**Software Development Life Cycle(SDLC):**

* Requirement gathering & Analysis
* Design-> HLD/LLD
* Environment (Development)
* Testing
* Deployment
* Post deployment & Maintenance

**V Model:**

Static Testing Dynamic Testing

Verification Validation

Quality Assurance(QA) Quality Control(QC)

Requirement Acceptance testing (Operational AT/User AT/Alpha/Beta)

High Level Design System testing(End to End)

Low Level Design Integration testing

Development Unit testing

Requirement-> Bussiness Requirement Document(BRD)

High Level Design-> System Requirement Specification/Technical Requirement Document

Low Level Design-> CAD/DFD/UMC/ERD

Development->

**Static testing**-> testing without code

**Types of static testing**:

1. review -> do the work & check

* formal
* informal

1. walkthrough -> presentation given by higher one(like Manager)
2. bench/peer

**Dynamic testing**-> testing with code

**Types of testing**

1. **Functional testing (unit testing level & integration testing level) – Selenium tool**

* Black box testing-> concentrating more on input (all testing)
* White box testing-> going through every line of code(comes only in unit testing)

1. **Non-functional testing ( system testing level)**

* Normal testing
* Regression testing
* Performance testing

1. **UI/UX testing**

**GIT:**

**Remove command**

* git rm -r \*
* git commit -m “message”
* git push -u start main

**.net core architecture**

**3 tier architecture**

**C#**

* Char set: A-Z a-z 0-9
* Keywords: using
* Data types: int, float, double, char, bool, string, byte, short, long
* Reference types: object, String….
* Constants: final int pi=3.14;
* Variables: int x;
* static

**String Manipulation:**

* Equals
* ToUpper()
* ToLower()
* Trim
* TrimEnd
* Substring
* Split

**OOP**

* Class
* Object
* Abstraction
* Encapsulation
* Inheritance: Single, Multi-level, hierarchical
* Polymorphism: Compile time, Run time

**Garbage Collection:**

create remove

constructor destructor

**Arrays:** they aremutable

* **1D**

Declaration:

int[] numbers= new int[3];

int[] numbers = {1,2,3};

* **2D**
* **Jagged Array**

**Readonly:** is only possible for instance variable.

**Interface**

**Generics**

**Collections**

* **Generic collections-** same type(strongly typed)
* List
* Stack (LIFO)
* Queue (FIFO)
* Dictionary
* Sorted list(sequential data), Sorted dictionary(random data), Sorted Set(no duplications allowed)
* **Non-generic collections-** different type
* Array List
* Stack
* Queue
* Hash table
* Sorted list

**Exceptions:**

* System Exception
* User-defined/Custom Exception

**Delegates:**

**3 steps:**

* Declare a delegate

Ex: public delegate void MyDelegate(string msg);

* Set a target method

Ex: MyDelegate myDelegate = new MyDelegate(MethodA);

Or

MyDelegate myDelegate = MethodA

Or set lambda expression

MyDelegate myDelegate =(string msg)=>Console.WriteLine(msg);

* Invoke a delegate

myDelegate.Invoke(“Hello”);

Or

myDelegate(“Hello”);

**LINQ**

**Data Structure**

1. **Linear Data structure**
2. **Static data structure**

* Array

1. **Dynamic data structure**

* Stack
* LIFO
* Queue
* FIFO
* Linked List
* Random memory allocation

**2. Non-linear Data Structure**

* Tree

**Module 2:**

**Requirement Specification Document**

**Table of Contents**

1. Introduction
2. Importance of Understanding Requirements
3. Functional vs Non-Functional Requirements
4. Business Requirement Specification (BRS)
5. Software Requirement Specification (SRS)
6. Conclusion

**Selenium**

**Unit Testing** (All functional requirements comes under this)

**Test scenarios**: they are high level testing features

**Test cases**: written in excel with many columns

**Test Priority**: talks about how fast or important test case is

**Test Severity:** how badly the defect has affected the application’s functionality.

**Unit testing Framework tools:**

**NUnit**

* External tool to dotnet
* **Console application**-> Add nuget packages for existing project in visual studio( package name= NUnit, NUnit3TestAdapter, Selenium.WebDriver , Selenium.WebDriver.ChromeDriver)

**or**

* create directly **NUnit test project** in visual studio and add nuget package=> Selenium.WebDriver, Selenium.WebDriver.ChromeDriver
* Create ConfigSettings folder-> Config.properties text file
* Create CoreCode class
* Create public IWebDriver driver; ->variable
* Create dictionary with type string for both key & value (for reading properties written inside the Config.properties text file
* Create a method to read the Config.properties text file
* Create a method to initialize Chrome browser
* Create a method to close the browser
* **For Data Driven Testing,**
* right click on project-> open folder in file explorer
* inside that create .xlsx file named **InputData** and change the sheet name particular to project
* **Inside project** -> install NuGet package ->ExcelDataReader, ExcelDataReader.DataSet, System.Text.Encoding.CodePages
* Create a 2 classes ExcelData.cs and ExcelUtils.cs
* **Extent Reports**
* install NuGet package -> ExtentReports
* to open generated report-> right click-> open with -> Microsoft edge option
* **Log**
* install NuGet package -> Serilog, Serilog.Sinks.File, Serilog.Sinks.Console

