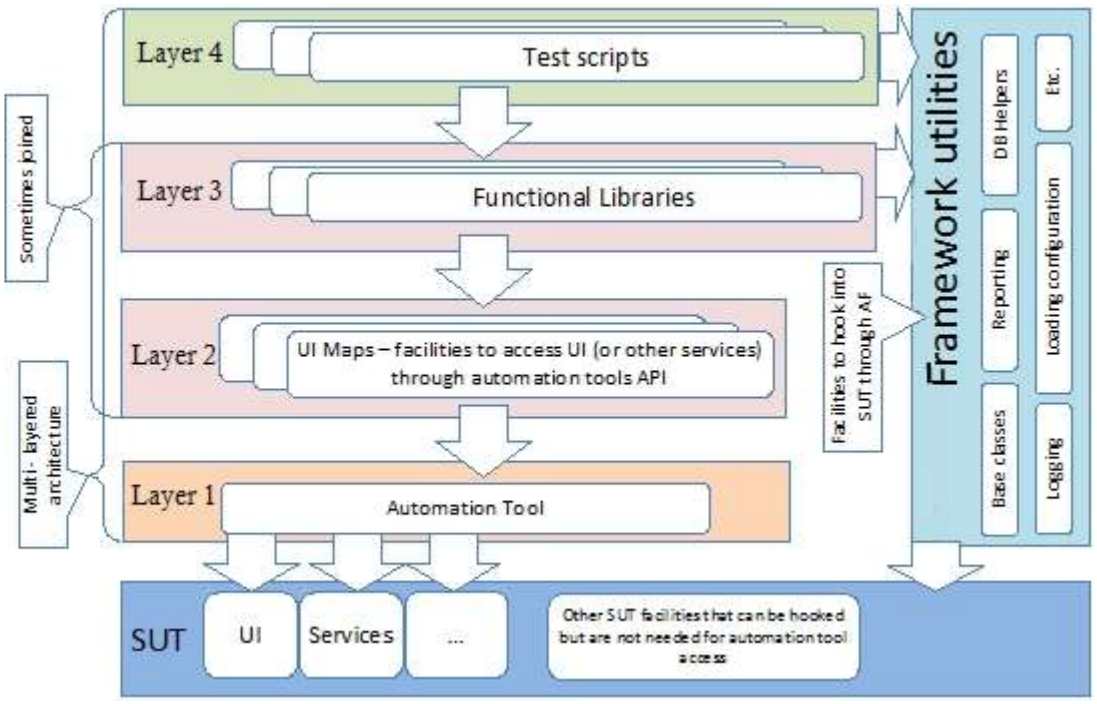
In many cases, real data that the application uses is unavailable during the testing stage. This can be due to data privacy regulations, in which the tester might not have access to real data. Alternatively, testing may be carried out during an early development phase of the application when the real data doesn't exist or is not structured in a way that the application assumes. Even when the real data is available, it might not have the required characteristics for thorough testing.

To overcome the limitations of using a RDBMS while handling huge data, an alternate approach needs to be followed because it would not be feasible to integrate the Automation Framework for Data Intensive Applications with RDBMS, in situations where the test data is large, considering the time consumed for data retrieval using RDBMS.

Using Excel to store large amounts of test data and integrating it with the framework would prove as an efficient method as it is a lot quicker in retrieving the data when compared to data retrieval using RDBMS. The following are the main benefits of using Excel approach:

- Re Usability of Components
- Cleaner Folder Structure
- Support for various Data Sources
- Readily available, hence faster script development

6. Testing Framework


Sometimes joined
Multi - layered architecture
Layer 4
Test scripts
Layer 3
Functional Libraries
Layer 2
UI Maps – facilities to access UI (or other services) through automation tools API
Layer 1
Automation Tool
Facilities to hook into SUT through AF
Framework utilities
DB Helpers
Etc.
Reporting
Loading configuration
Base classes
Logging
SUT
UI
Services
...
Other SUT facilities that can be hooked but are not needed for automation tool access

Test Automation Framework is a basement to undertake Automated Testing by using automated tools. It is a set of concepts, assumptions and practices which, all-together, provides support for the Automated Software Testing. Framework is built to carry out the automation testing successfully.

