Syrian Private University

Faculty of Informatics Engineering

Senior project – part 2

Automation Testing Tool for Web Application

Senior project

Version 2.0

By:

Ali Al Khalil

Fayez Al Ibrahim

22/2/2020

Table of Contents

1. Introduction:

Automated testing is an ideal way of ensuring that new versions of an application don't introduce bugs or performance issues. It allows development teams to complete projects more quickly because they can quickly verify functionality after each change, Test Automation has become an important phase of the development process. Verifying the functionality, testing for regression and executing the tests are part of the Test automation process. These are carried out simultaneously and in an efficient way. Manual testing is the traditional method adopted by organizations. However, with the increasing number of web- based applications and quality tools in the market, this is slowly vanishing. Verification and testing web-based interfaces are made easier with test automation.

1.1 Purpose

The purpose of this document is to give a detailed description of the requirements for the "Automation Testing tool" (ATT) software. It will illustrate the purpose and complete declaration for the development of system. It will **also** explain system constraints, interface and interactions with other systems. This document is primarily intended to be proposed as a report for a senior project, in this Release we will cover the below main points:

- Add/Edit projects
- Add/Edit requirements
- Add/edit test cases
- Run test cases manually & automatically
- Creating defects regarding failed test cases
- Backup and log all actions and process

1.2 Scope

Our testing tool is a free and robust automation solution for Web testing. It integrates all necessary components with built-in keywords and project templates into a complete automation framework. this tool is easy to use for beginners but still offers advanced capabilities for experienced users.

We wanted to develop two separating systems, projects repository and the automation testing tool, we also make sure that tester can use the first system "Projects repository" separately if he wants to, in case the tested application is not automated for any reason and must be tested manually.

Although we cannot use the automation testing tool without the project repository in order to edit, retrieve, and save the result of the automation testing.

1.3 Definitions, acronyms, and abbreviations

Table 1 – Definitions

Term	Definition
Preparation	Creating test cases phase
Execution	Running the test cases against the system
ATT	Automation Testing Tool
Test case	A test case is a specification of the inputs, execution conditions, testing procedure, and expected results that define a single test to be executed to achieve a particular software testing objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
Defect (BUG)	is any variance between actual and expected results according to Requirement
SRS	is a testing approach in which test cases, conditions and data are derived from requirements, it includes functional tests and also non- functional attributes such as performance, reliability or usability
Regression testing	is the process of testing changes to computer programs to make sure that the older programming still works with the new changes. Regression testing is a normal part of the program development process and, in larger companies, is done by code testing specialists.
Non-Functional Testing	Such as Performance, endurance, load, volume, scalability, usability so on
Smoke Testing	is a kind of Software Testing performed after software build to ascertain that the critical functionalities of the program are working fine. It is executed “before" any detailed functional or regression tests are executed on the software build
DUSK	It’s a Laravel framework built on php unit provides an expressive, easy- to-use browser automation and testing API. By default, Dusk does not require you to install JDK or Selenium on your machine. Instead, Dusk uses a standalone ChromeDriver installation. However, you are free to utilize any other Selenium compatible driver you wish.
ChromeDriver	is a separate executable that WebDriver uses to control Chrome. It is maintained by the Chromium team with help from WebDriver contributors
RUN	**Executing the test cases through the automation testing too**

1.4 Problem

As testing process consists of many levels and methods, in some levels we may need to repeat a specific test many times, which is a time consuming and a waste of recourse whether human resources, time, or even money and usually, the application development process became worthless because exceeding the expected time frame and its deadline, In each Regression phase of each release of the application we must make sure that the pervious tested release is still working and no changes occurred, and also to make sure that all functions are running perfectly.

And to reach this point, testers have to re-test all the related test cases and especially the ones in the execution phase, which time and resource's consuming, In the next point we will discuss cons and pons of each test and when to automate.

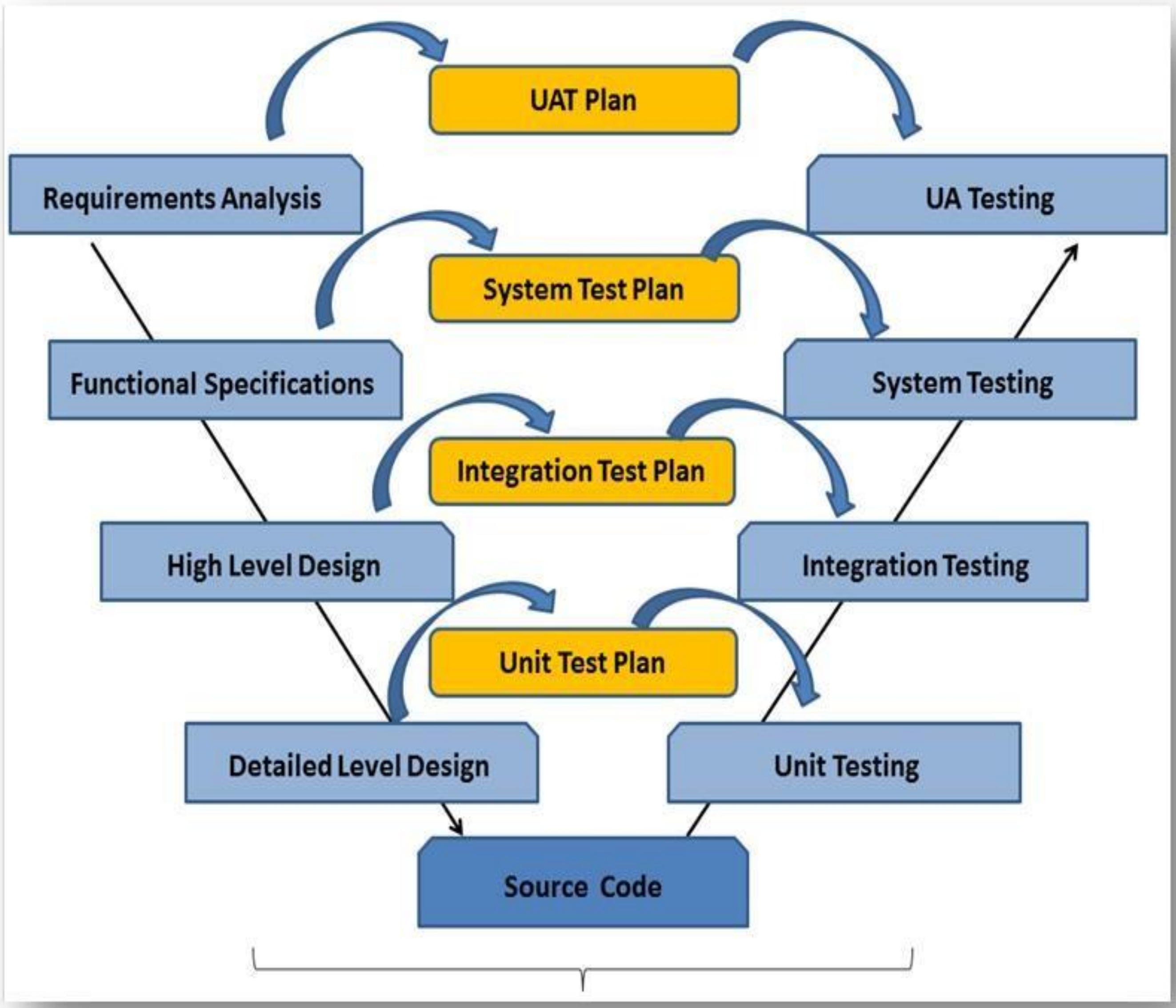


Figure 2 -testing levels

1.5 Cons Vs Pons

1.5.1 Pros of Automated Testing:

❖ Runs tests quickly and effectively

While the initial setup of automated test cases may take a while, once you've automated your tests, you're good to go. You can reuse tests, which is good news for those of you running regressions on constantly changing code. You won't have to continuously fill out the same information or remember to run certain tests. Everything is done for you automatically.

❖ Can be cost effective

While automation tools can be expensive in the short-term, they save you money in the long-term. They not only do more than a human can in a given amount of time, they also find defects quicker. This allows your team to react more quickly, saving you both precious time and money.

❖ More interesting

Filling out the same forms time after time can be frustrating, and not to mention boring. Test automation solves this problem. The process of setting up test cases takes coding and thought, which keeps your best technical minds involved and committed to the process.

❖ Everyone can see results:

When one person is doing manual testing, the rest of the team can't see the results of the tests being run. With automated tests, however, people can sign into the testing system and see the results. This allows for greater team collaboration and a better final product.

1.5.2 Cons of Automated Testing:

❖ Tools can be expensive

The automation tools can be an expensive purchase. As a result, it is important to only use the ones that will give you full, or as close to full coverage, as you can find, The best applications for implementing automated testing are when the tests are repeatable and it's necessary to run them many times (either because the app will have many versions, maintenance reasons, or because it must be tested on different platforms). This is also known as regression testing.

❖ Tools still take time

While the automation process cuts down on the time it takes to test everything by hand, automated testing is still a time intensive process. A considerable amount of time goes into developing the automated tests and letting them run. For example, a large client of ours ran into trouble when their daily run of automated tests exceeded the 24-hour mark.

❖ Tools have limitations

While automated tests will detect most bugs in your system, there are limitations. For example, the automated tools can't test for visual considerations like image color or font size. Changes in these can only be detected by manual testing, which means that not all testing can be done with automatic tools.

1.6 Manual Testing

Manual testing is the process through which software developers run tests manually, comparing program expectations and actual outcomes in order to find software defects. These manual tests are no more than the tester using the program as an end user would, and then determining whether or not the program acts appropriately. Manual testing is a good fit for smaller projects as well as companies without significant financial resources.

