# Introduction

It’s been almost 11 years in the software testing industry and I have seen a lot of changes in the way the industry has progressed in terms of its view on quality and the importance of testing the software.

During the start of my career, I got a feeling that Quality Assurance is just a supporting team which ensures that we are delivering the right product to the clients. The QA team used to be the final gate-keeping authority which would examine the software and validate it for functionality and also some non-functional aspect like the look-and-feel of the software, its performance under load, accessibility in order to cater to the needs for the specially-abled people, security and many more aspects.

The entire team was broken down accordingly and various phases of testing would be executed based on the Test Strategy and Test Plan outlined at the start of the project. The team would raise defects and then would begin a duel between the Development Team and the Testing Team about *whether it is a defect or not a defect*, *how did we miss out this crucial defect*, *this defect is not reproducible on my machine* and so many other hilarious conversations. This was when the waterfall models or the V-models were in existence where the lifecycle of the project would have various phases and after a year or two the software would be delivered or the website would be live for the various audience. However, it resulted into issues like:

1. Production defects being costly
2. Cost of adding new features were too high and time-consuming
3. Lesser collaborations between various teams
4. Time-To-Market was higher

Since the timeframe was higher, some improvements to the process were eminent. The first team to face the challenge of improvisation would be the QA team. Why? Because in those days, the QA team was considered to be just a support team whose existence could neither be attributed nor be undermined. Thus was born the world of automation. It started with tools which could simulate an action on the application with some verification points. Tools like WinRunner, QTP etc. provided a mechanism to record and playback the actions on the application. Then came tools like Selenium which started off as just a Mozilla Firefox plugin called Selenium IDE which then got built into a Selenium Core and Selenium Remote Control allowing you to write code to test your web application. Eventually, it went on to become Selenium WebDriver which currently is a major tool being used in the Testing Field.

Though it reduced the Time-To-Market to some extent, there were still some of the problems which the teams encountered with automation. Some of them were:

1. Since the tools were point-and-click, changes in the application would render the previously recorded script useless and the recording had to be redone.
2. With tools like Selenium RC or Selenium WebDriver, writing scripts consumed a lot of time and was again an overhead. The more regression scripts and the more the regression cycles the better the Return on Investment (ROI) for the software development. That would then become a yardstick to choose whether to automate or not to automate.
3. Automation had a lag of one complete release cycle or there was partial automation and partial manual regression testing for the initial release

Also the processes still remained the same which were in phases. That’s how, Agile processes came into existence where the product was broken down into smaller chunks and there was an incremental delivery of the software application. This resulted in two key benefits:

1. Faster Time-To-Market
2. Better collaboration between the teams

Automation in the Agile world was included as part the *Sprint.* However, some of the issues remained the same in terms of automation. They were:

1. Scripting time still constituted to be an overhead
2. Modularizing or Framework-ising the test automation consumed more time before starting to develop the scripts
3. Reliability on a stable application meant more maintenance or rework if the application underwent change in design/refactor

That’s when we talked about Shift-Left and Shift-Right approach to also support Release Deployment and various other aspects which today is defined as DevOps (Development and Operations put together becomes DevOps).

**What is Shift-Left approach?**

Shift-Left approach attempts to have more collaboration between the Development team and the Testing team. One of the way this is achieved, is by setting up a Test Driven Development (TDD) or Behavior Driven Development processes which means the Test drives the Development or the Development Team writes Tests first and then develops the code that conforms to the test.

**So then, what is Shift-Right approach?**

Once and application is developed and tested, it goes through a process of deployment. The process involves:

1. Integrating the latest stable build
2. Ensuring that nothing broke during the integration
3. Installing the pre-requisite software and utilities to the production environment
4. Deploying the application
5. Smoke Testing the application
6. Go Live

This is a tedious task and takes days or weeks to accomplish. That’s how the Shift Right approach was devised where the Testing team is involved in assisting the Operations team to deliver the product with the right code, with the right configuration and with the right features that is committed to the clients. That’s how the DevOps team came into existence.

Coming back to our discussion on Automation with the Shift-Left approach, the QA team can write automated test scripts using BDD tools like Cucumber, JBehave, SpecFlow etc. and execute the test scenarios even though the application is under development. This leads the tests to fail at the start and the tests are expected to pass in order to deliver the application successfully. I will discuss this in a subsequent chapter as to how this can be achieved. However, for now hold on to this thought and let’s move forward.

In this book, I intend to touch upon the Science of Test Automation. This book is not intended to make you an expert in a particular tool like Selenium or QTP because tools come and go. But what I found out during all these years is that some of the concepts of Test Automation remains the same. Some of the pains in Automation in the fast moving Agile world of Software Development remains the same. That is what this book is intended to do, *To change the way you think about automation and to change way you Approach Automation.*

I hope and wish that after you finish reading this book, there would be a paradigm shift in the way you look at automation. Let’s move on…

# What to expect from this book?

You might have bought this book with an expectation to learn a tool like Selenium or UFT. Sorry to give you the bad news, you have probably picked up the wrong book. My intention here is not to teach you an automation tool. Like I said before, tools come and go, but the principles of setting up the Test Automation Suite remains unchanged irrespective of the tools you use and irrespective of the project you are working on.

