

# **SOFTWARE ENGINEER TRAINING REPORT**

SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE AWARD OF

**DEGREE OF BACHELOR OF TECHNOLOGY IN COMPUTER  
SCIENCE ENGINEERING**



**Submitted By**

**Submitted To: - Er Ashwani Kumar**

**Name: Priyanka Mehta**

**University Roll No 19117**

**HEAD OF DEPARTMENT- Ms. NEHA SINGLA**  
**Department of Computer Science & Engineering**

*DRONACHARYA COLLEGE OF ENGINEERING,  
KHENTAWAS, GURGAON, HARYANA*



# **TRAINING REPORT**

## **SOFTWARE ENGINEER**

Submitted in partial fulfillment of the  
Requirements for the award of

**Degree of Bachelor of Technology in Computer Science Engineering**



**Submitted By**  
**Name: Priyanka Mehta**  
**University Roll No 19117**

**Submitted To: -**  
**Er. Ashwani Kumar**



## STUDENT DECLARATION

I hereby declare that the Industrial Training Report entitled ("**SOFTWARE ENGINEER**") is an authentic record of my own work as requirements of 6-months Industrial Training during the period from **February 17, 2020** to **August 17, 2020** for the award of degree of B.Tech (Computer Science & Engineering), DCE, under the guidance of Mr. Abhimanyu kumar.

(Signature of Student)  
Priyanka  
University roll no  
19117

Date: \_\_\_\_\_

Certified that the above statement made by the student is correct to the best of our knowledge and belief.

**Signatures**

**Examined by:**

1.

2.

3.

**Head of Department**  
(Signature and Seal)



**Date: 12<sup>th</sup> February, 2020**

**To,  
Ms. Priyanka Mehta,  
MDU**

**Subject: Offer Letter**

**Dear Priyanka,**

We are pleased to appoint you as a **Software Engineer- Trainee** in our organization on the following terms and conditions:

1. You will be getting INR **10,598** for 6 months training period.
2. You are required to join on **17<sup>th</sup> February, 2020** unless the date is extended by us and communicated to you in writing.
3. You will be based at our **Gurugram** office in India. You should be prepared to work anywhere in India or abroad without claiming any extra remuneration for such transfers. The Company reserves the right to transfer you to any office, department or establishment forming a part of our Company or any establishment wherever our company will be having interest.
4. In case of further clarifications, please communicate with the HR department, and quote the reference as above.
5. Without prejudice, please note that company reserves the right to withdraw this offer made to you, before receipt of your acceptance of the same, without providing any reasons to you.
6. If on verification, at the time of appointment or at a later date it is found that you have furnished wrong information, in such a case, your services with the company will be liable to termination.
7. After successful completion of 6 months training, CTC would be between 2.7 LPA – 3 LPA depending upon the performance during the training period.

We welcome you to Kellton Tech Solutions Limited and look forward to a long and mutually beneficial association.

**For Kellton Tech Solutions Limited**

**Authorized Signatory  
(Megha Thakur)  
Manager – Human Resources**

**Annexure**

**CHECK LIST OF DOCUMENTS**

At the time of joining, you are requested to submit the copies of the following documents:

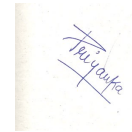
1. Bank Guarantee documents.
2. Certificates supporting your educational and professional qualifications along with marks sheets  
(10+12+ Graduation + Post Graduation + Course Certifications)
3. Three (3) Passport Size color photographs of a recent date.
4. Photo Copy of Pan Card.
5. Valid Passport, Aadhar Card and/or Election Commission Card.

**DECLARATION**

This is to certify that I have read and understood all the above terms and conditions mentioned in Annexure and I hereby accept and agree to abide by them. I will be reporting for duty on **17<sup>th</sup> February, 2020.**

Date:

Signature:



**(Priyanka Mehta)**



## ACKNOWLEDGMENT

It is my proud privilege to express my profound gratitude to the entire management of Dronacharya College of Engineering for providing me with the opportunity to avail the excellent facilities and infrastructure. The knowledge and values inculcated have proved to be of immense help at the very start of my career.

I am grateful to **Ms. Neha Singla** for her astute guidance, constant encouragement and sincere support for this project work. I am thankful to all the teaching and non-teaching staff members.

I would like to thank **Dronacharya College of Engineering** for providing me with the opportunity to pursue my industrial training, as it is an important part of the Btech. course and it is the one that exposes you to the industry standards and makes you adapt yourself to the latest trends and technologies. At the same time, it gives an experience of working on a project.

I feel pride and privileged in expressing my deep sense of gratitude to all those who have helped me in presenting this assignment. I would be failing in my endeavor if I do not place my acknowledgement.

I express my sincere gratitude to **All Teachers** for their inspiration, constructive suggestion, mastermind analysis and affectionate guidance in my work, without which this project work completion would have been impossible for me.

**Priyanka Mehta**  
**University roll no**

# COMPANY PROFILE

## KELLTON TECH



Continuing the quest for infinite possibilities with technology is Kellton Tech, **a pioneer of digital transformation.** We believe in disruptive innovation to deliver infinite technology solutions to every organization that we work with.

Over the years, we have grown by leaps and bounds and now have over 200 clients ranging from start-ups to early stage enterprises and Fortune 1000 companies. Today, we are a global force in digital transformation and enterprise solutions, offering end-to-end IT solutions, strategic technology consulting and product development services in Social, Mobile, Analytics, Cloud, S/4HANA, Mule Soft, SAP, IoT and Artificial Intelligence.

Empowering 30% of the \$ 21 bun dollar ecommerce market

We enable our clients to deliver a sophisticated and secured experience for online shopping. We enable enterprises to unlock their potential and take the digital leap. The company is listed on India's largest stock exchange BSE and NSE [Code: KELLTONTEC; BOM: 519602] and is ISO 9001:2015 and CMMI Level 5 certified. Our methodology of inventing infinite possibilities with technology helps us develop best in-class and cost effective solutions for our clients. Using our flagship product, KLGAME™ (Kellton Tech Location based Gamification, Analytics and rich Messaging Engine), we have helped clients across the globe think disruptively about their business in a cost effective manner.

Kellton Tech is a Microsoft Certified Gold Partner and hosts Amazon, MS Azure professionals. We have served clients coming from a wide range of verticals including retail, travel, e-commerce,

education, hospitality, advertising, market research, manufacturing, consumer goods, logistics, SCM, lifestyle, non-profits, banking, financial services and insurance,.

The company has offices in India (Gurgaon, Hyderabad, Lucknow), USA (Cupertino, CA; McLean, VA; New Brunswick, NJ, Chicago) and Europe (Ireland, United Kingdom) and employed over 1400 people across the globe.



## TRAINING AT KELLTON TECH

**KELLTON TECH** works on varied technologies and hence provides students with training on different technological platforms. I have been chosen to be trained on Software Quality Assurance(Software Testing). Each trainee at KELLTON TECH has to implement a series of assignments to get hands on with the technological platform he/ She is trained on. These assignments proved to be really challenging and demands great effort from my side. Each assignment is thoroughly reviewed by a mentor . After the assignments comes the big real life scenario project, in my. I will be presenting this report on the training that I have on Software Quality Assurance.



## **Table of Contents**

<b>Chapter1: INTRODUCTION .....</b>	<b>Page no.5</b>
<b>Chapter2: BRIEF EXPLANATION OF TRAINING .....</b>	<b>Page no.7</b>
2.1 Problem Statement.....	Page no.7
2.2 Objective .....	Page no.7
2.3 Technology Used.....	Page no.9
2.3.1 Manual Testing.....	Page no.9
2. Automation Testing.....	Page no.11
3. SDLC (Software Development life cycle).....	Page no.13
4. STLC (SOFTWARE TESTING LIFE CYCLE).....	Page no.15
<b>Chapter3: TRAINING MODULE.....</b>	<b>Page</b>
<b>no.19 3.1 AGILE MODEL.....</b>	<b>Page no.19</b>
3.1.1 Test Plan for Agile.....	Page no.19
2. Agile Testing Strategies.....	Page no.19
3. Risk of Automation in Agile Process.....	Page no.22
3.2 TEST LINK .....	Page
<b>no.23 3.2.1 Test Plan.....</b>	<b>Page</b>
	<b>no.23</b>
3.2.2 Test Case .....	Page no.24
3.2.3 Test Projects .....	Page no.25
3.2.4 Test Specifications.....	Page no.25
3.2.5 Advantages of Test Link .....	Page no.25
3.2.6 Steps for creating a Test case: .....	Page no.27
<b>Chapter4: TEST MANAGEMENT TOOLS .....</b>	<b>Page no.29</b>
4.1 JIRA .....	Page no.29
4.1.1 Use of JIRA .....	Page no.29

4.1.2 For creating an Issue.....	Page no.30
4.2 Selenium .....	Page no.34
4.2.1 Components.....	Page no.34
4.2.2 Locators in Selenium .....	Page no.34
References.....	Page no.51

## **List of Figures**

Figure 1.1 Software Testing Methodology in Software Engineering .....	Page no.5
Figure 2.1 White box Testing... ..	Page no.10
Figure 2.2 Black box Testing .....	Page no.10
Figure 2.3 SDLC.....	Page no.13
Figure 2.4 STLC .....	Page no.15
Figure 3.1 Agile Testing Strategy .....	Page no.20
Figure 3.2 Technology Facing... ..	Page no.21
Figure 3.3 Test Link... ..	Page no.26
Figure 3.4 Test Cases .....	Page no.26
Figure 3.5 Step Action .....	Page no.27
Figure 4.1 JIRA Dashboard... ..	Page no.29
Figure 4.2 Create Issue.....	Page no.30
Figure 4.3 Issue Created... ..	Page no.31
Figure 4.4 Open Issue.....	Page no.31
Figure 4.5 Bug Detail.....	Page no.32
Figure 4.6 Bug Life Cycle .....	Page no.33
Figure 4.7 Locating by ID .....	Page no.35
Figure 4.8 Sign up.....	Page no.35
Figure 4.9 Locating by Name .....	Page no.36
Figure 4.10 Username .....	Page no.36
Figure 4.11 Search link text element .....	Page no.36
Figure 4.12 Find register element .....	Page no.37
Figure 4.13 Register element.....	Page no.37
Figure 4.14 Register element containing webpage.....	Page no.38
Figure 4.15 Search CSS selector element .....	Page no.39
Figure 4.16 Find CSS selector element .....	Page no.39
Figure 4.17 Search CSS selector class .....	Page no.40
Figure 4.18 Find element in CSS selector .....	Page no.40
Figure 4.19 ID element in CSS selector .....	Page no.41
Figure 4.20 Tag element in CSS selector .....	Page no.41
Figure 4.21 Last name field in inspect .....	Page no.42
Figure 4.22 Find last name element .....	Page no.42
Figure 4.23 class tag in CSS selector .....	Page no.44
Figure 4.24 Find element in CSS selector class .....	Page no.44
Figure 4.25 Password field element .....	Page no.44
Figure 4.26 Find password element .....	Page no.45
Figure 4.27 Find element in places .....	Page no.46
Figure 4.28 DOM inspect id element .....	Page no.47
Figure 4.29 Find checkbox element .....	Page no.47

Figure 4.30 Inspect username element .....	Page no.49
Figure 4.31 Find username element .....	Page no.49

# CHAPTER 1

## INTRODUCTION

Software testing is nothing but an art of investigating software to ensure that its quality under test is in line with the requirement of the client. Software testing is carried out in a systematic manner with the intent of finding defects in a **system**. It is required for evaluating the system. As the technology is advancing we see that everything is getting digitized. You can access your bank online, you can shop from the comfort of your home, and the options are endless. Have you ever wondered what would happen if these systems turn out to be defective? One small defect can cause a lot of financial loss. It is for this reason that software testing is now emerging as a very powerful field in IT.