1.6.1 Pros of Manual Testing

❖ Short-term cost is lower

Buying software automation tools is expensive. With manual testing, you won't have to put the same up-front costs into the software, so this is the team leader decision decide if the project will be tested manually or through the automation testing tools based on his own experience and on specific criteria.

❖ More likely to find real user issues

Automated tests are just that – automatic. They're robotic and don't necessarily act as a real user would. Manual testing, on the other hand, allows the developing program to be used as it would be upon launch. Any bugs that may pop up when a user handles the program in a certain way are more likely to be caught with manual testing.

❖ Manual testing is flexible

When one of those brilliant thoughts comes to you, something that could change the course of the project, you want to be able to work on it immediately. With automated testing this is difficult. You have to set up test cases, program it into the automated tool, and then run the tests. With manual testing, you can just quickly test and see the results. Automatic tests take more time to set up, which doesn't allow you to test ideas quickly and easily, Manual testing is cost-efficient for the tests that are run only a few times, some tests like usability testing can be most effectively done by humans manually, A software's UX, especially a graphics heavy, can only be best evaluated by human testers.

1.6.2 Cons of Manual Testing:

❖ Certain tasks are difficult to do manually

There are certain actions that are difficult to do manually. For example? Low level interface regression testing. This kind of testing is extremely difficult to perform manually, and, as a result, is prone to mistakes and oversight when done by hand. Automated testing, once set up, is much better equipped to find errors for this kind of testing.

❖ Not stimulating

Manual testing can be repetitive and boring – no one wants to keep filling out the same forms time after time. As a result, many testers have a hard time staying engaged in this process, and errors are more likely to occur, an automated testing tool is able to playback pre-recorded and predefined actions, compare the results to the expected behavior and report the success or failure of these manual tests to a test engineer. Once automated tests are created, they can easily be repeated and they can be extended to perform tasks impossible with manual testing.

❖ Can’t reuse manual tests

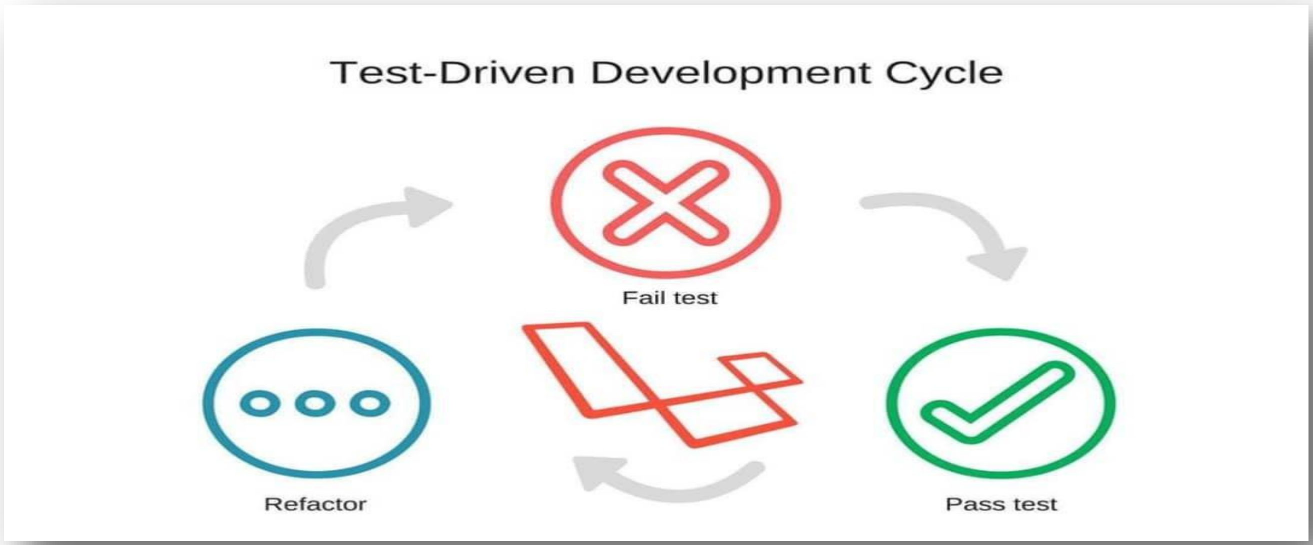
With automated tests, if you add anything to the program, you can rerun all of the required tests instantly the tests are already set up. This isn’t the case with manual testing. If there is any change to the software, you have to run the tests again by hand. This is valuable time lost. And its Involves more cost as human test resources are more expensive than test machines, So, the simulation of large numbers of users or configurations is nearly impossible to accomplish manually, and Manual testing is prone to human error, so may lead to inconsistent or misleading results.

2. Overall description:

Automation Testing means using an automation tool to execute your test case suite, the automation software can also enter test data into the System Under Test, compare expected and actual results and generate detailed test reports. Test Automation demands considerable investments of money and resources.

Successive development cycles will require execution of same test suite repeatedly. Using a test automation tool just like our tool, it's possible to record this test suite and re-play it as required. Once the test suite is automated, no human intervention is required. This improved Test Automation. The goal of Automation is to reduce the number of test cases to be run manually and not to eliminate Manual Testing altogether.

Our system will let the tester to enter and save the testcases, then and after edit the format, system will be able to run these test cases by its own, and as many times as we want, In order to let the tester able to check the results, we user another system “Project Repository”, where all projects, test cases defects, and many other items are saved and retrievable by our testing tool, Noting that “Project Repository” is a fully Independent system and can be used for manual test.



2.1 Product perspective

The project will consist of two parts: the automation test tool and the project repository, the testing tool is built using a framework called DUSK which is basically created by Selenium, software tests 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 Php unit is based on the idea that developers should be able to find mistakes in their newly committed code quickly and assert that no code regression has occurred in other parts of the code base. Much like other unit testing frameworks, PHPUnit uses assertions to verify that the behavior of the specific component - or “unit” - being tested behaves as expected. Manual software testing is performed by a human sitting in front of a computer carefully going through application screens, trying various usage and input combinations, comparing the results to the expected behavior and recording their observations. Manual tests are repeated often during development cycles for source code changes and other situations like multiple operating environments and hardware configurations.

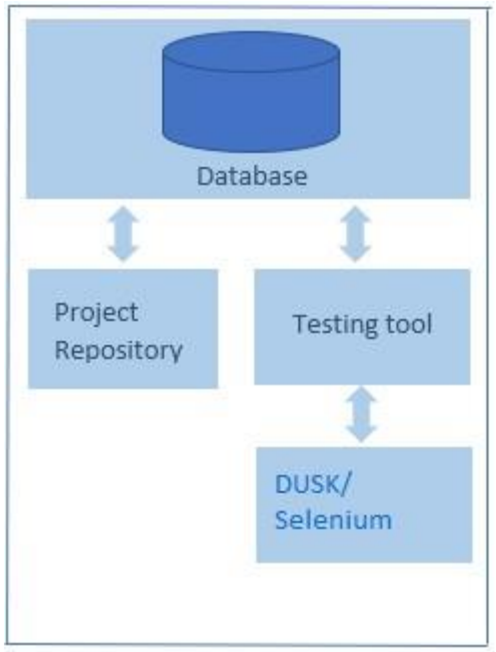


Figure 1 - Block diagram

2.2Product functions

2.2.1Shared function

- **Create Test case**

Tester will be able to create test case using any of the two systems, and while "project repository" will be used regarding manual test, so there is big deal if the tester did not add specific and details steps, and sometimes it's enough to enter only test case title and that will be sufficient to deliver the required idea of the test, In other side, tester will be required to enter full and detailed testcase steps, so the system can understand the needed end point.

Noting that each details step will contain basically the action and the object where this action should be performed.

- **Create Bug**

In automation testing tool and after run is finished, tester will only have to click one button to create defect with its error message, Where the process is a little bit complicated in the manual way where tester have to enter the defect description

- **List Projects:**

In both system, tester will be able to list all project where click on each project will lead to its requirement page

- List **Project Requirement**

Each project is consisting of modules and for each module there should be a detailed report called Requirement, and in both system, tester will be able to list all project Requirement where click on each Requirement will lead to its testcases page.

- **List Requirements test cases**

For each test case, title, status, id and assigned to information will be available in the test cases listing page.

Requirement details page will consist of Requirement title according to the module and Requirement content and a link leads to this Requirement related testcases.

2.2.2 Project Repository functions

- **create project**

Creating new project for each new application where tester should only enter project title and click add button

- **create requirement**

Creating new requirement for each module in the application where tester should only enter requirement title and content.

2.2.3 Testing tool functions

- **Run**

With one click, the automation testing tool will execute each test case in the specific project through Chrome Driver and after run is finished, system will reflect the test cases new status on Database where passed test cases will be marked as passed and closed and for failed test case will be marked with failed and tagged with its error message and other options.

- **show Run Results**

For each failed test case and after the run is finished, the system will show a table contain the test case title and its related error message which clarify why this test case failed in the execution phase. of course, with ability of create defect item.

2.3 User characteristics

As this system as oriented for testers, currently only test character will contact with the system.

2.4 Constraints

By default, Dusk does not require you to install JDK or Selenium on your machine. Instead, Dusk uses a standalone ChromeDriver installation. However, you are free to utilize any other Selenium compatible driver you wish, the Internet connection is also a constraint for the application. Since the application fetches data from the database over the Internet, it is crucial that there is an Internet connection for the application to function.