Framework is of five types:

- Test Script Modularity Framework
- Test Library Architecture Framework
- Keyword Driven Testing Framework
- Data Driven Testing Framework
- Hybrid Test Automation Framework

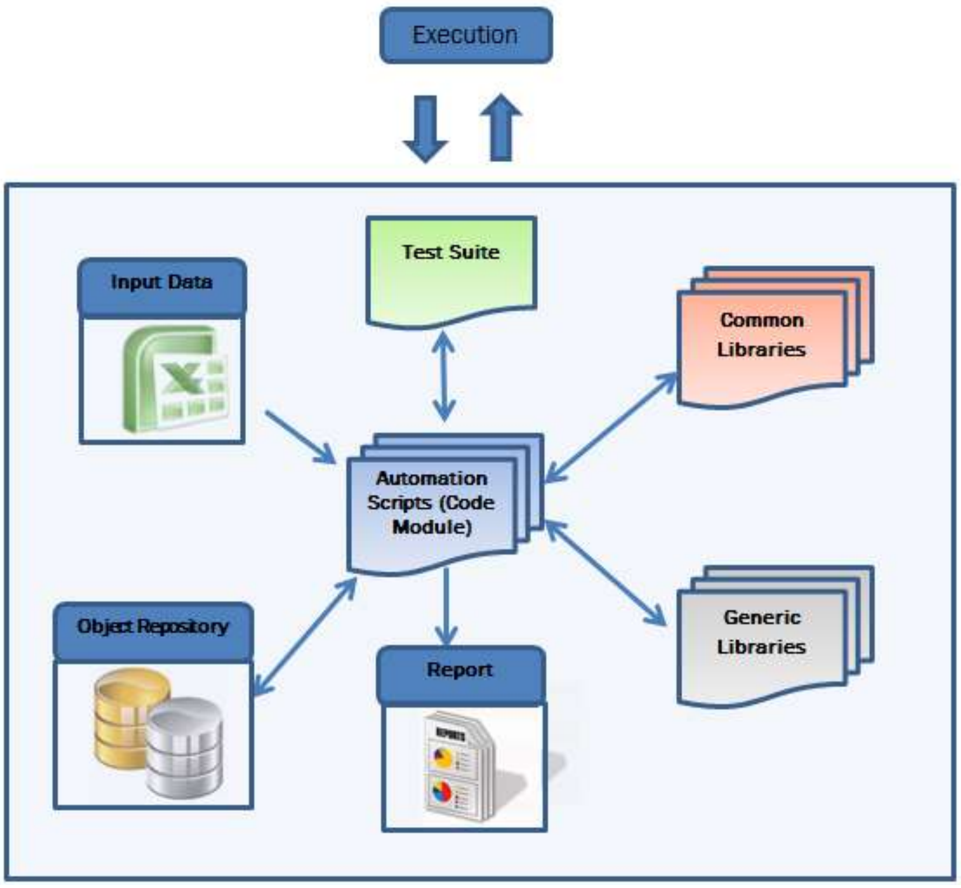
6.1 Guidelines for Framework

Frameworks will be normally designed using the traditional way (which can be seen in most of the companies). Why to be in the old traditional way. Day by day applications are changing; new technologies are coming up and new domains are rising. With these, the data for the applications also are increasing. But still we are using the same traditional framework. Here we will suggest you some useful approaches as an alternative for the Traditional Framework.

6.2 Traditional Framework

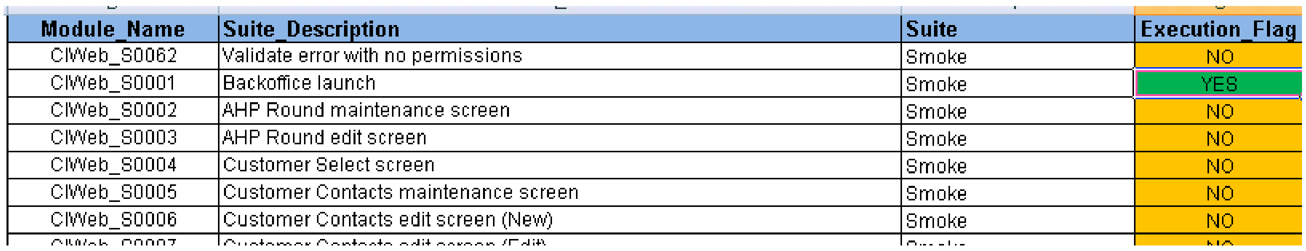
As we know, the framework will have a particular folder structure which contains the following:

1. Suite Settings
2. Test Data
3. Scripts
4. Libraries
5. Object Repository
6. Reports



6.2.1 Simplifying Suite Settings

The traditional Framework will have its Execution Flags (Yes/No), Test Priority (Low/Medium/High) to be selected for each and every test case. When the Data is less, it is obvious that all these can be selected.



Module_Name	Suite_Description	Suite	Execution_Flag
CIWeb_S0062	Validate error with no permissions	Smoke	NO
CIWeb_S0001	Backoffice launch	Smoke	YES
CIWeb_S0002	AHP Round maintenance screen	Smoke	NO
CIWeb_S0003	AHP Round edit screen	Smoke	NO
CIWeb_S0004	Customer Select screen	Smoke	NO
CIWeb_S0005	Customer Contacts maintenance screen	Smoke	NO
CIWeb_S0006	Customer Contacts edit screen (New)	Smoke	NO

When talking about Intense Data, how would it be possible to make that many selections? If the data is more and what you want to execute is less, do we set the flags as **Yes** or **No** by searching and changing the status for all?

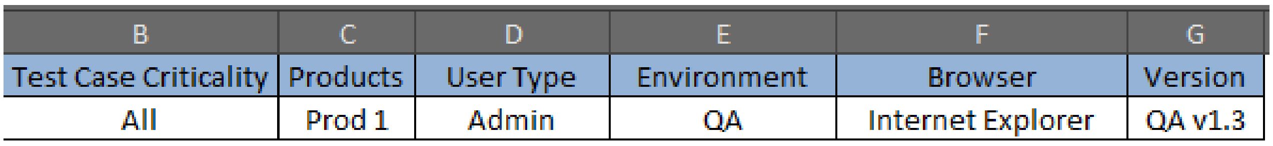
In this case, if there are thousands of cases and some hundreds are to be executed, it would take time to go and search for execution flag as **Yes**, execute that case and again come and search for the next **Yes** case.

Here, we would like to suggest 2 approaches which would consume less time when compared to the previous way.

Macro Approach: We shall have the same execution flag. But before starting the execution, just run through a code which will search for the cases set as **Yes** in the flag, copy those cases to another excel. Now using that excel, execute those cases.

In this approach, since the cases to be executed would already be copied in another excel, it will just run through all the cases and execute all those. There is no need to wait to get the case checking for the next **Yes** flag.

Traditional Approach: We can have all the cases, that are to be executed, manually placed in a single excel and execute that excel separately. Also when you are executing whole module, it's better to have all the modules separated to single excel worksheets. So when you are required to execute that module, just select that product.



B	C	D	E	F	G
Test Case Criticality	Products	User Type	Environment	Browser	Version
All	Prod 1	Admin	QA	Internet Explorer	QA v1.3

6.2.2 Usage of Macros

Since we run around data mostly using Excel Datasheets, instead of writing codes to do all operations in Excel, the usage of Macros would solve most of the problems. Here we have listed some advantages of usage of Macros over normal scripting process:

1. Will reduce the possibility of human errors, reduces repetitive keystrokes
2. Reduce the amount of time taken to perform some basic tasks
3. Since it's directly integrated to excel, no need to create and connect the object to excel and perform operations, as we do while scripting
4. Easy to find the macro in single excel, instead of trying to find the function written in libraries of the Framework

7. Case Study

The case study taken here is the real time example that we had experienced in one of our projects. There were more than 3000 cases to be automated and executed. It was difficult to execute all the cases at a time as a single whole set. It used to take days together to be executed.