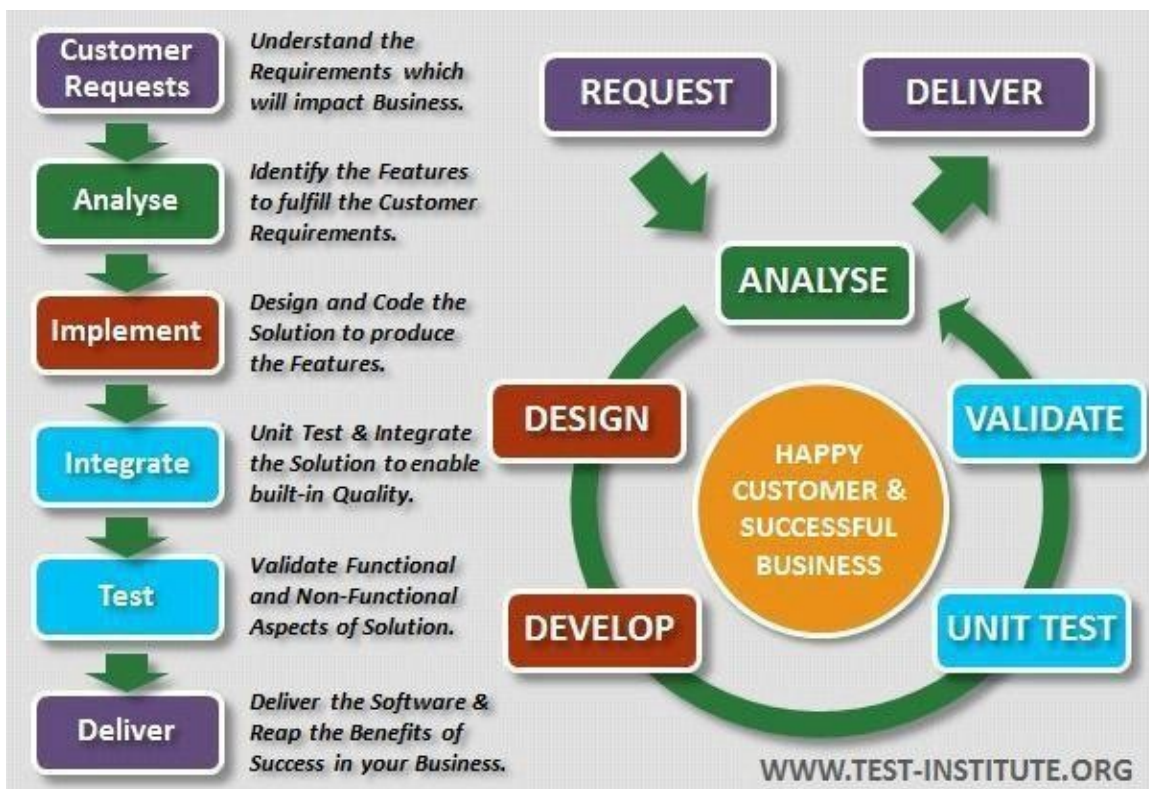


Figure 1.1 Software Testing Methodology in Software Engineering

Although like other products software never suffers from any kind of wear or tear or corrosion but yes, design errors can definitely make your life difficult if they go undetected. Regular testing ensures that the software is developed as per the requirement of the client. However, if the software is shipped with

bugs embedded in it, you never know when they can create a problem and then it will be very difficult to rectify defect because scanning hundreds and thousands of lines of code and fixing a bug is not an easy task. You never know that while fixing one bug you may introduce another bug unknowingly in the system.

Software testing is now a very significant and integral part of software development. Ideally, it is best to introduce software testing in every phase of software development life cycle. Actually a majority of software development time is now spent on testing.

## **CHAPTER 2**

### **BRIEF EXPLANATION OF TRAINING**

#### **1. Problem Statement**

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- it is sufficiently usable,
- can be installed and run in its intended environments, and
- achieves the general result its stakeholders desire.

#### **2. Objective**

Although testing can determine the correctness of software under the assumption of some specific hypotheses (see the hierarchy of testing difficulty below), testing cannot identify all the defects within the software. Instead, it furnishes a *criticism* or *comparison* that compares the state and behavior of the product against test oracles—principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, contracts, comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions, but only that it does not function properly under specific conditions.<sup>[4]</sup> The scope of software testing often includes the examination of code as well as the execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team.



There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed. Every software product has a target audience.

## **Principle of Software Testing**

### **1) Exhaustive testing is not possible**

Yes! Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application.

And the million dollar question is, how do you determine this risk?

### **2) Defect Clustering**

Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

By experience, you can identify such risky modules. But this approach has its own problems. If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs.

### **3) Pesticide Paradox**

Repetitive use of the same pesticide mix to eradicate insects during farming will over time lead to the insects developing resistance to the pesticide. Thereby ineffective of pesticides on insects. The same applies to software testing. If the same set of repetitive tests are conducted, the method will be useless for discovering new defects. To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects. Testers cannot simply depend on existing test techniques. He must look out continually to improve the existing methods to make testing more effective. But even after all this sweat & hard work in testing, you can never claim your product is bug-free. To drive home this point, let's see this video of the public launch of Windows 98

### **4) Testing shows a presence of defects**

Hence, testing principle states that - Testing talks about the presence of defects and don't talk about the absence of defects. i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

But what if, you work extra hard, taking all precautions & make your software product 99% bug-free. And the software does not meet the needs & requirements of the clients.

### **5) Absence of Error - fallacy**

It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not mere finding

defects, but also to check that software addresses the business needs. The absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system build is unusable and does not fulfill the user's needs & requirements.

## **6) Early Testing**

Early Testing - Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages. It is much cheaper to fix a Defect in the early stages of testing. But how early one should start testing? It is recommended that you start finding the bug the moment the requirements are defined

## **7) Testing is context dependent**

Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application. All the developed software's are not identical. You might use a different approach, methodologies, techniques, and types of testing depending upon the application type. For instance testing, any POS system at a retail store will be different than testing an ATM machine.

# **3. Technology Used**

## **2.3.1 Manual testing**

Manual Testing is a type of Software Testing where Testers manually execute test cases without using any automation tools. Manual Testing is the most primitive of all testing types and helps find bugs in the software system.

Any new application must be manually tested before its testing can be automated. Manual Testing requires more effort but is necessary to check automation feasibility.

Manual Testing does not require knowledge of any testing tool.

## **Types of Manual Testing:**

- Black Box Testing
- White Box Testing
- Unit Testing
- System Testing
- Integration Testing
- Acceptance Testing

## **White-box testing**

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) verifies the internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing, an internal perspective of the system (the source code), as well as programming skills, are used to design test cases. The tester chooses

inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g., in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration, and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system-level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.



**Figure 2.1 White box testing**

## **Black-box testing**

Black-box testing (also known as functional testing) treats the software as a "black box," examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it. Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing, and specification-based testing.

Specification-based testing aims to test the functionality of software according to the applicable requirements. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional.



**Figure 2.2 Black box testing**

## **Unit testing**

Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to ensure that the building blocks of the software work independently from each other.

Unit testing is a software development process that involves a synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Unit testing aims to eliminate construction errors before code is promoted to additional testing; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development process.

Depending on the organization's expectations for software development, unit testing might include static code analysis, data-flow analysis, metrics analysis, peer code reviews, code coverage analysis and other software testing practices.

## **Integration testing**

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

## **System testing**

System testing tests a completely integrated system to verify that the system meets its requirements. For example, a system test might involve testing a logon interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff.

## **Acceptance testing**

Operational acceptance is used to conduct operational readiness (pre-release) of a product, service or system as part of a quality management system. OAT is a common type of non-functional software testing, used mainly in software development and software maintenance projects.

## **2. Automation Testing**

Manual Testing is performed by a human sitting in front of a computer carefully executing the test steps.

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, 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 ROI of 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.

## Automation Testing Tool

There are tons of Functional and Regression Testing Tools available in the market. Here are 5 best tools certified by our experts

### Selenium

It is a software testing tool used for Regression Testing. It is an open source testing tool that provides playback and recording facility for Regression Testing. The Selenium IDE only supports Mozilla Firefox web browser.

- It provides the provision to export recorded script in other languages like Java, Ruby, RSpec, Python, C#, etc
- It can be used with frameworks like JUnit and TestNG
- It can execute multiple tests at a time
- Autocomplete for Selenium commands that are common
- Walkthrough tests
- Identifies the element using id, name, X-path, etc.
- Store tests as Ruby Script, HTML, and any other format
- It provides an option to assert the title for every page
- It supports selenium user-extensions.js file
- It allows to insert comments in the middle of the script for better understanding and debugging

## Issue and Project Tracking software

### JIRA

JIRA is a tool developed by Australian Company Atlassian. It is used for **bug tracking**, **issue tracking**, and **project management**. The name "JIRA" is actually inherited from the Japanese word "Gojira" which means "Godzilla".

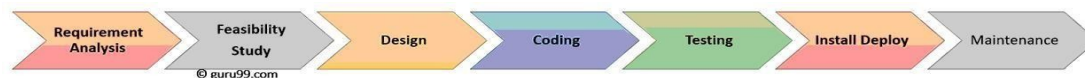
The basic use of this tool is to track issue and bugs related to your software and Mobile apps. It is also used for project management. The JIRA dashboard consists of many useful functions and features which make handling of issues easy.

- **SDLC & STLC**

### **3. SDLC (SOFTWARE DEVELOPMENT LIFE CYCLE)**

The Software Development Lifecycle is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software which meets customer expectations. The software development should be complete in the pre-defined time frame and cost.

SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC lifecycle has its own process and deliverables that feed into the next phase.



**Figure 2.3 SDLC**

#### **Requirement collection and analysis**

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognition of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project. Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

#### **Feasibility study**

Once the requirement analysis phase is completed the next step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

#### **There are mainly five types of feasibility checks:**

- **Economic:** Can we complete the project within the budget or not?
- **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances .
- **Operation feasibility:** Can we create operations which is expected by the client?
- **Technical:** Need to check whether the current computer system can support the software

- **Schedule:** Decide that the project can be completed within the given schedule or not.

## **Design**

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model. There are two kinds of design documents developed in this phase:

### **High-Level Design (HLD)**

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

### **Low-Level Design (LLD)**

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

## **Coding**

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

## **Testing**

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

## **Installation/Deployment**

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

### **Maintenance**

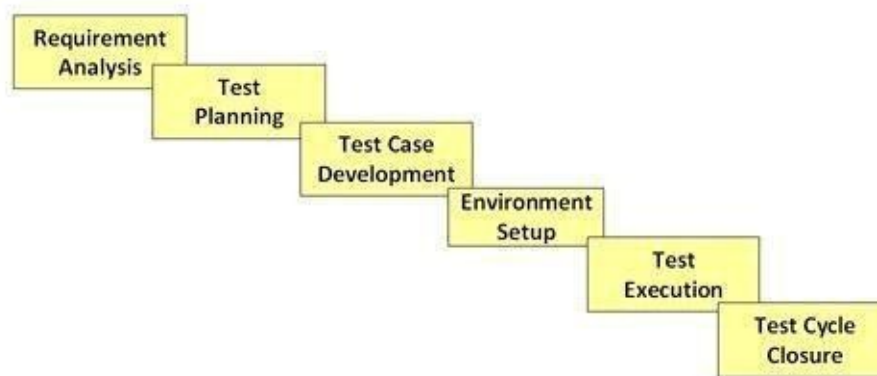
Once the system is deployed, and customers start using the developed system, following 3 activities occur

- Bug fixing - bugs are reported because of some scenarios which are not tested at all
- Upgrade - Upgrading the application to the newer versions of the Software
- Enhancement - Adding some new features into the existing software
- 

## **4. STLC (SOFTWARE TESTING LIFE CYCLE)**

Software Testing Life Cycle (STLC) is defined as a sequence of activities conducted to perform Software Testing.

Contrary to popular belief, Software Testing is not a just a single activity. It consists of a series of activities carried out methodologically to help certify your software product.



**Figure 2.4 STLC**

### **Requirement Analysis**

During this phase, test team studies the requirements from a testing point of view to identify the testable requirements.

The QA team may interact with various stakeholders (Client, Business Analyst, Technical Leads, System Architects etc) to understand the requirements in detail. Requirements could be either Functional (defining what the software must do) or Non Functional (defining system performance /security availability )

Automation feasibility for the given testing project is also done in this stage.

Activities



- Identify types of tests to be performed.
- Gather details about testing priorities and focus.
- Prepare Requirement Traceability Matrix (RTM).
- Identify test environment details where testing is supposed to be carried out.
- Automation feasibility analysis (if required).