Both the web portal and the mobile application will be constrained by the capacity of the database. Since the database is shared between both application it may be forced to queue incoming requests and therefor increase the time it takes to fetch data.

2.5 Assumptions and dependencies

our assumption about the product is that it will always be used on local host, also we assumed that a tester will run the system although there is no high level of testing experience required.

Another assumption is that the PC or the laptop will be at least 4 GB RAM in order to keep the performance as expected.

3.0 State of Art

A couple of decades ago, many parts of software were tested only manually or not at all. The integration of testing into development was through a wall over which developers threw the software to dedicated testers. Test coverage analysis and A/B testing were techniques many of us only heard of in college and never saw applied in practice.

The most striking sign of progress is visible in industrial practice, which used to trail academic research at an embarrassing distance but now often leads the way.

First, pair the routines you write with their unit tests. These tests exercise the code in isolation, preventing problems from surfacing during integration. They also promote more modular design, protect you during refactoring, and document how the code you write is supposed to be used. So important are these tests that Michael Feathers considers software lacking them to be legacy code.2 Adopt a framework, such as one from the unit family, for writing and running your unit tests. There are (more than) plenty to choose from; various modern languages, such as Go, Python, Ruby, and Rust, even include unit-testing support as part of their standard library.

Some of you might decide to go even further by adopting test-driven development (TDD), progressing step-by-step by writing a test based on the software's requirements and then implementing the code that implements the test. This development style helps you focus on the requirements from the outset, drives you to design testable software, and ensures that each feature is coupled with its test code. TDD also helps your organization stay honest regarding testing, by minimizing the temptation to skimp on the implementation of tests after the code gets written.

Continue by establishing what Mike Cohn called a test pyramid.3 At the bottom, write plenty of unit tests to ensure that your methods are correct. These are relatively cheap to write, are robust in the face of software changes, and can run very fast. Supplement them with a selective dose of component and integration tests that run below the application's user interface. In modern applications you should be able to write these easily through (for example, REST—Representational State Transfer) service calls. At the pyramid's top, write a few end-to-end tests that exercise the user interface. These can be expensive to write, brittle, and slow, so exercise restraint in what you test here, strive to automate all types of tests. This minimizes their cost, simplifies their running, and offers you many opportunities to measure and optimize the process. Automated tests are the machine oil that keeps the development engine running smoothly. As an added bonus, test automation provides more meaningful and stimulating tasks to testers, letting them, focus on the tests' quality and process optimization, rather than miring them in the drudgery of manually executed test cases.

Code and its tests tend to decay over time. So, ensure that both are always up to scratch by running your tests during continuous integration (CI). Most CI frameworks support this functionality; all you have to do is configure it. By running tests after each commit, you minimize unpleasant surprises during integration. Code committers get an immediate warning if their code broke their own or somebody else's tests. I've seen that this process, when applied to thoroughly tested code, makes it a lot easier to onboard new developers into a project. With guard railings protecting all parts of the code, the chance of somebody driving over the cliff is minimized.

This brings me to another important practice: test coverage analysis. With this, you want to measure and thoughtfully (rather than blindly4) evaluate what code and what percentage of code are covered by tests. Achieving 100 percent coverage is neither easy nor a guarantee of faultless code. However, low or decreasing levels of test coverage are a warning sign that something is amiss. Coupled with automated testing, the measurement of code coverage as part of your CI process with tools such as Coveralls (coveralls.io) can help guide your organization toward a test quality baseline.

When it comes to testing the user experience and usability, automation is more difficult. Nevertheless, there are still methods that can help you a lot. In particular, consider A/B testing, in which you deploy a given feature to only a subset of your user base and compare the two groups' behavior. In services delivered over the web, deploying both versions can be simplified through software option switches, which enable a feature only for specific users. Measuring the two versions' outcomes is also easy, just have your server keep a detailed log of user interactions.

As is always the case in software engineering, the icing on the software development cake entails measurement, evaluation, and improvement. When testing, first examine your test cases' effectiveness. A successful test case is one that catches a bug. For example, testing a class's getters and setters is rarely worthwhile; focus instead on eliminating error-prone boilerplate code with approaches.

Two other metrics to examine are the time it takes for test cases to run and their brittleness. Large code bases are often plagued by long testing times, unreliable test results, and other “test smells.” You can reduce testing times by having test execution tools intelligently select which test cases to run after a specific commit. Increase test stability by flagging and correcting nondeterministic test cases and implementing a stable staging environment.

The key trends in test automation tools cantered on open source and continuous testing. While those will remain prominent in the coming year, several broad technology trends will affect testing and testing tools.

A defining factor for successfully applying test automation in software projects is choosing and using the right set of test automation tools. This is a daunting task, especially for those new to software test automation because there are so many tools in the market to choose from, each having different strengths and weaknesses. There is no tool that can fit all automated testing needs which makes finding the right tool difficult. Learn how to identify the right automation tool for your project with this qualitative comparison of automated testing toolsets in the market.

3.1OVERVIEW OF TOOLS

Katalon Studio is an automated testing platform that offers a comprehensive set of features to implement full automated testing solutions for Web, API, and Mobile. Built on top of the open-source Selenium and Appium frameworks, Katalon Studio allows teams to get started with test automation quickly by reducing the effort and expertise required for learning and integrating these frameworks for automated testing needs.

Selenium is perhaps the most popular automation framework that consists of many tools and plugins for Web application testing. Selenium is known for its powerful capability to support performance testing of Web applications. Selenium is a popular choice in the open-source test automation space, partly due to its large and active development and user community.

Unified Functional Testing (UFT), formerly QuickTest Professional (QTP), is probably the most popular commercial tool for functional test automation. UFT offers a comprehensive set of features that can cover most functional automated testing needs on the desktop, mobile and Web platforms.

TestComplete is also a commercial integrated platform for desktop, mobile and Web application testing. Like UFT, TestComplete offers a number of key test automation features such as keyword-driven and data-driven testing, cross-browser testing, API testing and CI integrations. This tool supports a number of languages including JavaScript, Python, VBScript, JScript, DelphiScript, C++Script, and C#Script for writing test scripts.

3.2COMPARISON OF TOOLS

The table below provides a comparison of the tools based on the key features of software automation:

Features	**Katalon Studio**	**Selenium**	**UFT**	**TestComplete**
Test development platform	Cross-platform	Cross-platform	Windows	Windows
Application under test	Web, Mobile apps, API/Web services	Web apps	Windows desktop, Web, Mobile apps, API/Web services	Windows desktop, Web, Mobile apps, API/Web services
Scripting languages	Java/Groovy	Java, C#, Perl, Python, JavaScript, Ruby, PHP	VBScript	JavaScript, Python, VBScript, JScript, Delphi, C++ and C#
Programming skills	Not required. Recommended for advanced test scripts	Advanced skills needed to integrate various tools	Not required. Recommended for advanced test scripts	Not required. Recommended for advanced test scripts
Learning curves	Medium	High	Medium	Medium

Ease of installation and use	Easy to setup and run	Require installing and integrating various tools	Easy to setup and run	Easy to setup and run
Script creation time	Quick	Slow	Quick	Quick
Object storage and maintenance	Built-in object repository, XPath, object re-identification	XPath, UI Maps	Built-in object repository, smart object detection and correction	Built-in object repository, detecting common objects
Image-based testing	Built-in support	Require installing additional libraries	Built-in support, image-based object recognition	Built-in support
DevOps/ALM integrations	Many	No (require additional libraries)	Many	Many
Continuous integrations	Popular CI tools (e.g. Jenkins, Team city)	Various CI tools (e.g. Jenkins, Cruise Control)	Various CI tools (e.g. Jenkins, HP Quality Center)	Various CI tools (e.g. Jenkins, HP Quality Center)
Test Analytics	Katalon Analytics	No	No	No
Product support	Community, Business support service, Dedicated staff	Open source community	Dedicated staff, Community	Dedicated staff, Community
License type	Freeware	Open source	Proprietary	
Cost	Free	Free	License and maintenance fees	License and maintenance fees

3.3Strengths and weaknesses

The comparison table above mainly focus on the common features of an automated testing tool. The following presents another perspective by picking and comparing key strengths and limitations of the tools.

- Tools,Strength &Limitations:

❖ Katalon Studio

No licensing and maintenance fees required (paid dedicated support services are available if needed).

Integrating necessary frameworks and features for quick test cases creation and execution.

Built on top of the Selenium framework but eliminating the need for advanced programming skills required for Selenium. Emerging solution with a quickly growing community. Feature set is still evolving. Lack of choices for scripting languages: only Java/Groovy is supported.

❖ Selenium

Open source, no licensing and maintenance fees. Large and active development and user community to keep pace with software capability
Testing teams need to have good programming skills and experience to set up and integrate Selenium with other tools and frameworks.

New teams need to invest time upfront for setup and integration. Slow support from the community.

❖ UFT

Mature, comprehensive automated testing features integrated into a single

system. Dedicated user support plus an established large user community.

There is no one-size-fits-all tool for automated testing. It is highly recommended that testers evaluate
various tools in order to select what would best meet their automated testing needs. Programming languages and technologies used to develop software continue to evolve, as do the automated testing tools, making cost a significant factor in tool selection. Commercial vendors often charge for tool upgrades, which can be substantial if your software uses emerging and frequently changing technologies. Open source and non-commercial tools, on the other hand, do not incur additional charges but require effort and expertise for integrating new upgrades. It is difficult to find the support and expertise needed for integrating various tools and frameworks into open-source solutions. Emerging tools that integrate with open-source frameworks, like Katalon, offer a viable alternative to both commercial and

open-source automated testing solutions.

