Contents

[1. Introduction: 1](#_Toc528662804)

[2. Why automating: 1](#_Toc528662805)

[a) Speeding delivery and save time: 1](#_Toc528662806)

[b) Accuracy: 2](#_Toc528662807)

[3. Type of automation tests: 2](#_Toc528662808)

[a) Automated Unit Tests: 2](#_Toc528662809)

[b) Automated Web Service / API Tests 2](#_Toc528662810)

[c) Automated GUI Tests. 2](#_Toc528662811)

[4. Used tools: 2](#_Toc528662812)

[5. Our approach and strategy: 3](#_Toc528662813)

[a) Building framework: 3](#_Toc528662814)

[b) Start building scripts: 3](#_Toc528662815)

[6. Notes: 4](#_Toc528662816)

[7. Conclusion: 4](#_Toc528662817)

# Introduction:

In QA, automation testing is seen as one of the biggest promoters of speed. It is critical to maintaining quality during fast release cycles. Software tools can run automated scripts that help testers reduce repetitive tasks and shorten the time it takes to produce quality testing results.

# Why automating:

We usually automate for the below major reason:

## Speeding delivery and save time:

[Software tests](https://smartbear.com/products/testing/) have to be repeated often during development cycles to ensure quality. Every time source code is modified software tests should be repeated. For each release of the software it may be tested on all supported operating systems and hardware configurations. Manually repeating these tests is costly and time consuming. Once created, automated tests can be run over and over again at no additional cost and they are much faster than manual tests. Automated software testing can reduce the time to run repetitive tests from days to hours. A time savings that translates directly into cost savings.

## Accuracy:

Even the most conscientious tester will make mistakes during manual testing. Automated tests perform the same steps precisely every time they are executed and never forget to record detailed results. Testers freed from repetitive manual tests have more time to create new automated software tests and deal with complex features.

# Type of automation tests:

There are three main types of automated tests. However we can there are other areas we can automation likewise data preparation, create Excel macro…etc.

## **Automated Unit Tests:**

[Automated Unit tests](https://www.softwaretestinghelp.com/unit-testing/) are written to test on code level. Bugs are identified in the functions, methods and routines written by developers.

## **Automated Web Service / API Tests**

What we test here is usually the functionality, compliance and security issues. In web applications, we can test the Request and Response of our application that whether they are secure and encrypted or not.

## **Automated GUI Tests.**

This type of automated testing involves testing of a User interface of the application. this consists of validations on GUI and validates the displayed data and verifies that there is no broken links also takes screen shots …etc. this relatively the toughest as it needs maintenance as GUI’s are highly subject to change.

# Used tools:

There are many tools in the market, however, there few that are highly known and commonly used.

* HP-UFT(Unified functional testing): is an HP licensed product and uses VBS us programming language.
* Selenium: Is an open source tool that is viable in six programming languages including Java which makes it the most used in the market.

# Our approach and strategy:

## Building framework:

Every company has its own framework which is structured folders and packages that will contains all commonly used classes and functions that we can call keywords. This may take some time in the beginning as at this step, the code of every keyword(function) will be written and tested with different scenarios so it can be used in every created script in the future. This will avoid a lot of double effort by automation testers and will make it easier for any eventual maintenance in the future. This also can be used across all teams and different projects. For instance, we will create Gui-library class that will contain all the keywords that we may need to perform a GUI testing like clicking on a link/button, entering text, take screen shots…etc. The best way to perform this is to create these keywords on real life project (ADCVD - salesforce project in our case), which will allow us to cover one project while we are building this framework. Besides, most of the project are on salesforce platform, so the reusability will be very high therefore we will need shorter LOEs in the future.

## Start building scripts:

Usually the functional team elects the test cases to be automated from for their projects, these can be from regression, progression or smoke test suites. Then, a meeting with automation tester will be scheduled to discuss the below points:

* ROI: return on investment is the crucial point of doing automation. To automate any functionality, the team needs to be agree upon the worth of automating and should be beneficial for QA process. ROI is calculated based on multiple factors like LOE (Level of effort), which is the time spent by automation tester. How often the script will be executed and what is the time saved for every run. Also, what is the time it takes for script to run versus the manual effort.
* KT: After the ROI is validated, the functional testers should explain the requirement for the automation tester and what are all the possible scenarios and what are their expectations.
* Feasibility: After requirements are clear, the automation tester analyzes the feasibility. Even though most of functionality can be automated, however, there are some limits to every automation tool, besides other restrictions that can be related to security standards set by the company …etc.
* After both parts agreed to opt for automation for any given functionality, the automation tester should do quick analysis on the best tool to achieve efficiently the requirements and send out the LOE for leads for review.

# Notes:

1. working closely with developers and functional testers will facilitate this process and automation testers will come up with more functionalities to be automated rather than waiting for functional team to bring it up. The more automation we do the more time saved and the better quality of the product to be tested.
2. This is the beginning of automation, we will, in the future, do more reliable implementation when the automation team grow up. Like we will need to have GitEye server for versioning management. Maven to manage builds and automatic dependencies. And Jenkins to integrate our scripts for automatic and periodical run whenever the deployment of the new code occurs.

# Conclusion:

Writing automated test scripts can be time consuming. It’s not possible to automate everything, so the key to getting maximum ROI from time and money spent on automation is to develop a strategy that increases velocity in the short *and* long term. [Keyword-based tests](https://smartbear.com/learn/automated-testing/benefits-of-keyword-testing/) allow QA engineers to boost their ROI on time spent. These tests are a lot faster to write because they run on keywords understood by the app and the automation software instead of using a complex scripting language.

Create Framework:

We will be using Selenium/testNG to develop our framework. We opt for this solution as this the most powerful automation testing tool in addition to the fact it is an open source.

# Type of Framework:

This is, at the same time, data and key word driven framework.

Data driven: We try to 100% separate data from the code and no data should be hardcoded, so testers have the control over the data inputs and manage their scenarios without a need to access or know the code.

Keyword driven: the framework is a set of libraries of interconnected functions (Keyword) which is a piece of code with a meaningful name, so anyone can call it in his script as needed. This allow us to create every function just once to avoid redundancy.

# Objectives:

## Reusability:

The framework will contain all common functions required to perform DB, SOAP, GUI…etc testing. It will be used in any project which will save us tremendous amount of time. And if in any given project we need to create a function or library that doesn’t exist yet, we will create it and add it to the framework. The more we call it from different project, the more we enhance it and the more reusable it will be.

Independency: The framework should be as independent as possible. It will be called in all projects regardless of their platforms. For example, GUI functions should work with any web application (Salesforce, Peoplesoft, … etc.) This way, this can be used across project and across team.

Flexibility: We create this framework as a separate project, so it can be opened by anyone in any IDE (Eclipse, inteliG…etc.) for updates or enhancements. Also, it can be called by any other script as external project or as a Jar file. For Testers side, this can be delivered always as a Jar file, so nothing will be required to be installed in their machines besides Java.

Created Items:

This framework is set of functions (Keywords) grouped in categories called libraries. We create code for every keyword with its parameters and input/outputs.

Enhanced Items:

TestNg Framework not always suit our aims and expectations; therefore some enhancement should be performed.

1. In testing original framework, the user should enter the test cases to be executed in testing.xml before every run, this may look tedious for functional testers. We built a functionality to transform an Excel file, where users just have to put True/false for test cases to be included/excluded, into this required testing.xml
2. TestNG comes with built-in output reports which is not as detailed as we want it. We created our output report which is less congested and more user friendly.