#### Deliverables

- RTM
- Automation feasibility report. (if applicable)

### **Test Planning**

Typically, in this stage, a Senior QA manager will determine effort and cost estimates for the project and would prepare and finalize the Test Plan. In this phase, Test Strategy is also determined.

#### Activities

- Preparation of test plan/strategy document for various types of testing
- Test tool selection
- Test effort estimation
- Resource planning and determining roles and responsibilities.
- Training requirement

#### Deliverables

- Test plan /strategy document.
- Effort estimation document.

### **Test Case Development**

This phase involves the creation, verification and rework of test cases & test scripts. Test data, is identified/created and is reviewed and then reworked as well.

#### Activities

- Create test cases, automation scripts (if applicable)
- Review and baseline test cases and scripts
- Create test data (If Test Environment is available)

#### Deliverables

- Test cases/scripts
- Test data

## Environment Setup

Test environment decides the software and hardware conditions under which a work product is tested. Test environment set-up is one of the critical aspects of testing process and ***can be done in parallel with Test Case Development Stage. Test team may not be involved in this activity*** if the customer/development team provides the test environment in which case the test team is required to do a readiness check (smoke testing) of the given environment.

### Activities

- Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
- Setup test Environment and test data
- Perform smoke test on the build

### Deliverables

- Environment ready with test data set up
- Smoke Test Results.

## Test Execution

During this phase, the testers will carry out the testing based on the test plans and the test cases prepared. Bugs will be reported back to the development team for correction and retesting will be performed.

### Activities

- Execute tests as per plan
- Document test results, and log defects for failed cases
- Map defects to test cases in RTM
- Retest the Defect fixes
- Track the defects to closure

### Deliverables

- Completed RTM with the execution status
- Test cases updated with results
- Defect reports

## Test Cycle Closure

Testing team will meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in the future, taking lessons from the current test cycle. The idea is to remove the process bottlenecks for future test cycles and share best practices for any similar projects in the future.

### Activities

- Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software, Critical Business Objectives, Quality
- Prepare test metrics based on the above parameters.
- Document the learning out of the project
- Prepare Test closure report
- Qualitative and quantitative reporting of quality of the work product to the customer.
- Test result analysis to find out the defect distribution by type and severity.

#### Deliverables

- Test Closure report
- Test metrics

# **CHAPTER 3**

## **TRAINING MODULE**

### **1. AGILE MODEL**

Unlike the Waterfall method, Agile Testing can begin at the start of the project with continuous integration between development and testing. Agile Testing is not sequential (in the sense it's executed only after coding phase) but continuous.

An agile team works as a single team towards a common objective of achieving Quality. Agile Testing has shorter time frames called iterations (say from 1 to 4 weeks). This methodology is also called release, or delivery driven approach since it gives a better prediction on the workable products in short duration of time.

In this article, we will discuss

- Test Plan for Agile.
- Agile Testing Strategies.
- The Agile Testing Quadrant.
- QA challenges with agile software development.
- Risk of Automation in Agile Process.

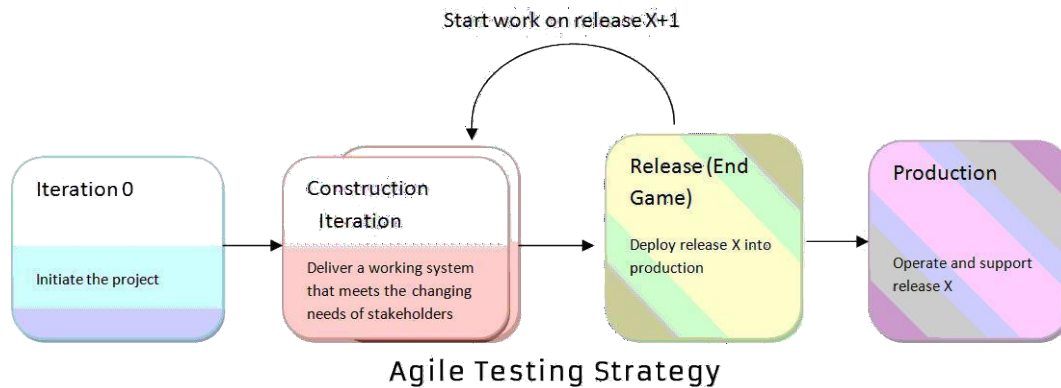
#### **1. Test Plan for Agile**

Unlike the waterfall model, in an agile model, a test plan is written and updated for every release. The agile test plan includes types of testing done in that iteration like test data requirements, infrastructure, test environments, and test results. Typical test plans in agile includes

- Testing Scope
- New functionalities which are being tested
- Level or Types of testing based on the features complexity
- Load and Performance Testing
- Infrastructure Consideration
- Mitigation or Risks Plan
- Resourcing
- Deliverables and Milestones

#### **2. Agile Testing Strategies**

Agile testing life cycle spans through four stages



**Figure 3.1 Agile Testing Strategy**

### **(a) Iteration 0**

During the first stage or iteration 0, you perform initial setup tasks. It includes identifying people for testing, installing testing tools, scheduling resources (usability testing lab), etc. The following steps are set to achieve in Iteration 0

- a) Establishing a business case for the project
- b) Establish the boundary conditions and the project scope
- c) Outline the key requirements and use cases that will drive the design trade-offs
- d) Outline one or more candidate architectures
- e) Identifying the risk
- f) Cost estimation and prepare a preliminary project

### **(b) Construction Iterations**

The second phase of testing is Construction Iterations, the majority of the testing occurs during this phase. This phase is observed as a set of iterations to build an increment of the solution. In order to do that, within each iteration, **the team implements** a hybrid of practices from XP, Scrum, Agile modeling, and agile data and so on.

In construction iteration, the agile team follows the prioritized requirement practice: With each iteration, they take the most essential requirements remaining from the work item stack and implement them.

Construction iteration is classified into two, confirmatory testing and investigative testing. **Confirmatory testing concentrates** on verifying that the system fulfills the intent of the stakeholders as described to the team to date, and is performed by the team. While the investigative testing detects the problem that confirmatory team has skipped or ignored. In Investigative testing, tester determines the potential problems in the form of defect stories. Investigative testing deals with common issues like integration testing, load/stress testing, and security testing.

Again for, confirmatory testing there are two aspects **developer testing** and **agile acceptance testing**. **Both of them** are automated to enable continuous regression testing throughout the lifecycle. Confirmatory testing is the agile equivalent of testing to the specification.

Agile acceptance testing is a combination of traditional functional testing and traditional acceptance testing as the development team, and stakeholders are doing it together. While developer testing is a mix of traditional unit testing and traditional service integration testing. Developer testing verifies both the application code and the database schema.

### (c) Release End Game Or Transition Phase

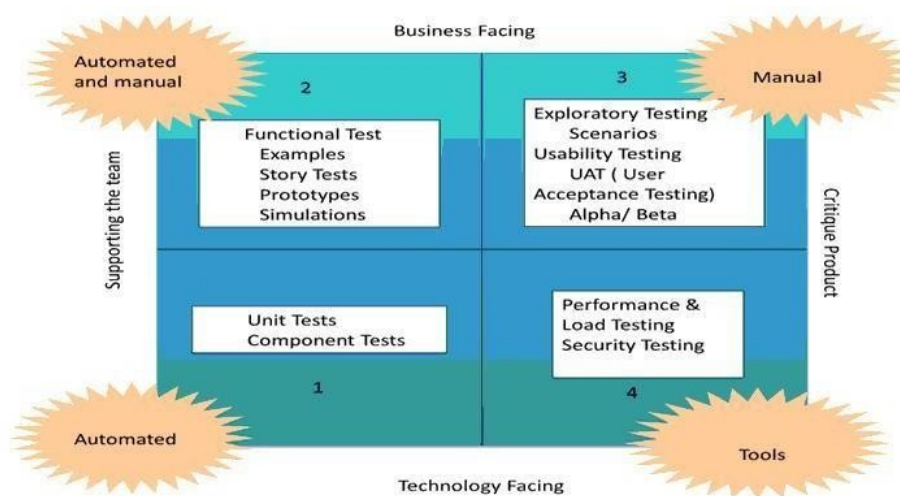
The goal of —Release, End Game! is to deploy your system successfully into production. The activities include in this phase are training of end users, support people and operational people. Also, it includes marketing of the product release, back-up & restoration, finalization of system and user documentation.

The final testing stage includes full system testing and acceptance testing. In accordance to finish your final testing stage without any obstacles, you should have to test the product more rigorously while it is in construction iterations. During the end game, testers will be working on its defect stories.

### (d) Production

After the release stage, the product will move to the production stage.

### The Agile Testing Quadrants



**Figure 3.2 Technology Facing**

The agile testing quadrants separate the whole process in four Quadrants and help to understand how agile testing is performed.

a) **Agile Quadrant I** – The internal code quality is the main focus in this quadrant, and it consists of test cases which are technology driven and are implemented to support the team, it includes

1. Unit Tests
2. Component Tests

b) **Agile Quadrant II** – It contains test cases that are **business driven and are implemented** to support the team. This Quadrant focuses on the requirements. The kind of test performed in this phase is

1. Testing of examples of possible scenarios and workflows
2. Testing of User experience such as prototypes
3. Pair testing

c) **Agile Quadrant III** – This quadrant provides feedback to quadrants one and two. The test cases can be used as the basis to perform automation testing. In this quadrant, many rounds of iteration reviews are carried out which builds confidence in the product. The kind of testing done in this quadrant is

1. Usability Testing
2. Exploratory Testing
3. Pair testing with customers
4. Collaborative testing
5. User acceptance testing

d) **Agile Quadrant IV** – **This quadrant concentrates** on the non-functional requirements such as performance, security, stability, etc. With the help of this quadrant, the application is made to deliver the non-functional qualities and expected value.

1. Non-functional tests such as stress and performance testing
2. Security testing with respect to **authentication** and hacking
3. Infrastructure testing
4. Data migration testing
5. Scalability testing
6. Load testing

### **QA challenges with agile software development**

- a) Chances of error are more in agile, as documentation is given less priority, eventually puts more pressure on QA team
- b) New features are introduced quickly, which reduces the available time for test teams to identify whether the latest features are according to the requirement and does it truly address the business suits
- c) Testers are often required to play a semi-developer role
- ) Test execution cycles are highly compressed
- e) Very less time to prepare test plan
- f) For regression testing, they will have minimal timing
- g) Change in their role from being a gate-keeper of quality to being a partner in Quality
- h) Requirement changes and updates are inherent in an agile method, becoming the biggest challenge for QA

### **3. Risk of Automation in Agile Process**

- Automated UI provides a high level of confidence, but they are slow to execute, fragile to maintain and expensive to build. Automation may not significantly improve test productivity unless the testers know how to test
- Unreliable tests are a major concern in automated testing. Fixing failing tests and resolving issues related to brittle tests should be a top priority in order to avoid false positives
- If the automated test are initiated manually rather than through CI (Continuous Integration) then there is a risk that they are not regularly running and therefore may cause failing of tests
- Automated tests are not a replacement for an exploratory manual testing. To obtain the expected quality of the product, a mixture of testing types and levels is required
- Many commercially available automation tools provide simple features like automating the capture and replay of manual test cases. Such tool encourages testing through the UI and leads to an inherently brittle and difficult to maintain tests. Also, storing test cases outside the version control system creates unnecessary complexity
- In order to save time, much times the automation test plan is poorly planned or unplanned which results in the test fail
- A test set up and tear down procedures are usually missed out during test automation, while Performing manual testing, a test set up and tear down procedures sounds seamless
- Productivity metrics such as a number of test cases created or executed per day can be terribly misleading, and could lead to making a large investment in running useless tests
- Members of the agile automation team must be effective consultants: approachable, cooperative, and resourceful, or this system will quickly fail
- Automation may propose and deliver testing solutions that require too much ongoing maintenance relative to the value provided
- Automated testing may lack the expertise to conceive and deliver effective solutions
- Automated testing may be so successful that they run out of important problems to solve, and thus turn to unimportant problems.

## **2. TEST LINK**

**Test Link** is a web-based test management system that facilitates software quality assurance. It is developed and maintained by Team test. The platform offers support for test cases, test suites, test plans, test projects and user management, as well as various reports and statistics.