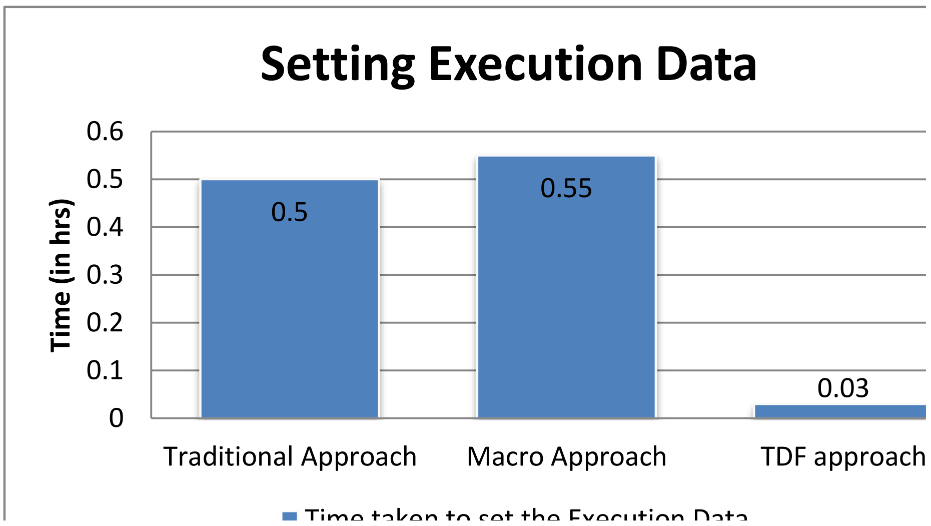
Here are the 3 approaches we tried to create the Execution Data for around 100 cases which were not in order.

Traditional Approach: As mentioned before, we started off with the Traditional Framework. This used to consume more time as we wanted to select flags for all the cases at a time and then start the execution.

Macro Approach: We came up with the use of macros, which would move the required test cases to be executed to one particular excel. Even this would take a little more time as it was required to enter the test case numbers which are to be executed.

TDF Approach: This is called as Test Data File Approach. We followed a new approach of manually copying the cases to a new excel file. This consumed very less time in comparison with the previous 2 approaches.

The below chart shows the time consumption for all 3 approaches:

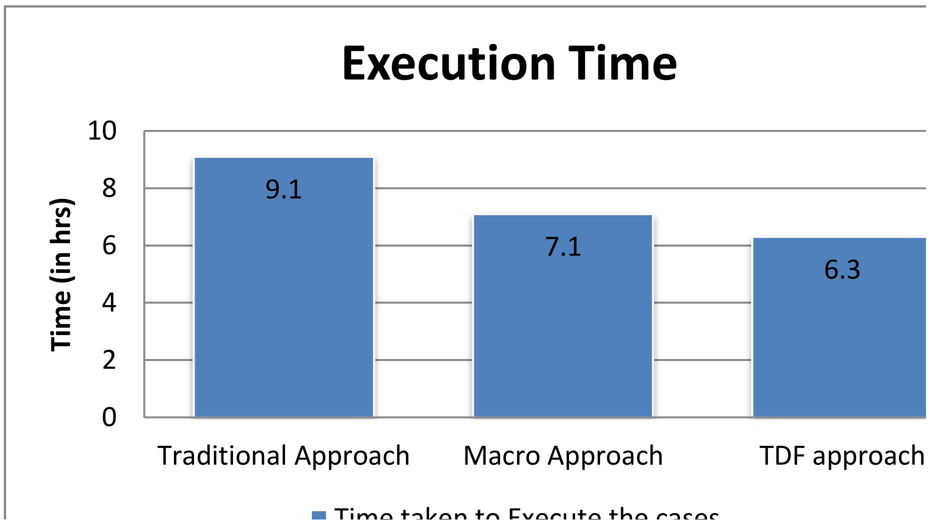


Test Execution of the test cases was tried with all the 3 approaches. We had very huge difference in the execution time. As an example, here we shall consider execution for those 3000 cases taken before while Setting Execution Data.