- ❖ Katalon Studio

No licensing and maintenance fees required (paid dedicated support services are available if needed).

Integrating necessary frameworks and features for quick test cases creation and execution.

Built on top of the Selenium framework but eliminating the need for advanced programming skills required for Selenium. Emerging solution with a quickly growing community. Feature set is still evolving. Lack of choices for scripting languages: only Java/Groovy is supported.

3.4 Maintainability comparing:

Compare	Tosca	HPE QTP/UFT	Selenium	Worksoft
Automation type	Model-based	Script-based	Code-based	Object-driven
Scripting required	No	Yes, VB Script	Yes, multiple	Yes, Business-level
Supported techs	> 150	> 35	1	10
Ease of adoption	Very high	Average	Low	Average
Ease of maintenance	Very high	Average	Low	Average
Reusability of test artifacts and data	Very high	Average	Low	Average

3.5 Testing Capabilities

Compare	Tosca	HPE QTP/UFT	Selenium	Worksoft
API testing	Yes	HP UFT	No	No
Cross-browser testing	Yes	Yes	Yes	Yes
Mobile testing	Yes	Yes	Yes	Requires 3rd-party tool
SAP testing	Yes	Yes	No	Yes
SAP test data management	Yes	No	No	No

SAP impact analysis	Yes	No	No	Yes
BI/DWH testing	Yes	No	No	No
End-to-end testing	Yes	Yes	No	No
Risk-based testing	Yes	Requires HP QC/ALM	No	Yes
Test case design	Yes	No	No	No
Test case optimization	Yes	No	No	No
Test data management	Yes	No	No	Yes
Service virtualization	Yes	Yes	No	No

3.6 Deployment and Adoption

Compare	**Tosca**	**HPE QTP/UFT**	**Selenium**	**Worksoft**
Dedicated support	Yes	Yes	No	Yes
On-premise solution	Yes	Yes	Yes	Yes
SaaS platform	Yes	Requires HP QC/ALM	Yes	No
Distributed execution	Yes	Requires HP QC/ALM	Yes	Yes

3.7 Integrations

Compare	Tosca	HPE QTP/UFT	Selenium	Worksoft
CI integration	Any CI tool	Jenkins	Jenkins	Jenkins
ALM/DevOps integrations	> 20 tools	> 20 tools	No	4 ALM tools
SoapUI integrations	Yes	No	No	No
HPE QC/UFT	Yes	N/A	No	No
Selenium integration	Yes	No	N/A	No
Worksoft integration	Yes	No	No	N/A
SAP Solution Manager	Yes	Yes	No	Yes

4.0 System Use cases

4.1 Add TC

Use case: Add new test case

ID:1

Brief description: Adding new test case to repository

Main Actor: The tester

Secondary Actor: Not exist

Pre-condition: choose on of exist requirement

Main flow:
The process starts when:
The tester clicks on add test case button
The below interface appears to tester:

- Box for adding the title
- Box for the tester which test case assign to him
- Drop down list for select the state
- Drop down list for select the Priority
- Drop down list for select the Severity
- Button to add new step by adding attribute:
    - Drop down list to select The Intended page.
    - Drop down list to select The Intended object
    - Drop down list to select The Intended action
- The Added steps
- Button to add the test case

Post-condition: store the new test case in database

4.2 Add requirement

Use case: Add requirement
ID:2
Brief description: Adding new requirement to repository
Main Actor: The tester
Secondary Actor: Not exist
Pre-condition: Not exist
Main flow: The process starts when: The user clicks on add requirement button The below interface appears to user: • Box for adding title to the requirement • Button for open pop interface to adding the content
Post-condition: store the requirement in database

4.3 Create Defect

Use case: create defect
ID:3
Brief description: create defect for failed test case
Main Actor: The tester
Secondary Actor: Not exist
Pre-condition: Run all test cases
Main flow: The process starts when: The tester clicks on create defect button for failed test case The below interface appears to tester: • Test case name • Box for adding title to the created defect • Drop down list for select the state which will be active when creating • Drop down list for adding the Priority • Drop down list for select the Severity • The Error messages • Button for saving the action
Post-condition: store created defect in data base

4.4 Automatic Run

Use case: run test cases
ID:4
Brief description: Run all test cases in one project
Main Actor: The tester
Secondary Actor: Not exist
Pre-condition: choose on of exist project
Main flow: The process starts when: The user clicks on Run test cases button The below interface appears to user: • the running process through chrome driver • running results regarding the failed and passed test cases • button for create defect for failed test case
Post-condition: store the result for all running test case

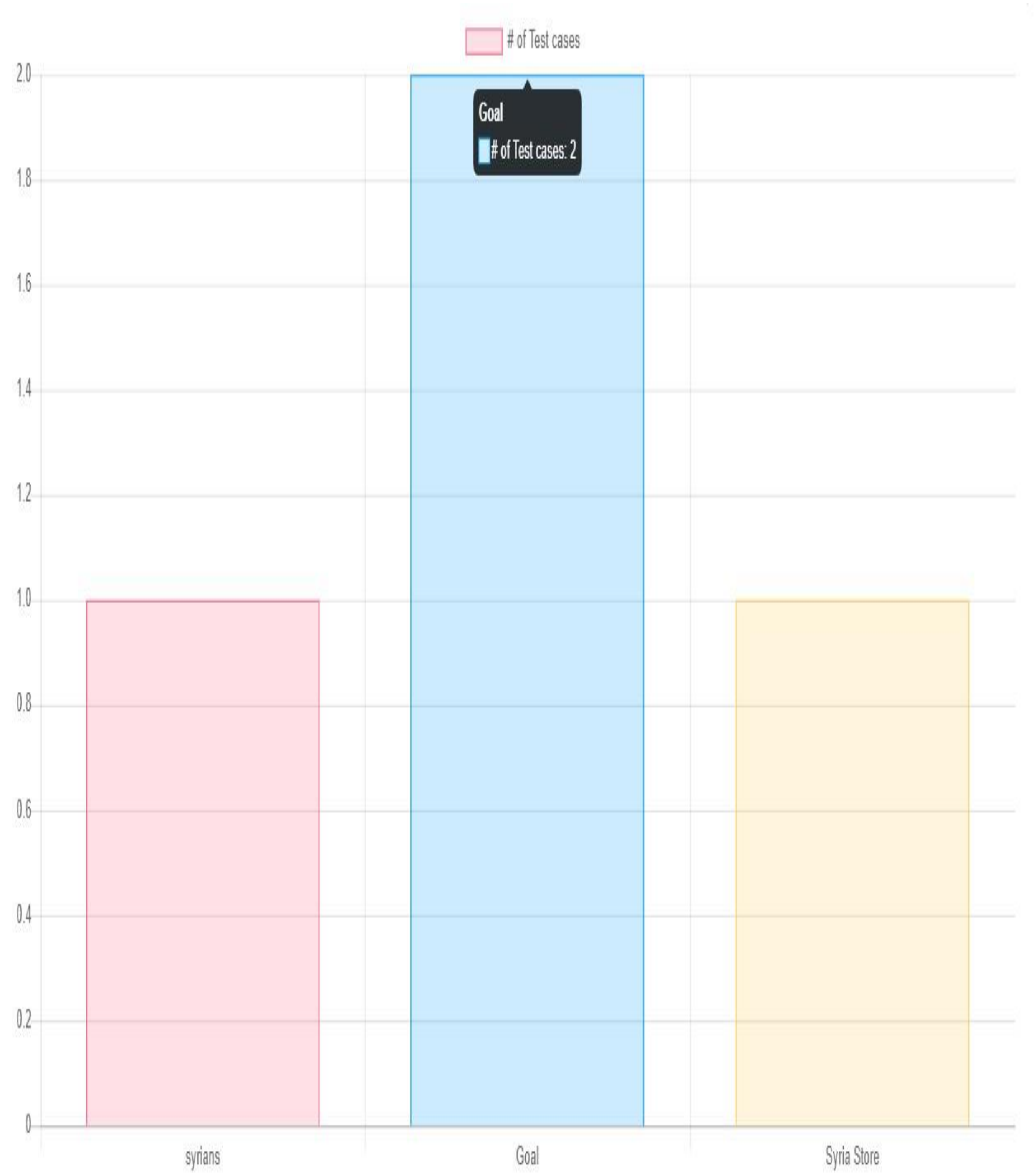
4.5 Add new Project

Use case: Add Project
ID:5
Brief description: Adding new project to repository
Main Actor: The tester
Secondary Actor: Not exist
Pre-condition: Not exist
Main flow: The process starts when: The user clicks on add project button The below interface appears to user: • Box for adding title for the project
Post-condition: store the project in database