The basic units used by Test Link are: Test Case, Test Suite, Test Plan, Test Project and User.

### **1. Test Plan**

Test Plans are the basic unit for executing a set of tests on an application. Test Plans include Builds, Milestones, User assignment and Test Results.



A Test Plan contains name, description, collection of chosen Test Cases, Builds, Test Results, milestones, tester assignment and priority definition. Each Test Plan is related to the current Test Project.

Test Plans may be created from the "Test Plan management" page by users with lead privileges for the current Test Project. Press "Create" button and enter data.

Test Plan definition consists of title, description (html format) and status "Active" check-box. Description should include the next information with respect to company processes:

- Summary/Scope
- Features to be tested
- Features to not be tested
- Test criteria (to pass tested product)
- Test environment, Infrastructure
- Test tools
- Risks
- References (Product plan or Change request, Quality document(s), etc.)

Test Plans are made up of Test Cases imported from a Test Specification at a specific point of time. Test Plans may be created from other Test Plans. This allows users to create Test Plans from Test Cases that exist at a desired point in time. This may be necessary when creating a Test Plan for a patch. In order for a user to see a Test Plan they must have the proper rights. Rights may be assigned (by leads) in the define User/Project Rights section. This is an important thing to remember when users tell you they can't see the project they are working on.

Test Plans may be deleted by users with lead privileges.

## **2. Test Case**

A Test Case describes a simple task in the workflow of an application. A test case is a fundamental part of TestLink. After a tester runs a test case it can either pass, fail or block it. Test cases are organized in test suites. Test Cases have the following parts:

- Identifier of a Test Case is assigned automatically by TestLink, and cannot be changed by users. This ID composes from Test Project prefix and a counter related to the Test Project in which the Test Case is created.
- Title: could include either short description or abbreviation (e.g. TL-USER-LOGIN)
- Summary: should be really short; just for overview, introduction and references.
- Steps: describe test scenario (input actions); can also include precondition and clean-up information here.
- Expected results: describe checkpoints and expected behaviour of a tested product or system.
- Attachments: could be added if configuration allows it.
- Importance: Test designer could set importance of the test [HIGH, MEDIUM and LOW].
- Execution type: Test designer could set automation support of the test [MANUAL/AUTOMATED]

- Custom fields: Administrator could define own parameters to enhance Test Case description or categorization. Large custom fields (more than 250 characters) are not possible. But information could be added into parent Test Suite and referred via custom fields. For example, you can describe Configuration 'standard', 'performance', 'standard\_2' and refer via CF to this labels.

## **User**

Each TestLink user has an assigned Role that defines the features available. The default types are: Guest, Test Designer, Senior tester, Tester, Leader and Administrator but custom roles can also be created.

## **3. Test Projects**

Test Projects are the basic organisational unit of TestLink. Test Projects could be products or solutions of your company that may change their features and functionality over time but for the most part remains the same. Test Project includes requirements documentation, Test Specification, Test Plans and specific user rights. Test Projects are independent and do not share data.

## **4. Test Specifications**

TestLink breaks down the Test Specification structure into Test Suites and Test Cases. These levels are persisted throughout the application. One Test Project has just one Test Specification.

## **5. Advantages of TestLink**

- It supports multiple projects
- Easy export and import of test cases
- Easy to integrate with many defect management tools
- Automated test cases execution through XML-RPC
- Easy filtration of test cases with version, keywords, test case ID and version
- Easy to assign test cases to multiple users
- Easy to generate test plan and test reports in various formats
- Provide credentials to multiple users and assign roles to them

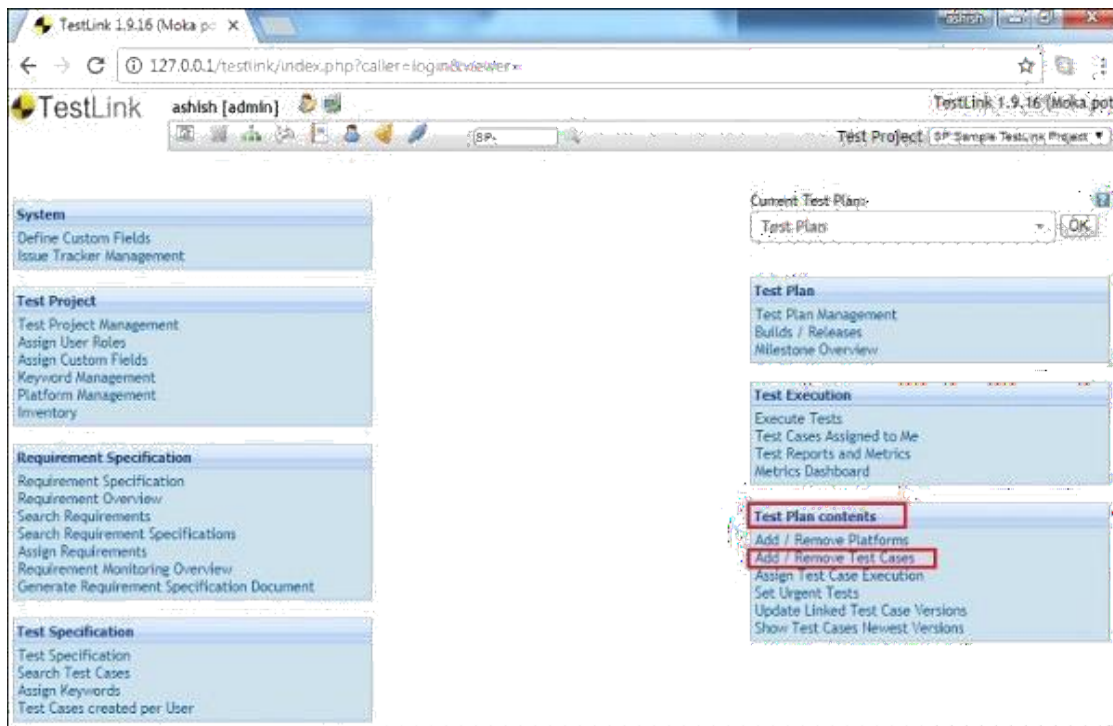


Figure 3.3 Test Link

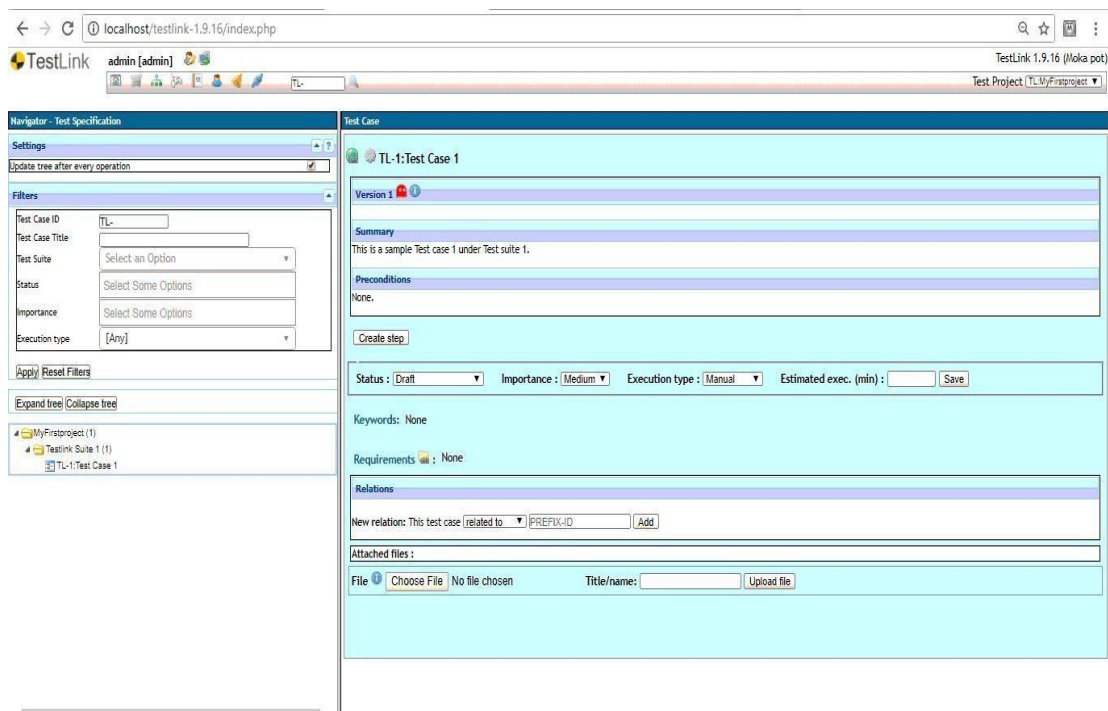












Figure 3.4 Test case

Version 1  				
<b>Summary</b>				
The test case is to check the login functionality of Guru 99 home-page				
<b>Preconditions</b>				
1. The user should be connected to the internet 2. User should have valid user#id and password				
#	Step actions	Expected Results	Execution	
1	Open Guru 99 website	The website should be opened	Manual	 
2	Enter username in the username textbox	Text-box should accept the entered data	Manual	 
3	Enter password in the password text-box	Text-box should accept the entered data	Manual	 
4	Click on "sign in" button	Login should success and navigate to the home page	Manual	 

The test step will appear like this on the window after save and exit

Figure 3.5 Step Action

## 6. Steps for creating a Test case:

- Login to your Test link account as the credentials given to you by the company.
- Create a test plan using Test plan management button.
- Then select that particular test plan and create a build in that plan.
- Now click on test specification button.
- When a new screen will open head on to the action button and select create.
- By this process your test suite will be created.
- The suite is visible at the scroll bar list of the left side of the page.
- After the test suite is created click on action button after select test suite and create a test case.
- Summary, precondition has to be mentioned for that particular test case
- What type of test case is it? Will decide the type of testing to be used and to be done.
- After the successful creation of a test cases you need to add steps action for the same.
- Step actions include – To observe the header section of the page, To validate the functionality of login button, etc.
- After that go to add and remove test cases. Click on all the check boxes and assign the test cases to the particular tester.
- When the test cases are assigned then click on execute test button.
- According to the test case if any of the step action is not working properly then you can simply click on failed.

- After that just click on test report and metrics and select report format, select all the check boxes field and click on print
- You are ready with your report.

# CHAPTER 4

## TEST MANAGEMENT TOOLS

### 1. JIRA

JIRA is a tool developed by Australian Company Atlassian. It is used for **bug tracking, issue tracking, and project management**. The name "JIRA" is actually inherited from the Japanese word "Gojira" which means "Godzilla".

The basic use of this tool is to track issue and bugs related to your software and Mobile apps. It is also used for project management. The JIRA dashboard consists of many useful functions and features which make handling of issues easy. Some of the key features are listed below. Let's learn JIRA Defect and Project tracking software with this Training Course.

#### **1.1. Use of JIRA**

- JIRA is used in Bugs, Issues and Change Request Tracking.
- JIRA can be used in Help desk, Support and Customer Services to create tickets and track the resolution and status of the created tickets.
- JIRA is useful in Project Management, Task Tracking and Requirement Management.
- JIRA is very useful in Workflow and Process management.