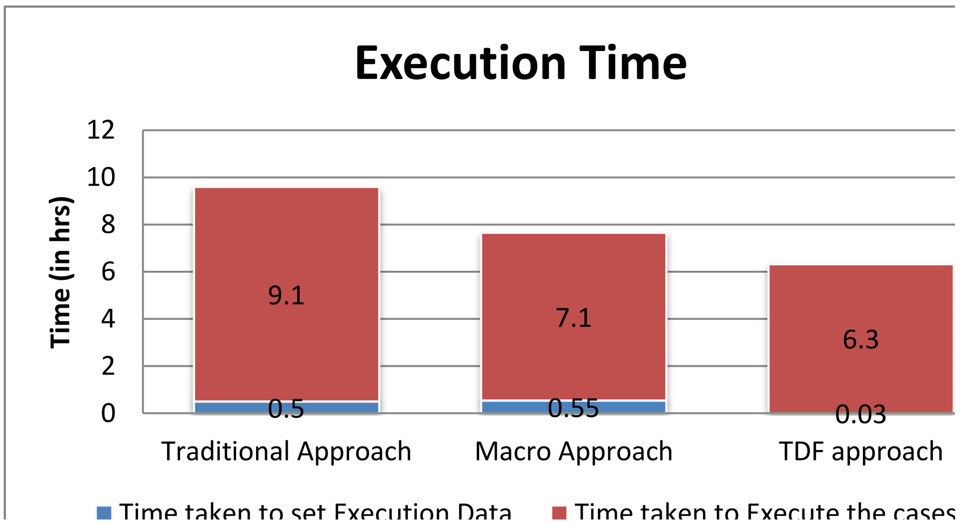
Traditional Approach: Here, the script would run through all the 3000 cases in the Excel, search for the case for which the Execution flag is True and then go and search that case in the Execution Data and execute it. Even though the case to be executed is at the last of the count, it used to search all the cases from 1st.

Macro Approach: In this approach, the macro would run through the Excel which we have created with only the test case names to be executed, then go and search that case in the execution data and add it into new Excel, from which the Execution would be triggered. This again would consume more time as it searches and appends data to another excel.

TDF Approach: This approach was very easy as there was no need for searching or rearranging. All the data in the Execution Data Sheet would be executed without consuming much time.



When we combine both the time taken to set the Execution Data and Script Execution, we can get the overall time taken for the execution of those 100 cases. We can examine and observe that there is drastic change in the overall time taken for execution. If the Traditional Approach takes 9.6hrs to complete the full execution, the TDF Approach takes just 6.33hrs to do that. The following chart depicts the same:



When there is a difference of approx. 3hrs in the execution of 100 cases, just imagine how much time and effort can be saved for the execution of 3000 cases.

Apart from this, there was a new logic called **Day Slices**.

The execution was to go on for 9 days where some cases would be divided equally on all days. Some cases may have input, output and verification to be done on same day and some cases may have output and verification on different days. There was one Excel which was having the input day and verification day for each case.

Let us consider 100 cases to be executed on all 9 days. Total would be 900 cases. One Excel will have the input day and verification day for all 900 cases. Again here we had 2 approaches.

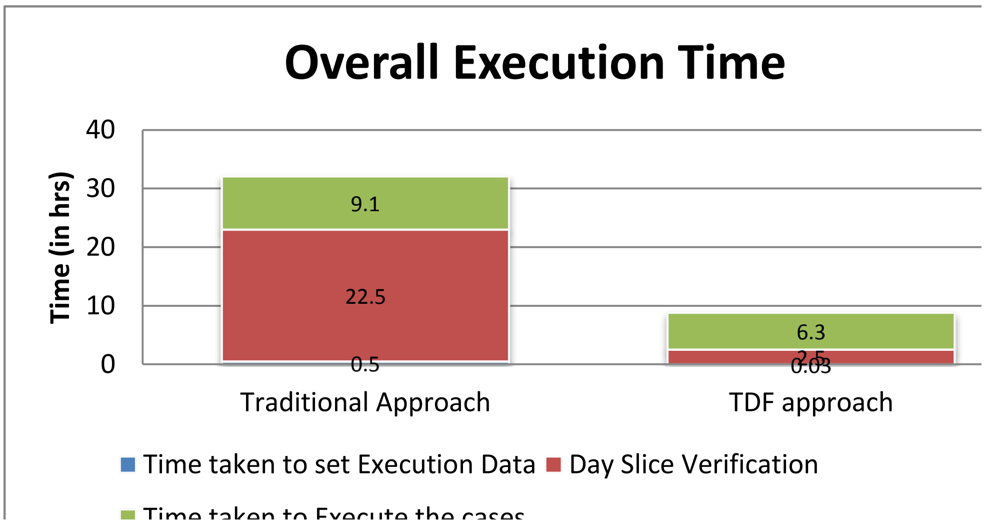
Traditional Approach: Having all 900 cases in one Excel and executing each day. This would take a long time as every time it gets a case from execution data, it would go and check whether that is to be executed on that particular day. Approximately, to just check whether the cases must be executed on that day or no, it used to take 1.5 min. In that case, the time taken would be:

Setting Execution Data (0.5hrs) + 900 * 1.5 = 1350mins (22.5hrs) + Execution time (9.1hrs) = 32.1hrs

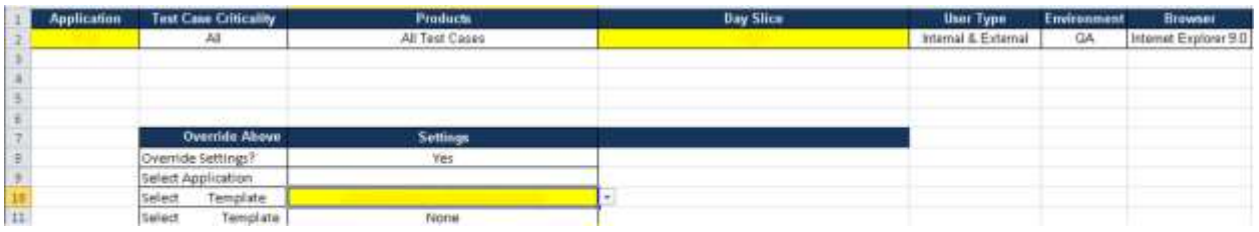
TDF Approach: Here, since the 100 cases to be executed would be ready in the Excel, it would only take time to verify whether that is to be executed on that day and then execute it. In this case, the time taken would be:

Setting Execution Data (0.03hrs) + 100 * 1.5 = 150mins (2.5hrs) + Execution time (6.3hrs) = 8.83hrs

Using TDF Approach, it is evident that the overall execution time was decreased by around 1/3 times to that of the Traditional Approach. The following chart represents the Overall Execution Time:



To achieve this, there was just a small modification that was made in the **Suite Settings**.



1	Application	Test Case Criticality	Products	Day Slice	User Type	Environment	Browser
2	[illegible]	All	All Test Cases	[illegible]	Internal & External	QA	Internet Explorer 9.0
3							
4							
5							
6							
7		Override Above	Settings				
8		Override Settings?	Yes				
9		Select Application					
10		Select Template					
11		Select Template	None				

We added the test case criticality with the products to be executed along with the Day Slice to specify which day we needed to execute. Then we were overriding the settings using Macros depending on the Template/Product we select to execute.

This approach is an enhancement of **Traditional** and **Macro** approach with some brain work!

8. References

- Wikipedia.com
- Springer.com
- Stickyminds.com
- ACM.org
- Automationframework.info

9. Author's Biography

Sandeep has 2 years of experience in IT industry. He has worked as an Intern (Junior Developer) in Buffered Software Solutions for 6 months. Presently he is associated with Indium Software since February 2013. He acquired his Post Graduation in MCA under Bangalore University. He is a DotNet and 3D Animation certified professional. He is interested in development of small applications for mobiles and computers. In his spare time, he reads novels, plays games, watches movies and listens to music.

Nivedita has over 3.5 years experience in Manual and Automation Testing in Indium Software and has worked on HealthCare and Banking domains. She holds an Engineering degree in Computer Science from Kumaraguru College of Technology, Coimbatore. Her hobbies are reading books, listening to music, travelling and playing games.