However, here is some good news. I will be dedicating one chapter to some of the features of the Selenium Webdriver. Also, during the course of this entire book, I will be using Selenium Webdriver as a reference to talk about the different mechanisms in Automation. I will be using Java language for the simple reason that I have used Java extensively during the course of my career as a Test Automation Specialist. However, the principles and techniques showcased here will assist you in implementing it in other tools that you might use apart from Selenium Webdriver. My opinion about automation tool is that it should suffice 2 basic needs:

1. How well it supports the application platform i.e. Browsers, Operating Systems etc.
2. How well it identifies the Test components

If the above two criterions are sufficed, I would go for that tool irrespective of the other features that it might have over and above this. Being an architect or an automation expert, it is your objective to make it a robust solution irrespective of the tools you might use.

This book will also talk about modularizing your frameworks to allow you to loosely couple the different components. I have termed it as “Controllers” and will be detailing out the complete science behind each controller being developed. Each controller is an individual chapter and will answer three basic questions pertaining to a controller:

1. Why this controller?
2. What is the role of this controller in an automation framework?
3. How to integrate and consume this controller within the automation framework?

Another really painful aspect of the Test Automation is the way Test Data is managed. This is the biggest nightmare facing the Test Automation teams. Too many times, the script failures and maintenance can be attributed to the unavailability or redundancy of the test data. I will be covering a simple technique that will allow us to improve the way Test Data is managed and also keeping the Test Suite more relevant for a longer period of time.

Now that I have set the context for this book, let’s get straight into Why Automation?

# Why Automation?

Software Testing has come a long way from being debugging-oriented to today’s modern approach where Testing Drives the Development of the Software (Test Driven Development or TDD and Behavior Driven Development or BDD). In one of the conferences recently that I attended, one of the esteemed speakers, talked about how Testers will assist their Clients to come up with requirements that will help their product stay relevant and create a great experience for their audience.

Imagine a possibility where the testers gleans through a set of user’s use patterns and the debug logs or analytics for a given application and comes up with a solution to the client that addresses the User Experience on the application. We are already talking about the world of Internet of Things (IoT), Machine Learning and Artificial Intelligence and there is no better time than in the history to be a Software Tester in my opinion. Having said that, it is also important that we as Software Testers, upgrade ourselves to these changing expectations. From being a support team that keeps track of the quality of the software to becoming a consultant for the software solution is the journey we have been through. What was the expectation from us yesterday cannot be the expectation from us today and tomorrow. Hence, we should be up the challenge and the newly upgrade role that we play in ensuring software quality.

In upgrading ourselves, we should also have an arsenal of tools at our disposal to help us do a better job to Testing. Test Automation is one such tool. Today, when we talk about a software application, we consider the following aspects:

1. What features are supported by the application?
2. Is it a Web application and/or does it have a Mobile Version?
   1. If it is a Web Application,
      1. Which all browsers does it support?
      2. Which operating system does it support?
      3. Which screen resolution does it support?
      4. Is the web design responsive?
   2. If it is a Mobile application,
      1. Does have Android, iOS and other versions?
      2. Is it a Mobile Web, Native or Hybrid Application?
3. Can it be integrated easily with other applications?
4. Does it scale well?
5. Is the Web Design contemporary?

All the questions, leads us to think,

* How am I going to assure quality with so many aspects? or
* Do I have the time to validate all these functionalities?

That’s why it is important that we consider having a Test Automation Solution that can allow:

1. Testability of all the features
2. Testability on all the platforms
3. Testability with other applications for interoperability

That leaves you as testers, to focus on the Business Aspects, Context-Driven Testing, User Experience Testing and other Thoughtful Testing.

Another aspect to considering Test Automation is the Agile practices and methodologies that are being practiced in the Software Development Lifecycle. Catering to the ever changing requirements sometimes puts quality into jeopardy given the time crunch and the wide array of platforms to test the application. It also sometimes leaves us with a need to validate the previous features and functionality that might be broken as a result of the new requirements. The whole exercise has to be done over and over again to ensure that the quality is not compromised at any given point of time. What Test Automation intends to resolve here is:

1. More repeatability of the tests
2. Reducing efforts on regression Testing
3. Freeing up the time for the Tester to do more thoughtful testing
4. Quick Validation on the various platforms

With that said, it is important that while designing the Test Automation Framework, the following aspects are considered:

1. The framework should be scalable to,
   1. Multiple platforms
   2. Multiple technologies
   3. Multiple execution environments
2. The framework should be easy to maintain
3. The framework should have better validation mechanism
4. The test executes unattended.
5. The test data used should be relevant at all times.

From a business standpoint, one of the important indicators would be the Return on Investment (ROI) for the Test Automation. This is the bottom-line that every client is interested in viz. How much of savings do I make by putting together a Test Automation Suite. The way the ROI is calculated is different given the way the Software Teams are positioned.

If the application is developed by the employees of the organization, then the ROI depends on the number of days/hours reduced to market the product versus the investment in Test Automation. In this case the assumption is that, if the testing was carried out manually, what is the time taken by the manual testing team to complete the testing of all the functionalities on various configurations, platforms and how automating these functionalities has reduced the efforts and the accuracy of the execution results.

For an organization which outsources the development of the application, the ROI depends on the number of days/hours reduced in billing for the outsourced resource versus the investment in Test Automation. Again in this case, the ROI is calculated based on the assumption of the efforts carried our manually against the automation efforts.

In both the cases, from an automation standpoint, we also need to consider the efforts for executing the scripts manually in case of false-failures or analyzing the failures.