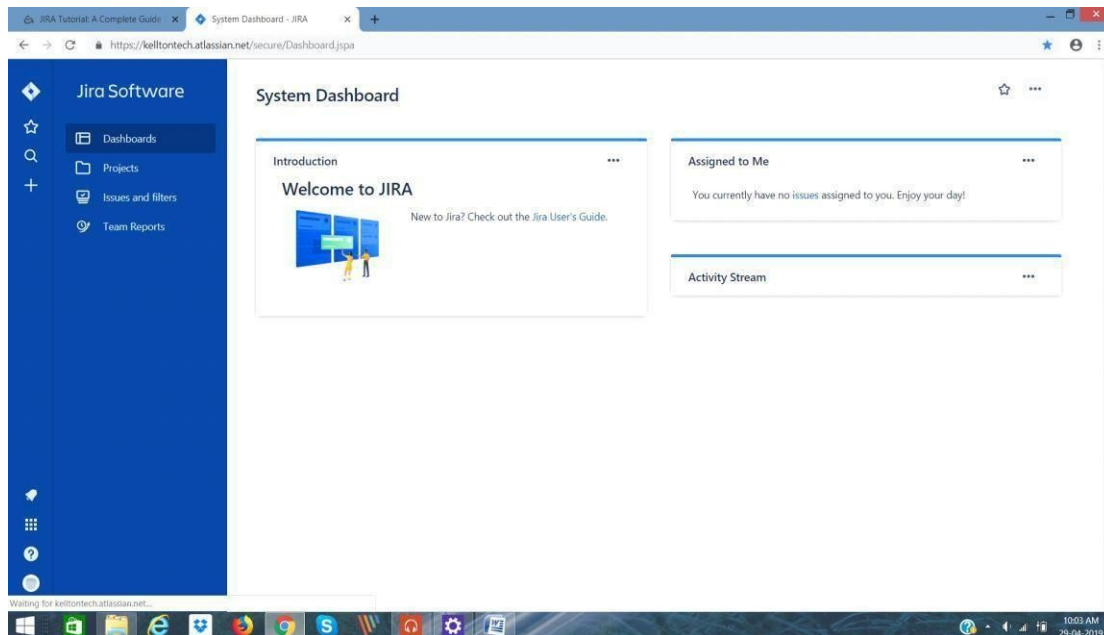


Figure 4.1 JIRA Dashboard

Create Issue

Summary\*

Component/s None

Description

?

Priority 

↑ Medium ↓

 ?

Environment

Style ▾ B I U A ▾ A ▾ + ▾ ≡

?

For example operating system, software platform and/or hardware specifications (include as appropriate for the issue).

Attachment

Drop files to attach, or browse.

Assignee 

Automatic ▾

Create another Create Cancel

### Figure 4.2 Create Issue

### 1.2. for creating an Issue:

JIRA issue would track bug or issue that underlies the project. Once you have imported project then you can create issues.

- **Priority** – Issue creator can set the priority to resolve the issue as High, Medium, Low, and Lowest.
- **Labels** – It is similar to **Tag**; it helps in filtering out specific types of issues.
- **Linked Issue** – It links other issues that are either dependent on this issue or this issue is dependent on them. Options in dropdowns are – block, is blocked by, duplicate, clone, etc.
- **Issue** – User can link the issue by the **Typing ID** or summary of those that are related to the linked issue field.
- **Assignee** – The person who is responsible to fix this issue. Assignee name can be entered by the issue creator.
- **Epic Link** – An Issue creator can provide an epic link, if the issue belongs to any of those.

- **Sprint** – The user can define in which sprint, this issue belongs to, when this issue should be addressed.

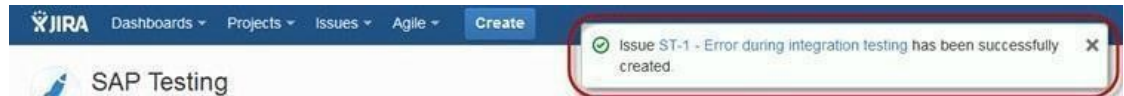


Figure 4.3 Issue Created

When issue is created then a popup will occur for the confirmation of creation.

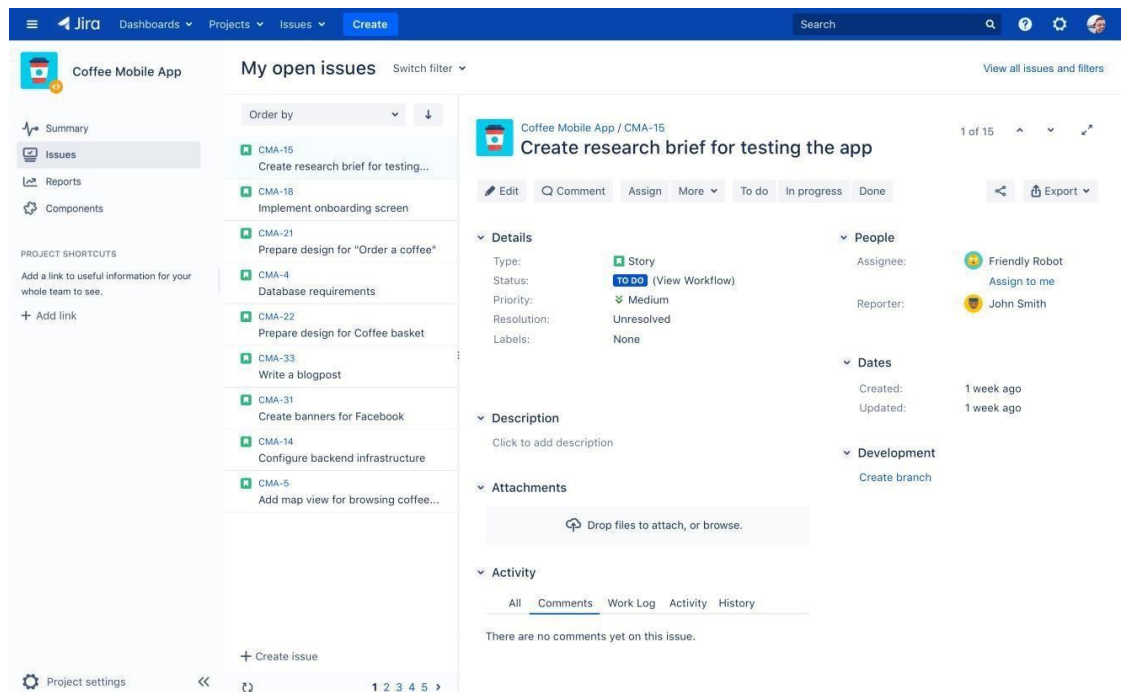


Figure 4.4 Open Issues

- The navigation bar, present at the top of the JIRA page, will be the same across all the pages/screens of JIRA. Dashboard, Projects, Issues, Boards and Create are the main links. These links have many sub-links to navigate other functionalities.
- Navigation bar contains links that provides a quick access to the most useful functions of JIRA.
- Just under navigation bar, there is a System Dashboard.
- The information provided in the system dashboard area can be customized by the Admin.
- By default, it has three main sections – **Introduction**, **Assigned to Me** (displays Issues list assigned to users) and **Activity Stream** (Activities done by the users).



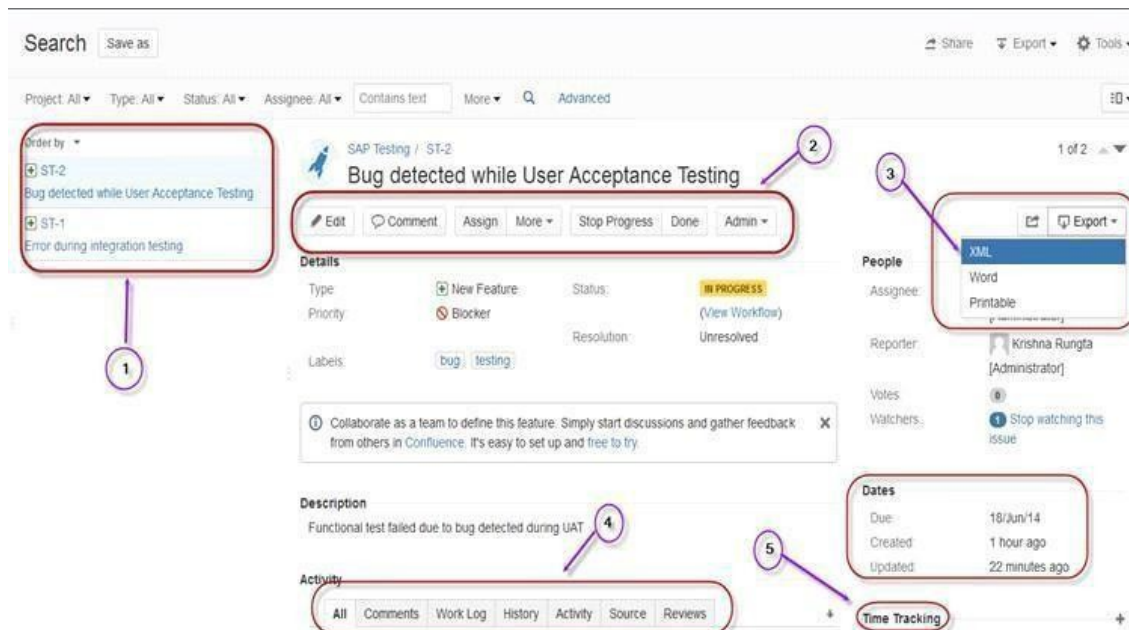


Figure 4.5

#### Bug Details

- **Search for issues** option will bring you to a window where you can see the issues created by you like here we have issues ST1 and ST2
- Here in the screen shot you can see the issue "**Bug detected while User Acceptance Testing**" and all the details related to it. From here, you can perform multiple tasks like you can **stop the progress on issues, edit the issues, comment on the issues, assigning issues** and so on
- Even you can export issue details to a XML or Word document.
- Also, you can view activity going on the issue, reviews on the issue, work log, history of the issue and so on.
- Under the time tracking option, you can even see the estimation time to resolve the issues

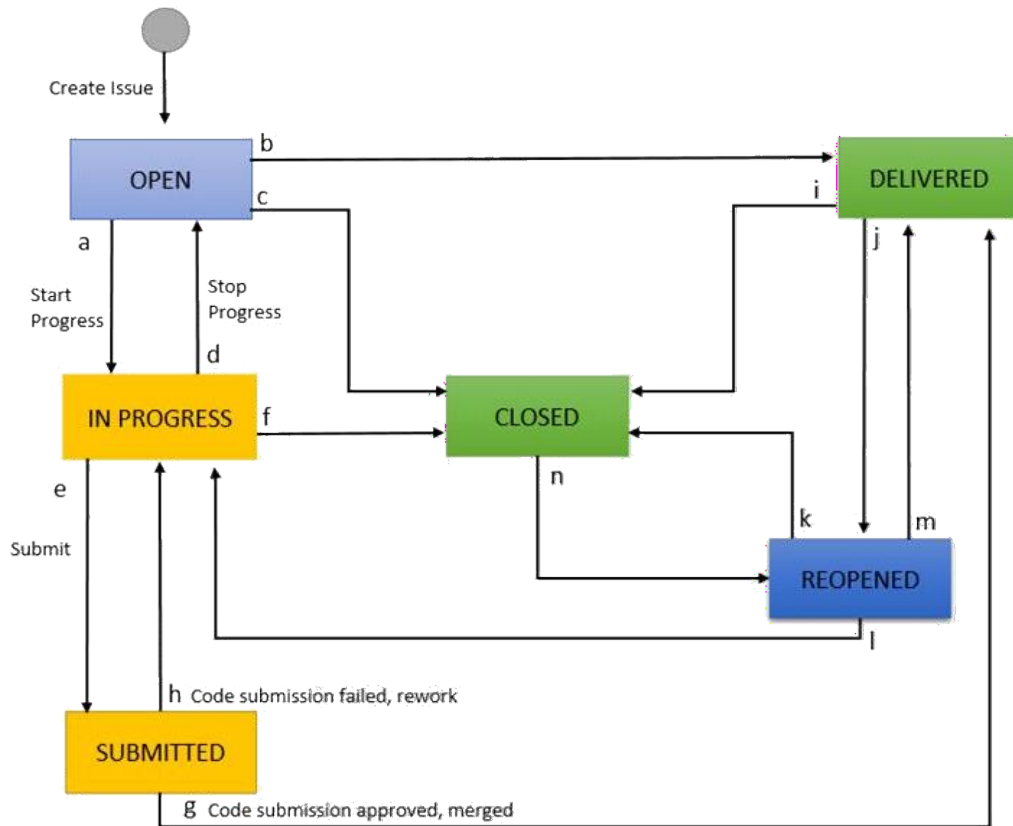


Figure 4.6 Bug life cycle

Different statuses are used to indicate the progress of a project like **To do, In Progress, Open, Closed, Re Opened, and Resolved**. Likewise, you have resolutions and priorities, in resolution it again tells about the progress of issue like **Fixed, Won't fix, Duplicate, Incomplete, Cannot reproduce, Done** also you can set the priorities of the issue whether an issue is **critical, major, minor, blocker and Trivial**.