5.0 Software Interfaces (functional Requirements)

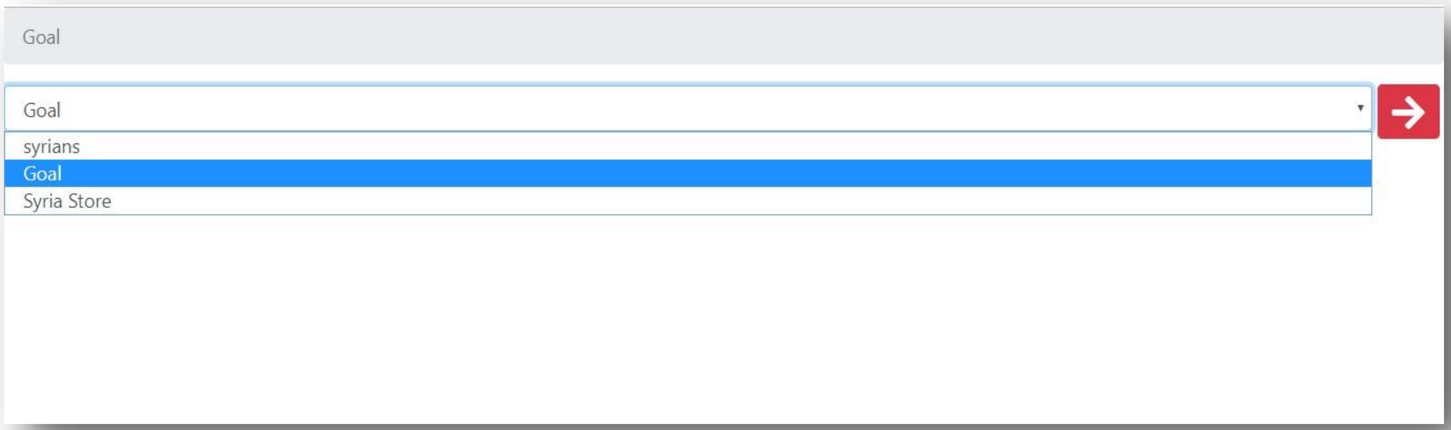
5.1 Welcome page

Welcome page is only view page and it will show a statistics chart of each project and its related testcases count



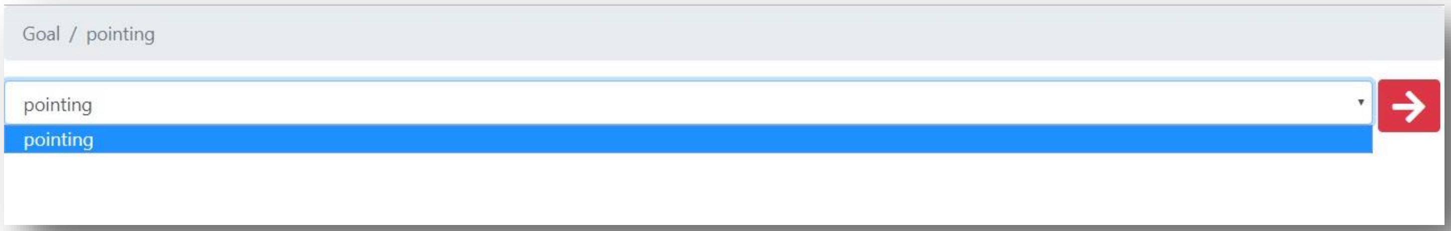
5.2 Select project page

In the next page, a list of all open projects will appear and the tester must select one to continue



5.3 Select Requirement

As we explained previously in the system hierarchically, each project node will have requirements as child nodes, so in the next page the tester must select a specific requirement



Noting that in each page, we used the “Bread Crumbs” in header design in order to mapping the current
page.

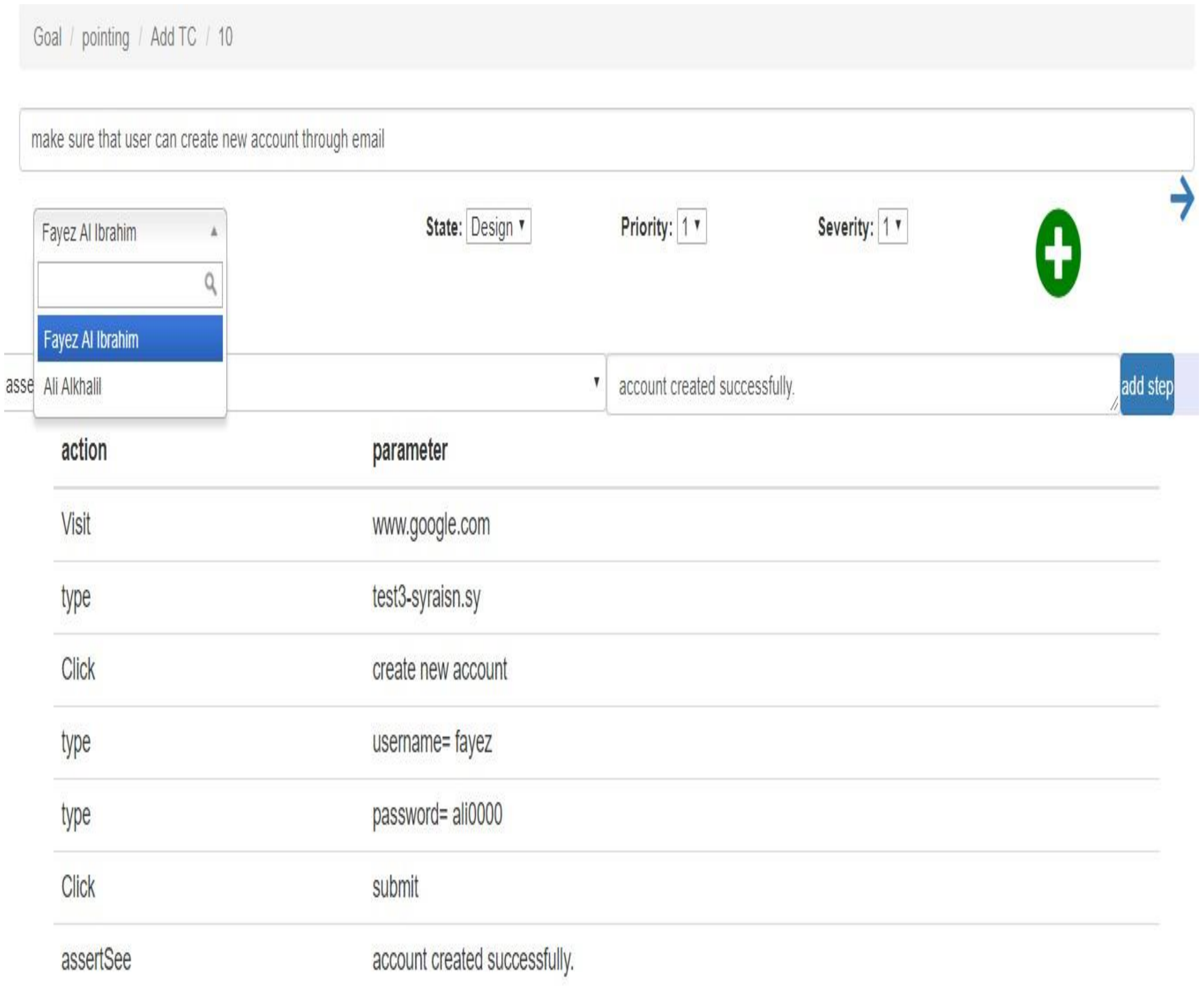
5.4 create testcase page

Next, the tester will create all the needed testcases related to the selected requirement, creating test case will be done by specific criteria, as the tester must first enter test case title to initiate the test case.

Then he must enter each step considering the action and the parameter of the step.

In this page header, a road map from the project to the requirement then to create test case and the new id.

Besides the title and the test case steps, the tester must enter the priority, severity and the assigned to value from a search input inside the dropdown list.



Goal / pointing / Add TC / 10

make sure that user can create new account through email

Fayez Al Ibrahim

Fayez Al Ibrahim

Ali Alkhalil

State: Design

Priority: 1

Severity: 1

asse

account created successfully.

action	parameter
Visit	www.google.com
type	test3-syraisn.sy
Click	create new account
type	username= fayez
type	password= ali0000
Click	submit
assertSee	account created successfully.

5.5 Run and show results page

After making sure that we covered all test cases, the next page will be run page, where chrome driver will start executing all the test cases of the selects project, then show messages error of each failed test case.

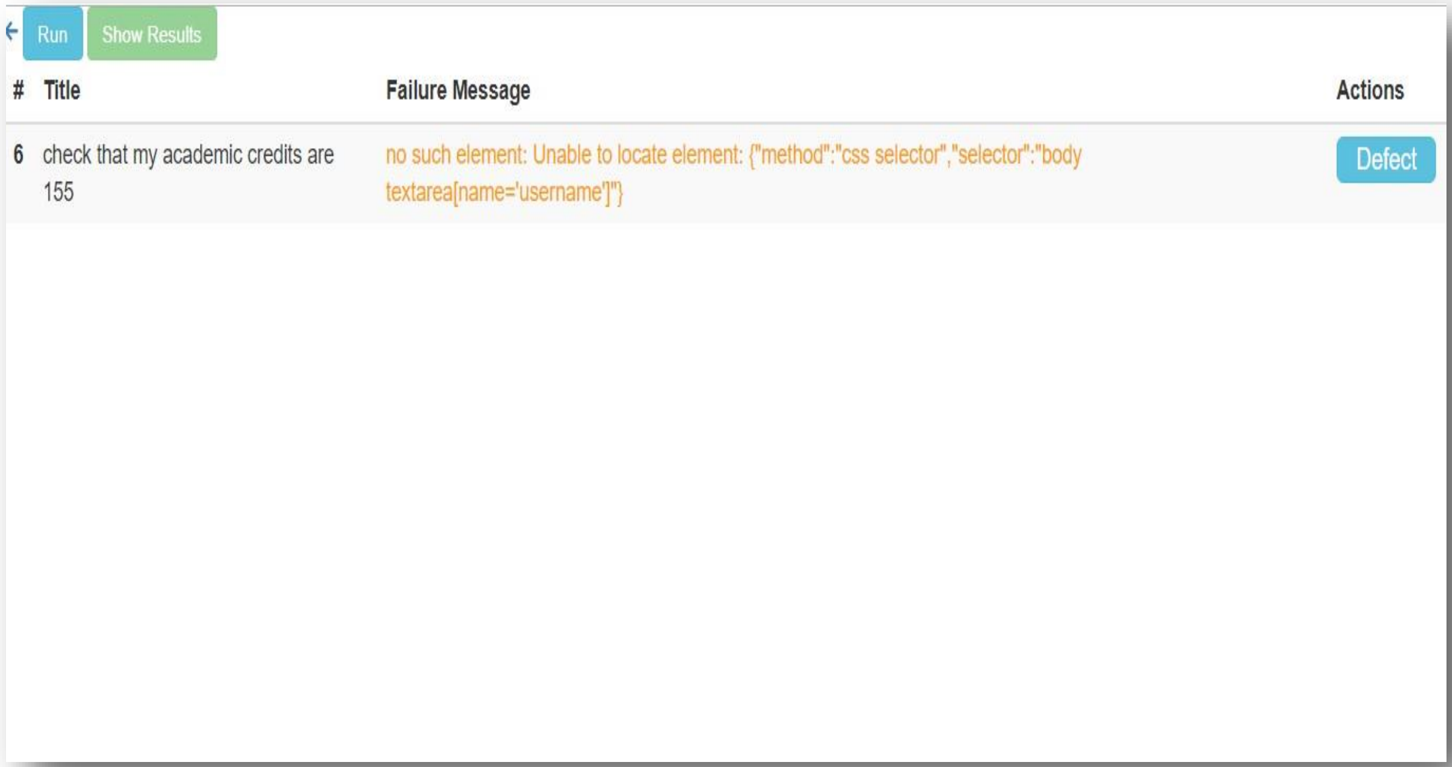
In the below example, the test case failed if any of its steps failed, and in this case the failure was due
unknown object, the user name object was not found in the specific page so it’s failed