- **Open Issue** – After creation, the issue is open and can be assigned to the assignee to start working on it.
- **In Progress Issue** – The assignee has actively started to work on the issue.
- **Resolved Issue** – All sub-tasks and works of that Issue are completed. Now, the issue is waiting to be verified by the reporter. If verification is successful, it will be closed or re-opened, if any further changes are required.
- **Reopened Issue** – This issue was resolved previously, but the resolution was either incorrect or missed a few things or some modifications are required. From Reopened stage, issues are marked either as assigned or resolved.
- **Close Issue** – The issue is considered as finished, resolution is correct as of now. Closed issues can be re-opened later based on the requirement.

## 2. Selenium

Selenium is a portable framework for testing web applications. Selenium provides a playback (formerly also recording) tool for authoring functional tests without the need to learn a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including C#, Groovy, Java, Perl, PHP, Python, Ruby and Scala. The tests can then run against most modern web browsers. Selenium deploys on Windows, Linux, and macOS platforms. It is open-source software.

### 2.1. Components

- **Selenium IDE**

Selenium IDE is a complete integrated development environment (IDE) for Selenium tests. It is implemented as a Firefox Add-On and as a Chrome Extension. It allows for recording, editing, and debugging of functional tests. It was previously known as Selenium Recorder. Selenium-IDE was originally created by Shinya Kasatani and donated to the Selenium project in 2006. Selenium IDE was previously little-maintained. Selenium IDE began being actively maintained in 2018. Scripts may be automatically recorded and edited manually providing autocompletion support and the ability to move commands around quickly. Scripts are recorded in *Selenese*, a special test scripting language for Selenium. Selenese provides commands for performing actions in a browser, and for retrieving data from the resulting pages.

- **Selenium WebDriver**

Selenium WebDriver accepts commands (sent in Selenese) and sends them to a browser. This is implemented through a browser-specific browser driver, which sends commands to a browser and retrieves results. Most browser drivers actually launch and access a browser application (such as Firefox, Chrome, Internet Explorer, Safari, or Microsoft Edge). Selenium WebDriver does not need a special server to execute tests. Instead, the WebDriver directly starts a browser instance and controls it.

### 2.2. Locators in Selenium

Locator is a command that tells Selenium IDE which GUI elements (Text Box, Buttons, Check Boxes) it needs to operate on. Identification of correct GUI elements is a prerequisite to creating an automation script.

The different types of Locators in Selenium IDE are-

#### a) Locating by ID

This is the most common way of locating elements since ID's are supposed to be unique for each element.

Syntax- `//input[@id='identifierId']`

Example- Inspect the "Email or Phone" text box using Firebug and take note of its ID. In this case, the ID is "email."

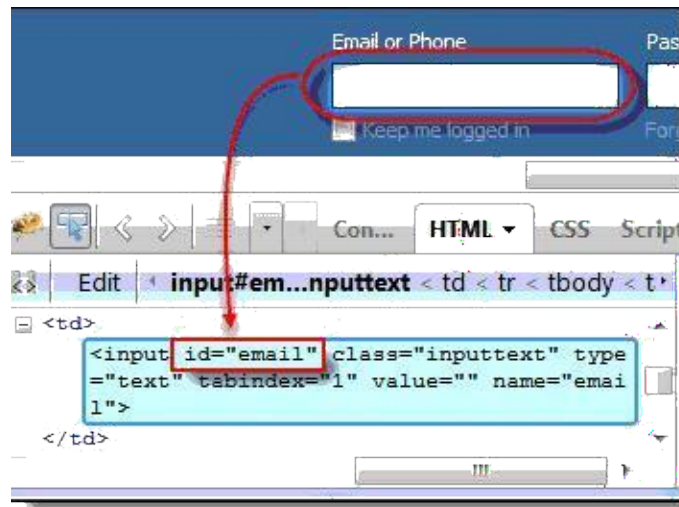


Figure 4.7 Locating by ID

Launch Selenium IDE and enter "id=email" in the Target box. Click the Find button and notice that the "Email or Phone" text box becomes highlighted with yellow and bordered with green, meaning, Selenium IDE was able to locate that element correctly.

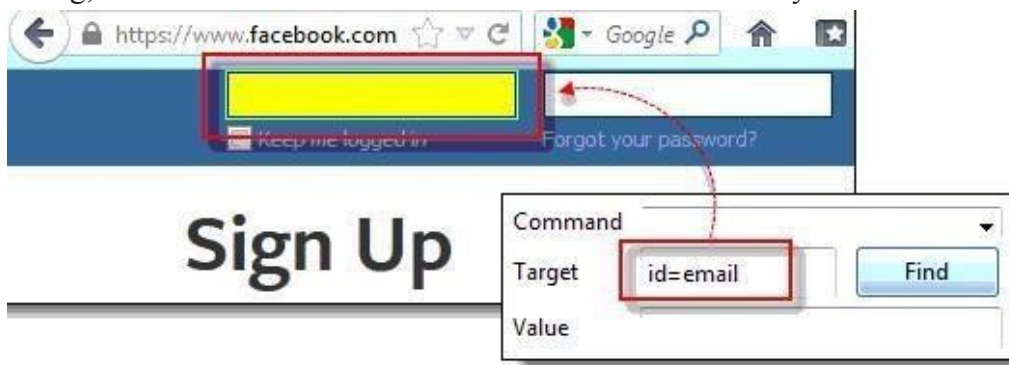


Figure 4.8 Sign up

## b) Locating By Name

Locating elements by name are very similar to locating by ID, except that we use the "name=" prefix instead.

Syntax- //input[@name='userName']

Example- Inspect the "User Name" text box.

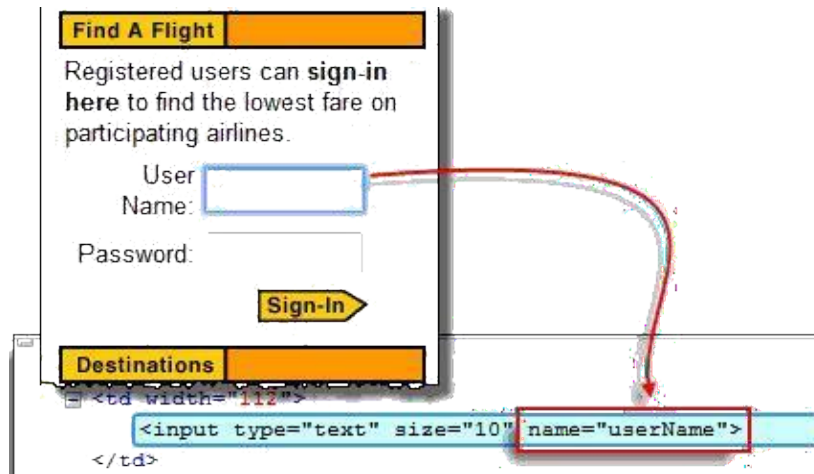


Figure 4.9 Locating by Name

Here, we see that the element's name is "userName".

In Selenium IDE, enter "name=userName" in the Target box and click the Find button. Selenium IDE should be able to locate the User Name text box by highlighting it.

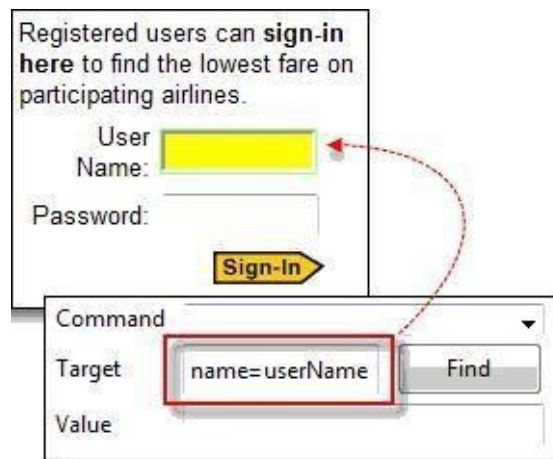


Figure 4.10 Username

### c) Locating by Link Text

This type of locator applies only to hyperlink texts. We access the link by prefixing our target with "link=" and then followed by the hyperlink text.

Example- First, make sure that you are logged off from Mercury Tours. Go to Mercury Tours homepage. Using Firebug, inspect the "REGISTER" link. The link text is found between and tags. In this case, our link text is "REGISTER". Copy the link text.



**Figure 4.11 Search link text element**

Copy the link text in Firebug and paste it onto Selenium IDE's Target box. Prefix it with "link=".



The screenshot shows the Selenium IDE interface. At the top, there is a table with three columns: Command, Target, and Value. The first row contains the text 'link=REGISTER'. Below this table, there is a form with three fields: Command, Target, and Value. The Target field contains the text 'link=REGISTER', which is highlighted with a red rectangular box. To the right of the Target field is a button labeled 'Find'.

**Figure 4.12 Find register element**

Click on the Find button and notice that Selenium IDE was able to highlight the REGISTER link correctly.



**Figure 4.13 Register element**

To verify further, enter "clickAndWait" in the Command box and execute it. Selenium IDE should be able to click on that REGISTER link successfully and take you to the Registration page shown below.

**REGISTER**

To create your account, we'll need some basic information about you. This information will be used to send reservation confirmation emails, mail tickets when needed and contact you if your travel arrangements change. Please fill in the form completely.

**Contact Information**

First Name:

Last Name:

Phone:

Figure 4.14 Register element containing webpage

#### d) Locating by CSS Selector

CSS Selectors are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes. Locating by CSS Selector is more complicated than the previous methods, but it is the most common locating strategy of advanced Selenium users because it can access even those elements that have no ID or name.

CSS Selectors have many formats, but we will only focus on the most common ones.

- i. Tag and ID
- ii. Tag and class
- iii. Tag and attribute
- iv. Tag, class, and attribute
- v. Inner text

When using this strategy, we always prefix the Target box with "css=" as will be shown in the following examples.

##### i. Locating by CSS Selector - Tag and ID

Again, we will use Facebook's Email text box in this example. As you can remember, it has an ID of "email," and we have already accessed it in the "Locating by ID" section. This time, we will use a CSS Selector with ID in accessing that very same element.

Syntax	Description
<code>css=tag#id</code>	<ul style="list-style-type: none"> <li>tag = the HTML tag of the element being accessed</li> <li># = the hash sign. This should always be present when using a CSS Selector</li> </ul>

	<p>with ID</p> <ul style="list-style-type: none"> <li>id = the ID of the element being accessed</li> </ul>
--	--

Keep in mind that the ID is always preceded by a hash sign (#).

Using Firebug, examine the "Email or Phone" text box.

At this point, take note that the HTML tag is "input" and its ID is "email". So our syntax will be "css=input#email".

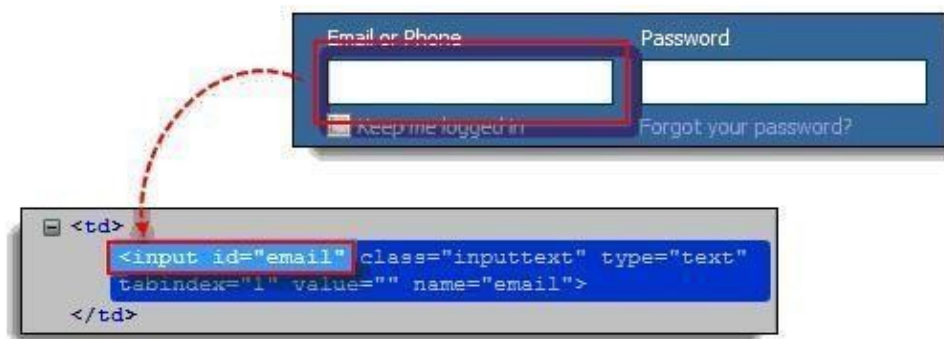


Figure 4.15 Search CSS selector element

Enter "css=input#email" into the Target box of Selenium IDE and click the Find button. Selenium IDE should be able to highlight that element.