For each failed test case, a defect button will be found in each row and after click on it a modal will appear in order to create the defect or the bug.

In creating bug page, the system will show the error message by default then the tester must enter the basic needed data which is:

- Bug title
- Priority & Severity
- Optional description
- Assign to value

Then by clicking on save button, the bug will be created and assigned successfully.



#	Title	Failure Message	Actions
6	check that my academic credits are 155	no such element: Unable to locate element: {"method":"css selector","selector":"body textarea[name='username']"}	Defect

6.0 future Release

As we tried to apply the basic and main features in this release, as creating test cases and execute them manually and automatically.

In the next release the system will contain the below extra features:

- Responsivity test
- Manage authorization of testers
- Detailed reports
- Add more actions
- Multi-threading execution
- Improve system UIs
- Add admin role

7.0 Applied Actions

Regarding actions in create test case in the automation test, currently we apply the below basic action which can be used by the tester:

1. Visit:
2. press
3. type
4. assert See
5. Click
6. Click Link
7. Mouse Over
8. pause
9. wait For
10. wait For Text
11. assert Path Is

8.0 Assertions

Visit	`//to navigate to specific URL`
	
	`$browser->visit('/login')`

press	`->press('Login')`
type	`->type('email', $user->email)`
Click	`$browser->click('.login-page .container div > button');`

clickLink	`//The method will click the link that has the given display text:`
	
	`$browser->clickLink($linkText);`

mouseOver	`$browser->mouseover('.selector');`
pause	`$browser->pause(1000);`

waitFor	`// Wait a maximum of five seconds for the selector...`
	
	`$browser->waitFor('.selector');`
	
	`// Wait a maximum of one second for the selector...`
	
	`$browser->waitFor('.selector', 1);`

waitForText	`// Wait a maximum of five seconds for the text...`
	
	`$browser->waitForText('Hello World');`

	`// Wait a maximum of one second for the text...`
	
	`$browser->waitForText('Hello World', 1);`

assertPathIs	`//Assert that the current path matches the given path:`
	`$browser->assertPathIs('/home')`

assertSee	`//Assert that the given text is present on the page:`
	
	`$browser->click('.some-action')`
	`->assertSee('something');`

assertDontSee	`$browser->assertDontSee($text);`

assertTitle	`//Assert that the page title matches the given text:`
	
	`$browser->assertTitle($title);`

assertTitleContains	`//Assert that the page title contains the given text:`
	
	`$browser->assertTitleContains($title);`

assertPathBeginsWith	//Assert that the current URL path begins with the given path:
	
	`$browser->assertPathBeginsWith($path);`

assertPathIsNot	`//Assert that the current path does not match the given path:`
	
	`$browser->assertPathIsNot('/home');`

assertSeeIn	`//Assert that the given text is present within the selector:`

	`$browser->assertSeeIn($selector, $text);`

assertDontSeeIn	`//Assert that the given text is not present within the selector:`
	
	`$browser->assertDontSeeIn($selector, $text);`

assertSeeLink	`//Assert that the given link is present on the page:`
	
	`$browser->assertSeeLink($linkText);`

assertDontSeeLink	`//Assert that the given link is not present on the page:`
	
	`$browser->assertDontSeeLink($linkText);`

assertInputValue	`//Assert that the given input field has the given value:`
	
	`$browser->assertInputValue($field, $value);`

assertInputValueIsNot	`//Assert that the given input field does not have the given value:`
	
	`$browser->assertInputValueIsNot($field, $value);`

assertChecked	`//Assert that the given checkbox is checked:`
	
	`$browser->assertChecked($field);`

assertNotChecked	`//Assert that the given checkbox is not checked:`

	`$browser->assertNotChecked($field);`
assertRadioSelected	`//Assert that the given radio field is selected:` `$browser->assertRadioSelected($field, $value);`
assertRadioNotSelected	`//Assert that the given radio field is not selected:` `$browser->assertRadioNotSelected($field, $value);`
assertSelected	`//Assert that the given dropdown has the given value selected:` `$browser->assertSelected($field, $value);`
assertNotSelected	`//Assert that the given dropdown does not have the given value selected:` `$browser->assertNotSelected($field, $value);`
assertSelectHasOptions	`//Assert that the given array of values are available to be selected:` `$browser->assertSelectHasOptions($field, $values);`
assertSelectHasOption	`//Assert that the given array of values are available to be selected:` `$browser->assertSelectHasOptions($field, $values);`

assertVisible	`//Assert that the element matching the given selector is visible:`
	
	`$browser->assertVisible($selector);`

assertMissing	`//Assert that the element matching the given selector is not visible:`
	
	`$browser->assertMissing($selector);`

9.0 Non-functional Requirements

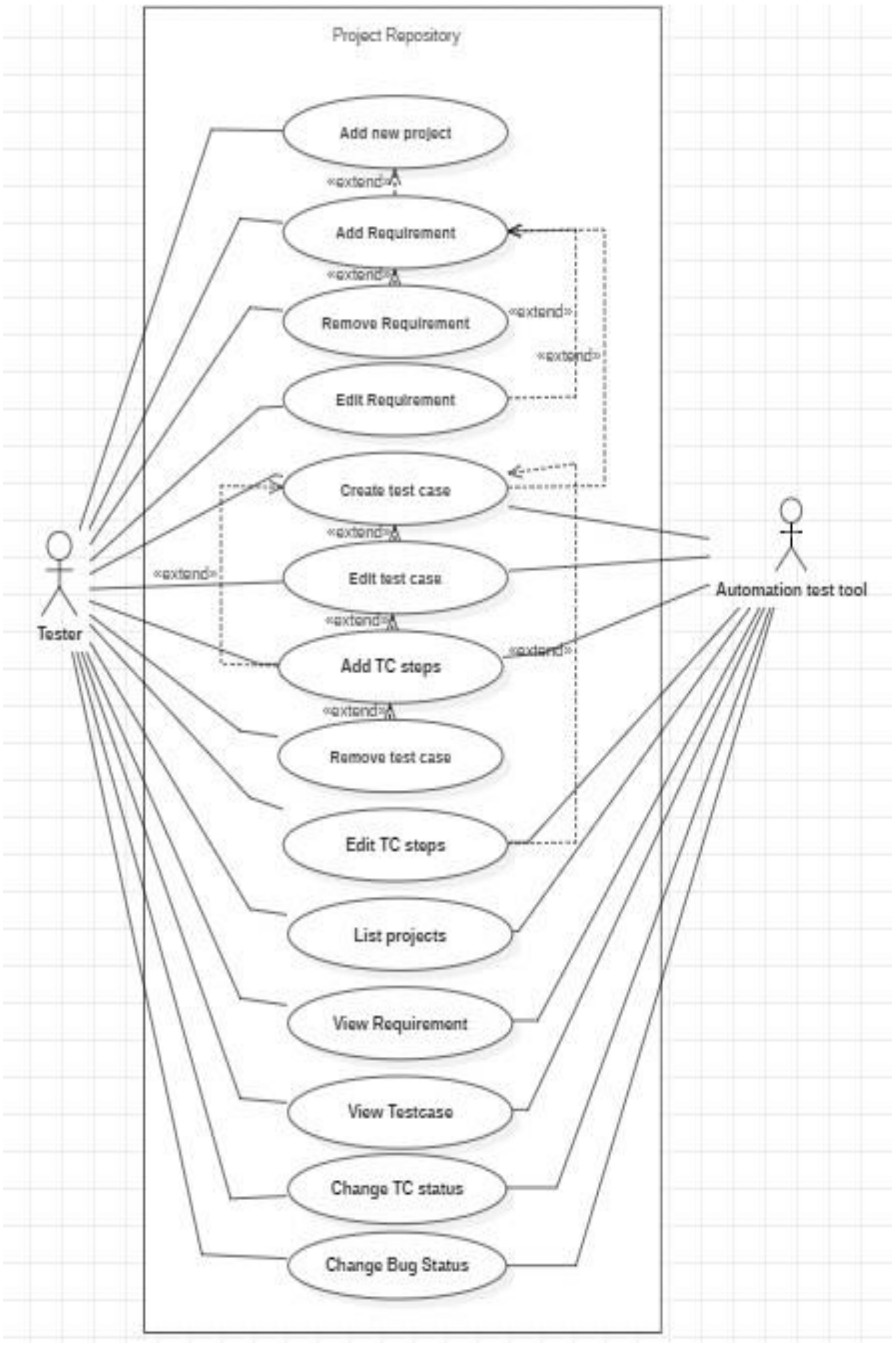
functional: average test case execution will be no longer that 1.50 – 1.40 seconds, system performance and the amount of test accomplished by the testing tool. performance is estimated in terms of accuracy, efficiency and speed of executing steps.

Reliability: the system is required to work without any failure as long as the servers are on.

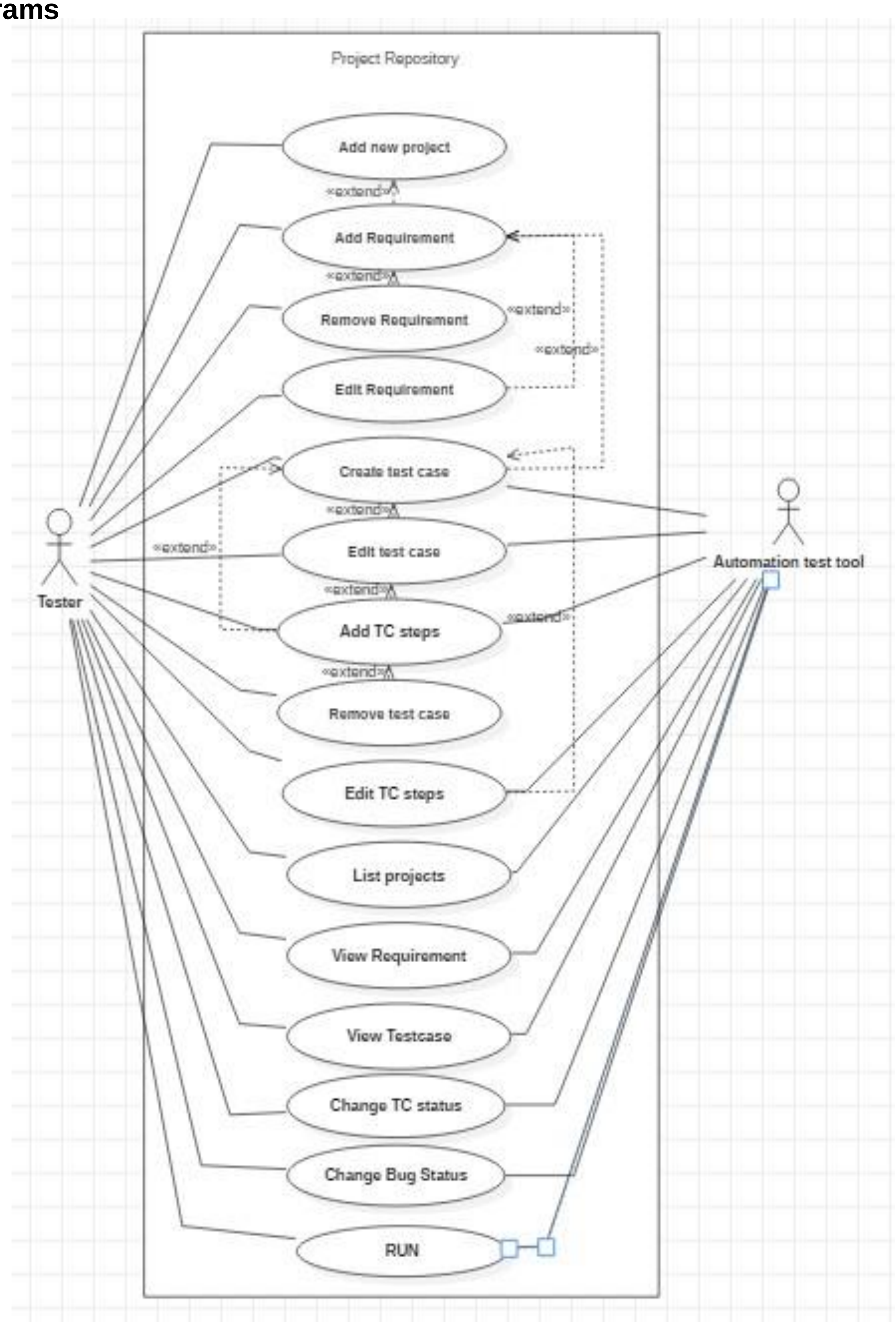
Response time: showing testing results with the error messaged will not take longer than 4-5 seconds