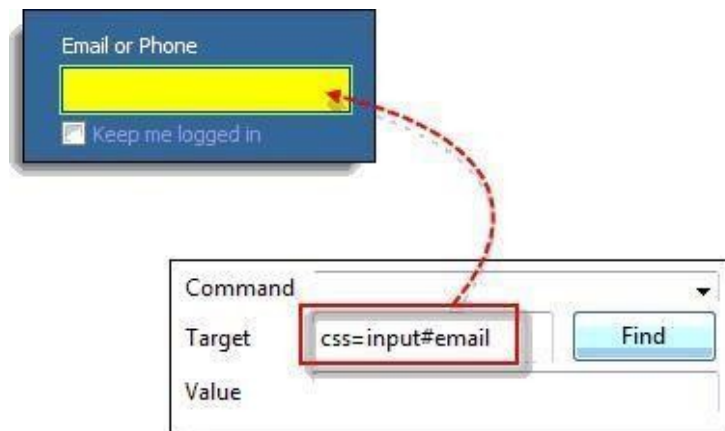


Figure 4.16 Find CSS selector element

## ii. Locating by CSS Selector - Tag and Class

Locating by CSS Selector using an HTML tag and a class name is similar to using a tag and ID, but in this case, a dot (.) is used instead of a hash sign.

Synta x	Descriptio n
------------	-----------------



$css=tag.class$	<ul style="list-style-type: none"> <li>• tag = the HTML tag of the element being accessed</li> <li>• . = the dot sign. This should always be present when using a CSS Selector with class</li> <li>• class = the class of the element being accessed</li> </ul>
-----------------	---

Go to the web page and use Firebug to inspect the "Email or Phone" text box. Notice that its HTML tag is "input" and its class is "inputtext."

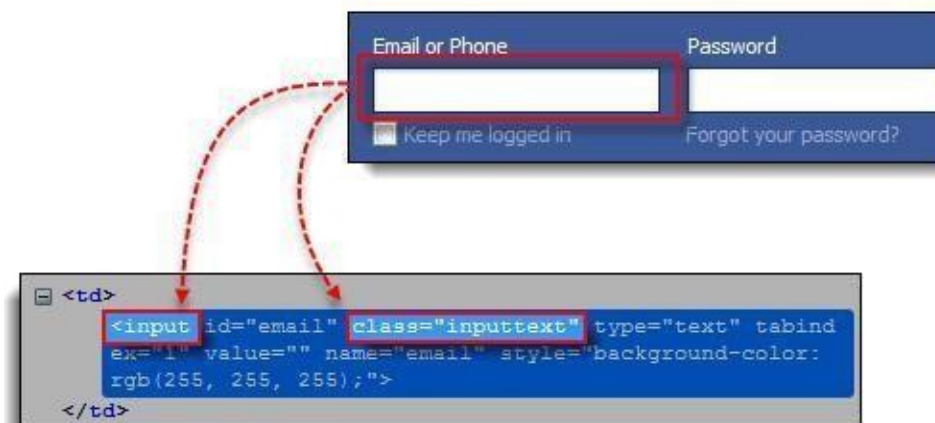


Figure 4.17 Search CSS selector class

In Selenium IDE, enter "css=input.inputtext" in the Target box and click Find. Selenium IDE should be able to recognize the Email or Phone text box.

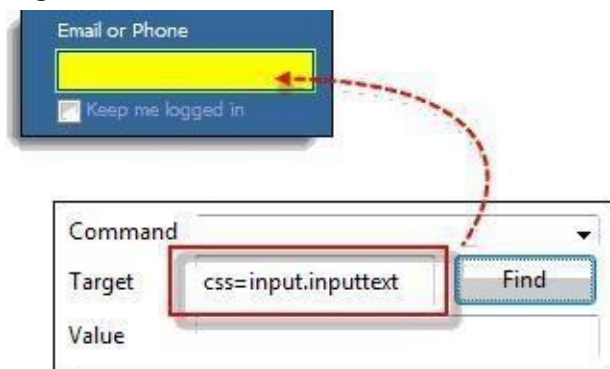


Figure 4.18 Find element in CSS selector

When multiple elements have the same HTML tag and name, only the first element in source code will be recognized. Using Firebug, inspect the Password text box in Facebook and notice that it has the same name as the Email or Phone text box.



Figure 4.19 ID element in CSS selector

The reason why only the Email or Phone text box was highlighted in the previous illustration is that it comes first in Facebook's page source.

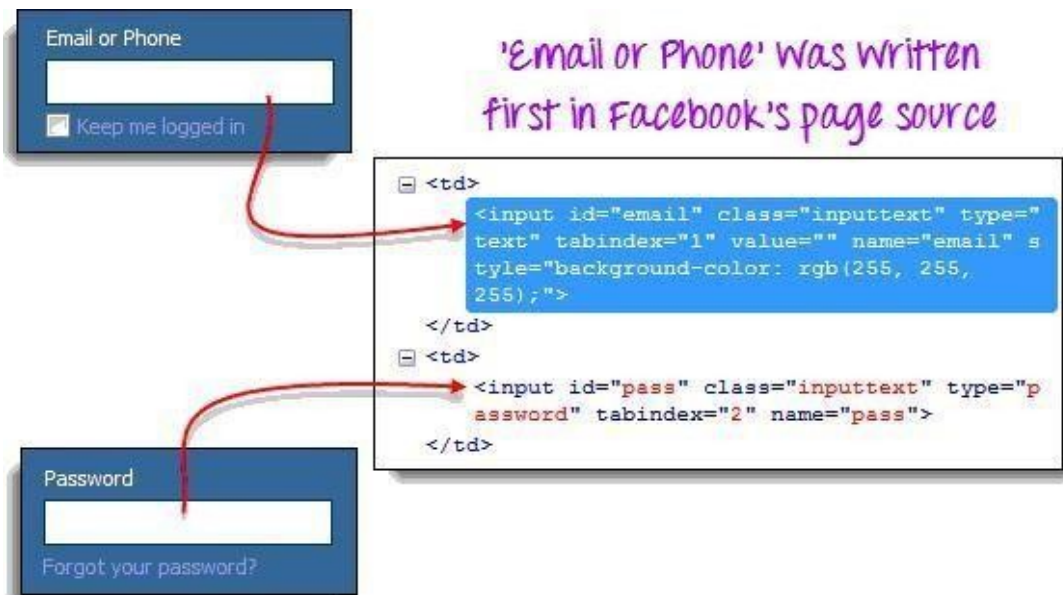


Figure 4.20 Tag element in CSS selector

### iii. Locating by CSS Selector - Tag and Attribute

This strategy uses the HTML tag and a specific attribute of the element to be accessed.

Syntax	Description
<code>css=tag[attribute=value]</code>	<ul style="list-style-type: none"> <li>tag = the HTML tag of the element being accessed</li> <li>[ and ] = square brackets within which</li> </ul>

	<p>a specific attribute and its corresponding value will be placed</p> <ul style="list-style-type: none"> <li>• attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID.</li> <li>• value = the corresponding value of the chosen attribute.</li> </ul>
--	--

Navigate to Mercury Tours' Registration page and inspect the "Last Name" text box. Take note of its HTML tag ("input" in this case) and its name ("lastName").

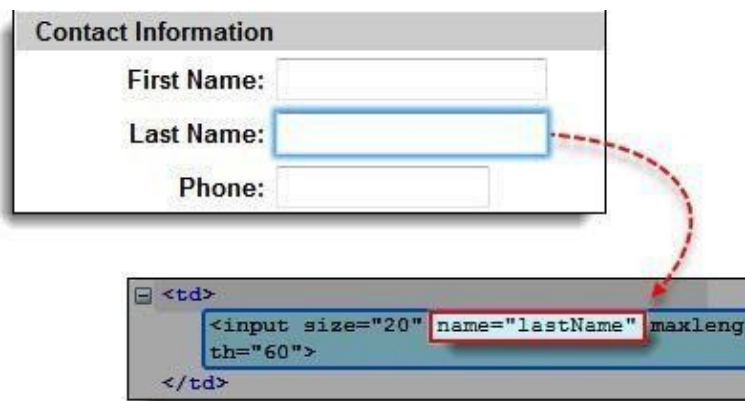


Figure 4.21 Last name field in inspect

In Selenium IDE, enter "css=input[name=lastName]" in the Target box and click Find. Selenium IDE should be able to access the Last Name box successfully.

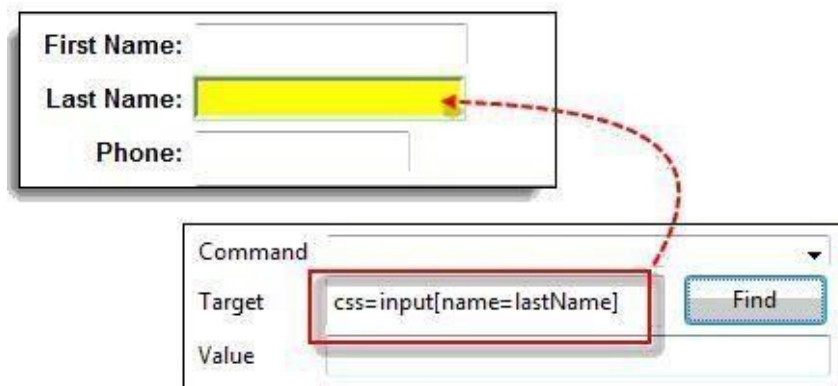


Figure 4.22 Find last name element

When multiple elements have the same HTML tag and attribute, only the first one will be recognized. This behavior is similar to locating elements using CSS selectors with the same tag and class.

#### iv. Locating by CSS Selector - tag, class, and attribute

Syntax	Description
<code>css=tag.class[attribute=value]</code>	<ul style="list-style-type: none"><li>• tag = the HTML tag of the element being accessed</li><li>• . = the dot sign. This should always be present when using a CSS Selector with class</li><li>• class = the class of the element being Accessed</li><li>• [ and ] = square brackets within which a specific attribute and its corresponding value will be placed</li><li>• attribute = the attribute to be used. It is advisable to use an attribute that is unique to the element such as a name or ID.</li><li>• value = the corresponding value of the chosen attribute.</li></ul>

Go to the demo page and use Firebug to inspect the 'Email or Phone' and 'Password' input boxes. Take note of their HTML tag, class, and attributes. For this example, we will select their 'tabindex' attributes.

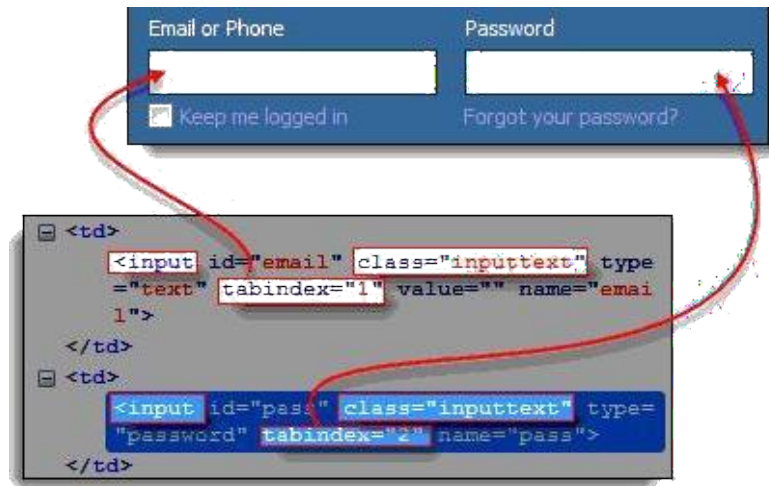


Figure 4.23 class tag in CSS selector

We will access the 'Email or Phone' text box first. Thus, we will use a tabindex value of 1. Enter "css=input.inputtext[tabindex=1]" in Selenium IDE's Target box and click Find. The 'Email or Phone' input box should be highlighted.

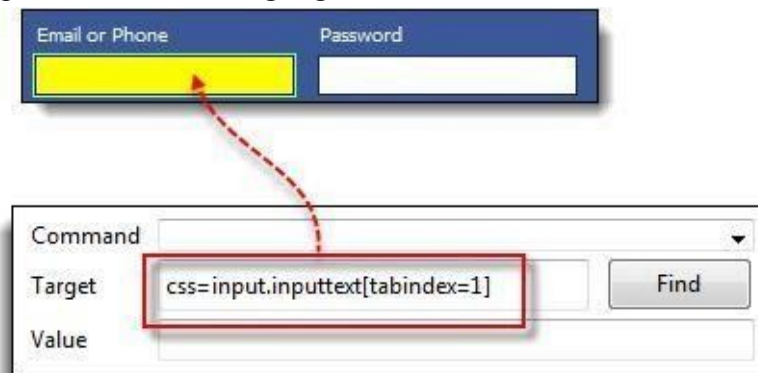


Figure 4.24 Find element in CSS selector class

To access the Password input box, simply replace the value of the tabindex attribute. Enter "css=input.inputtext[tabindex=2]" in the Target box and click on the Find button. Selenium IDE must be able to identify the Password text box successfully.

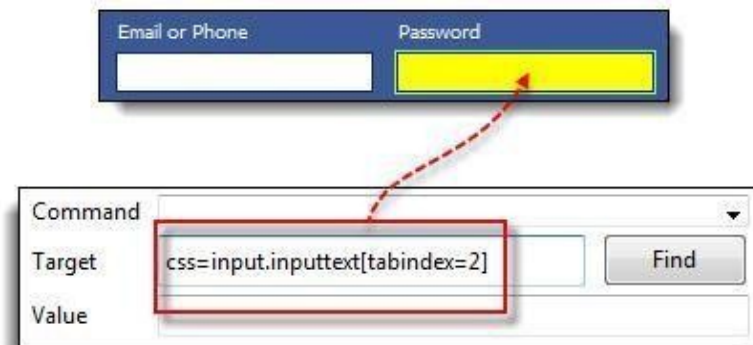


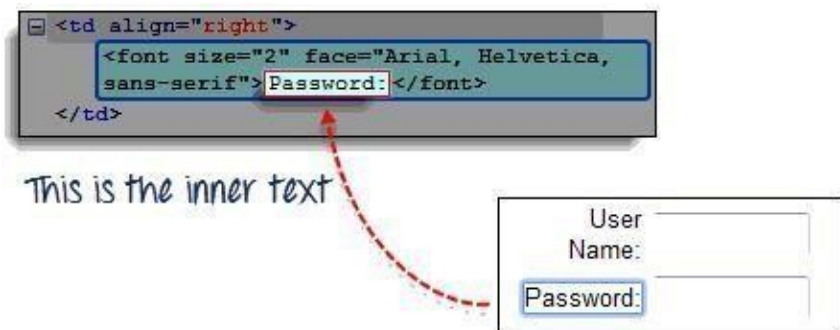
Figure 4.25 Password field element

## v. Locating by CSS Selector - inner text

HTML labels are seldom given id, name, or class attributes. So, how do we access them? The answer is through the use of their inner texts. Inner texts are the actual string patterns that the HTML label shows on the page.

Syntax	Description
<code>css=tag:contains("inner text")</code>	<ul style="list-style-type: none"><li>tag = the HTML tag of the element being accessed</li><li>inner text = the inner text of the Element</li></ul>

Navigate to Mercury Tours' homepage and use Firebug to investigate the "Password" label. Take note of its HTML tag (which is "font" in this case) and notice that it has no class, id, or name attributes.



Type `css=font:contains("Password:")` into Selenium IDE's Target box and click Find. Selenium IDE should be able to access the Password label as shown in the image below.

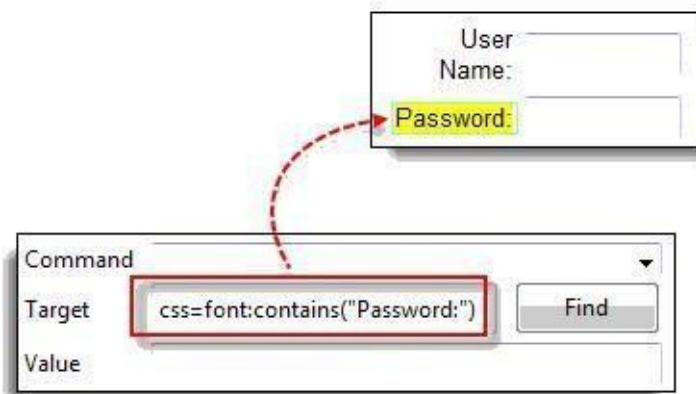


Figure 4.26 Find password element

This time, replace the inner text with "Boston" so that your Target will now become `"css=font:contains("Boston")"`. Click Find. You should notice that the "Boston to San Francisco"

label becomes highlighted. This shows you that Selenium IDE can access a long label even if you just indicated the first word of its inner text.



Figure 4.27 Find element in places

## vi. Locating by DOM (Document Object Model)

The Document Object Model (DOM), in simple terms, is the way by which HTML elements are structured. Selenium IDE is able to use the DOM in accessing page elements. If we use this method, our Target box will always start with "dom=document..."; however, the "dom=" prefix is normally removed because Selenium IDE is able to automatically interpret anything that starts with the keyword "document" to be a path within the DOM anyway.

There are four basic ways to locate an element through DOM:

- b. `getElementById`
- c. `getElementsByName`
- d. `dom:name` (applies only to elements within a named form)
- e. `dom:index`

- **Locating by DOM - `getElementById`**

Let us focus on the first method - using the `getElementById` method. The syntax would be:

Syntax	Description
<code>document.getElementById("id of the element")</code>	id of the element = this is the value of the ID attribute of the element to be accessed. This value should always be enclosed in a pair of parentheses ("").

Use the web page. Navigate to it and use Firebug to inspect the "Keep me logged in" check box. Take note of its ID.



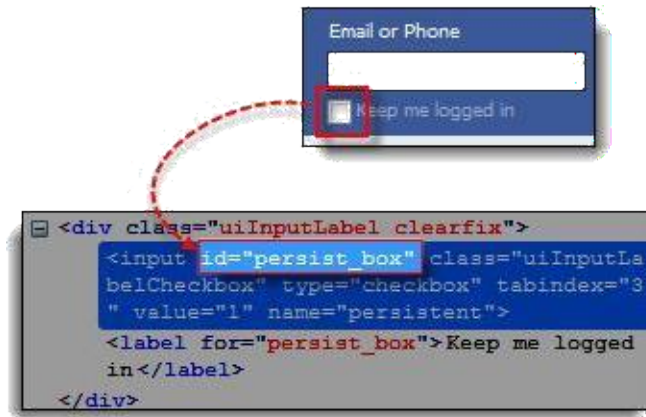


Figure 4.28 DOM inspect id element

Open Selenium IDE and in the Target box, enter "document.getElementById("persist\_box")" and click Find. Selenium IDE should be able to locate the "Keep me logged in" check box. Though it cannot highlight the interior of the check box, Selenium IDE can still surround the element with a bright green border as shown below.



Figure 4.29 Find checkbox element

- **Locating by DOM - getElementsByName**

The getElementById method can access only one element at a time, and that is the element with the ID that you specified. The getElementsByName method is different. It collects an array of elements that have the name that you specified. You access the individual elements using an index which starts.



- **Locating by DOM - dom:name**

As mentioned earlier, this method will only apply if the element you are accessing is contained within a named form.

Syntax	Description
<code>document.forms["<i>name of the form</i>"].elements["<i>name of the element</i>"]</code>	<ul style="list-style-type: none"> <li>• name of the form = the value of the name attribute of the form tag that contains the element you want to access</li> <li>• name of the element = the value of the name attribute of the element you wish to access</li> </ul>

Navigate to Mercury Tours homepage and use Firebug to inspect the User Name text box. Notice that it is contained in a form named "home."

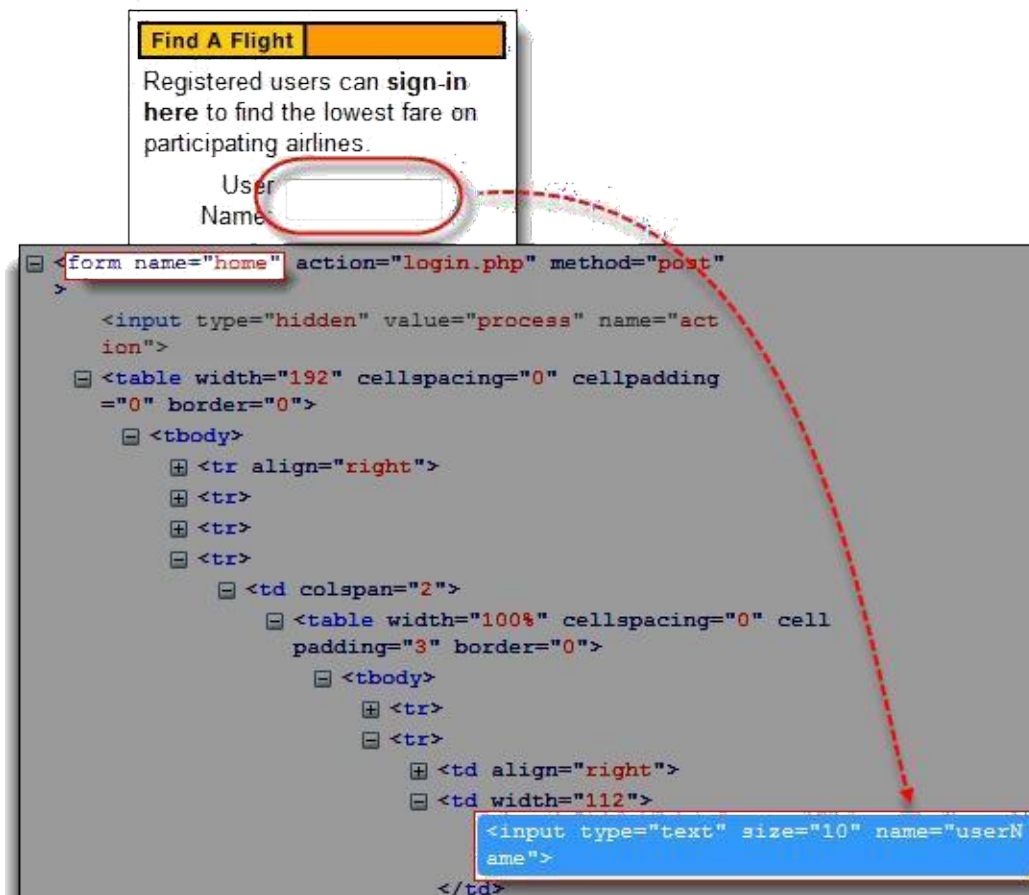


Figure 4.30 Inspect username element

In Selenium IDE, type "document.forms["home"].elements["userName"]" and click the Find button. Selenium IDE must be able to access the element successfully.

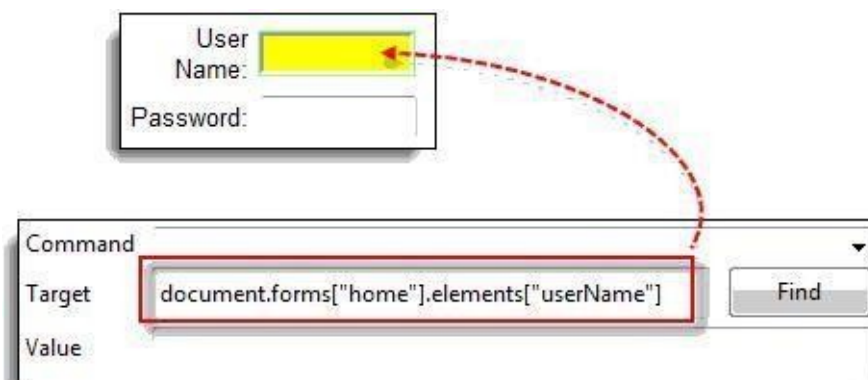


Figure 4.31 Find username element

## **Conclusion**

Agile testing involves testing as early as possible in the software development lifecycle. It demands high customer involvement and testing code as soon as it becomes available. The code should be stable enough to take it to system testing. Extensive regression testing can be done to make sure that the bugs are fixed and tested. Mainly, Communication between the teams makes agile testing success!!!

## **REFERENCES**

1. [www.guru99.com](http://www.guru99.com)
2. [www.softwaretestingmaterial.com](http://www.softwaretestingmaterial.com)
3. [www.softwaretestinghelp.com](http://www.softwaretestinghelp.com)
4. [www.tutorialspoint.com](http://www.tutorialspoint.com)
5. [www.google.co.in](http://www.google.co.in)
6. [www.wikipedia.com](http://www.wikipedia.com)
7. <https://kelltontech.atlassian.net/secure/Dashboard.jspa>
8. <http://qa.kelltontech.net/testlink/login.php>