Backup: a database backup will occur each 15 days

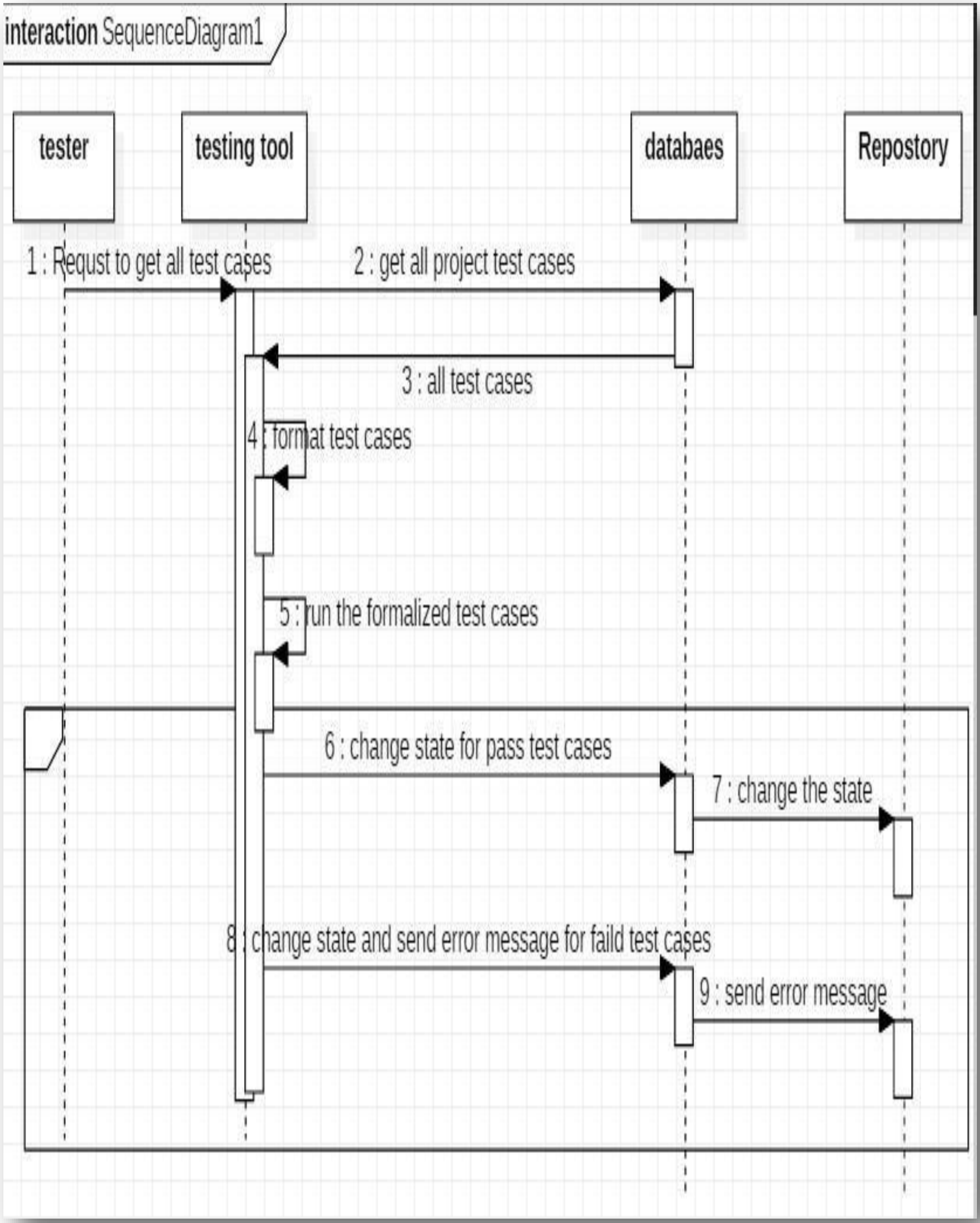
10.0 Diagrams



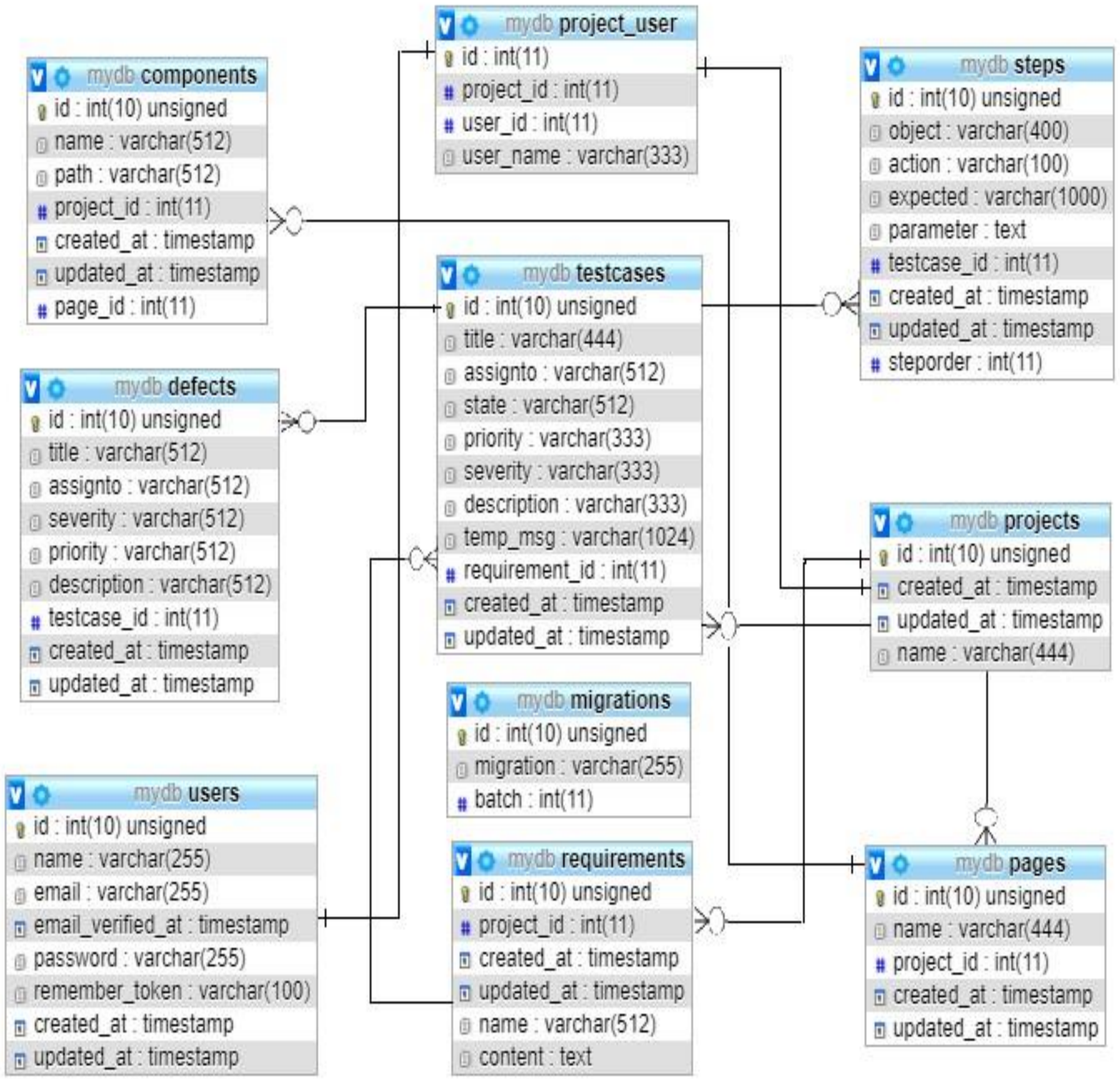
10.1 Diagrams



10.2 Diagrams:


interaction SequenceDiagram1
tester
testing tool
databaes
Repostory
1 : Requst to get all test cases
2 : get all project test cases
3 : all test cases
4 : format test cases
5 : run the formalized test cases
6 : change state for pass test cases
7 : change the state
8 : change state and send error message for faild test cases
9 : send error message

10.3 Diagrams:



11.0 Conclusion:

Automation testing tools helps the tester to easily automate the whole testing process. Automation testing improves the accuracy and also save time of the tester as compared to the manual testing. The main ideas in this reference study refer to:

- ❖ The importance of developing the techniques used in software engineering.
- ❖ The global trend towards self-testing of sites and applications in general
- ❖ Automatic testing systems capable of raising the organizational level.
- ❖ The presence of large numbers of developers and programming companies who use these platforms is clearly evidence of a tangible benefit from them.
- ❖ Automated automation systems exceeded the problem of space and time.

Traditionally, automated and manual testing are considered as different and separate approaches are used for their execution. In reality, they depend on each other, as the limitation of one is addressed by the other one. Manual testing can be useful for finding bugs in special cases where the requirement changes continuously, and situations where automated tests might not be the most effective. However, test automation has many advantages such as repeatability, consistency, better and effective handling of test cases (for situations where a large number of test cases need to be executed). The core difference between manual and automation testing is that test automation is most appropriate for the situation where repetitive work needs to be done (re- testing with same or different test data but the same test script). But it cannot eliminate all of the bugs in the application without the help of manual testing. So, automated tests are good at breadth but not much at depth [47]. In this paper, the analysis and comparison are made among different automated testing tools (Selenium IDE, Selenium Web driver, WATIR and UFT/QTP) on various quality factors. After the overall analysis, it is not easy to rank these tools based on a number of factors only. Selenium provides the freedom to work with all types and almost all the versions of browsers, operating system as well as flexibility to choose one among many programming languages. However, its access is limited to web applications only. Another automation tool WATIR has become also popular nowadays. It also supports almost all the browsers, but it lacks record and playback functionality, which could be a great functionality to get started with for beginners. Similar to Selenium, it also does not provide testing for windows applications. Finally, when it comes to UFT/QTP, it works well with both web and windows applications. It has a built-in mechanism of object identification as well as options to work with different add-ins. It also integrates with 83 Application Life cycle Management (ALM) tools (effective in managing several phases of SDLC). On the other hand, QTP is not compatible with latest versions of browsers and OS, although UFT supports all the latest versions of Internet Explorer and most of the Chrome and some of Firefox browsers as well as it performs API testing. The biggest limitation with UFT/ QTP is licensing cost, which is too high. Finally, all these tools have advantages, limitations as well as utilization for some certain types of testing, based on the scope of applications. None of these tools is absolutely perfect or best. Although, in recent years, Selenium has been preferred by most software industries and is used more than other tools by testers or developers. Selenium's variety of features and functionalities, ability to integrate with various frameworks, its number of different programming languages, and its free cost provides major advantages. However, not all the applications are web-based. For that reason, there will always be demand for UFT or tools which are able to perform testing in both web and windows-based applications. Moreover, WATIR has been gradually overcoming its limitations and gaining popularity within the industries. While concluding this research, I learned that, each software testing tool has its own distinguishing features and it’s a matter of time to learn each different tool and know how to utilize the tools for automation testing. It also takes a lot effort to find out the tool that works best to meet the goal of testing. From my own perspective, an ideal tool should meet at least some criteria. First, the installation process should be simple and quick. Second, getting started with the tool for novices should not be too difficult and there should be adequate learning material available for the tool, which helps to create and execute a basic set of test scripts. Finally, the tool should be friendly to work with as well as it should be able to generate an overall test result 84 (pass/fail). So, if the testing fails, it makes easily understandable logs to troubleshoot, which saves a lot of time. For future work, my goal is to extend this research by including a few more testing tools to analyze and compare. Then the selected tools should have latest features.

12.0 References

- IEEE Software Engineering Standards Committee, “IEEE Std 830-1998, IEEE Recommended
  Practice for Software Requirements Specifications”.
- https://laravel.com/docs/5.7/dusk
- https://www.seleniumhq.org/
- https://phpunit.de/
- https://en.wikipedia.org/wiki/Software_testing
- https://en.wikipedia.org/wiki/Test_automation
- https://en.wikipedia.org/wiki/HP_QuickTest_Professional
- https://laravel.com/
- https://www.ieee.org/
- https://cucumber.io/
- http://www.mit.edu/
- Richa Rattan, “Comparative Study of Automation Testing Tools: Quick Test Professional &
  Selenium”, Vsrd International Journal of Computer Science & Information Technology, Vol. 3 No. 6 June 2013.
- Manjit Kaur, Raj Kumar, “Comparative Study of Automated Testing Tools: Test Complete and Quick Tester”, International Journal of Computer Applications (0975-8887) Volume 24-No. 1, June 2011.
- Software Test Automation - Http://En.Wikipedia.Org/Wiki/Test_Automation
- Manual Testing Http://En.Wikipedia.Org/Wiki/Manual_Testing
- Leckraj Nagowah And Purmanand Roopnah, “Ast -A Simp Le Automated System Testing Tool”,
  Ieee, 978-1- 4244-5540-9/10, 2010
- S Thummalapenta, S Sinha, N Singhania, S Chandra, “Automating Test Automation”,
  Proceedings of the 34th International Conference on Software Engineering (Icse), May 2012
- Test Automation: Http://Researcher.Watson.Ibm.Com/Researcher/View_Group.Php?Id=3208
- Zhi Quan Zhou, Bernhard Scholz, Giovanni Denaro, “Automated Software Testing and Analysis: Techniques, Practices and Tools”, Proceedings of the 40th Annual Hawaii International Conference on System Sciences (Hicss'07) 0-7695-2755-8/07
- X. Wang and G. He, “The research of data-driven testing based on QTP,” Computer Science

Education (ICCSE), 2014 9th International Conference on. pp. 1063–1066, 2014.

- M. Monier and M. M. El-mahdy, “Evaluation of automated web testing tools,” vol. 4, no. 5, pp. 405–408, 2015.
- G. Saini, “Software Testing Techniques for Test Cases Generation,” vol. 3, no. 9, pp. 261–265.
- R. Gupta, Test Automation and QTP: QTP 9.2, QTP 9.5, QTP 10.0 and Functional Test 11.0. Pearson Education.