**Annotations:**

**[Test]-** 1 test method

**[TestFixture]-** group of test methods

**[SetUp]-** it will executes immediately before every test is run

**[TearDown]-** it will executes immediately after every test is run

**[OneTimeSetUp]-** runs only once before the child test method

**[OneTimeTearDown]-**

**[Order]-** defines the order that test will run

**[Ignore]-** particular test will be ignored & shows warning message

**Selenium**- it can test only web application (Functional requirement)

**Features:**

* Open source
* Run on any platforms.
* Open for multiple language support
* Cross-browser compatibility

**Components:**

* **Selenium IDE** (Integrated Development Environment)- record & playback tool
* **Selenium RC** (Remote Control)- bridge between selenium commands & browser
* **Selenium Web Driver** -single execution
* **Selenium Grid** -provides parallel execution

**Selenium Web Driver:**

* Create object for web driver (namespace using OpenQA.Selenium)
* driver.Close() – to close single tab
* driver.Quit() – to close multiple tab

**Web Driver Commands:**

* findelement
* findelements

**Locators:**

* id
* name
* class
* tag
* link text
* partial link test
* css selector
* xpath
* Absolute xpath (single slash)
* Relative xpath (double slash)
* Dynamic xpath

**Assertions or Asserts**

1. Hard Assertions- if 1 assert is failed, the remaining will be failed

2. Soft Assertions

Junit- JAVA

Xunit- dotnet

**Test case:**

* Positive
* Negative
* Null
* Field level

**Project:**

1. Test should run independently
2. Test order
3. Dependency
4. Modularized code
5. No hard coding
6. Asynchronous activities (wait) -> write inside TestScripts folder Testclasses
7. Able to handle multiple Windows alerts
8. Data driven testing or (POI) -> done in 2 ways 1)npoi & 2)Excel data reader
9. Page Object Model(POM)
10. Logs -> 1)To maintain security reasons 2) To maintain what is happening all times
11. Screenshots -> before every assert
12. Extent reports ->

**TestNg for Java & Nunit for C#**

**Test Driven Development (TDD) ->** writing test cases in modularized way

**Behavioral Driven Development (BDD) -> (Gherkin language)**

**2 parts in BDD:**

* Writing Feature file (like BRS written in simple English) –> **Gherkin**
* Step definition files-> **SpecFlow in C#** & **Cucumber in java**

**SpecFlow installation->** open visual studio-> click on continue without code-> click on Extension menu-> click on Manage extensions-> select online at left side-> type Specflow in search option-> click on download-> and close the visual studio-> installer will be loaded-> click on modify-> it will install

**Gherkin Syntax:**

Feature: Title of the Scenario

Given [Context]

When [Action]

Then [Outcome]

And

Scenario

**Data Driven Testing (DDT)->** writing data in separate excel sheet & reading from it

**Alerts:**

* Accept means Ok
* Dismiss means Cancel

**Waits->** Used to perform asynchronous operations

**Types:**

* Implicit waits
* Explicit waits or web driver waits => install **DotNetSeleniumExtras.WaitHelpers** NuGet package
* Fluent waits => polling frequency

**Category:**

* Regression testing
* Smoke testing

**Parameterized testing** => done using [TestCase], [TestCaseSource] annotation

**Defect reporting**

* Testing failed test again is called **Retest or confirmation test**

**JavaScript Executor**

* install nuget package **Selenium.JavascriptExecutor**

**Single Page Application (SPA)**

**Arrange Act Assert (AAA)**

**PageFactory ->**should be written inside constructor so that it will optimize the code

**IWebDriver** -> Hiding implementation, we can implement for higher browser

**BDD** (Acceptance testing) **SpecFlow**

Binding with attribute

Binding with tag

Hooks ->script function

Scope binding

**API Postman**

Microservices

Collections =>grouping test

Environment

Tests=> single request

Authorization

**Variables**

* Local variables
* Environmental variables
* Global variables
* Data variables
* Collection variables

**REST** (Representational State Transfer)

RestAssure for Java & RestSharp for C#

**Properties of Rest Architecture:**

* Uniform Interface
* Stateless
* Cacheable
* Client-Server
* Layered System
* Code on Demand

**RestSharp**

* Install NuGet Package => RestSharp
* Parse & handle API responses=> add nuget package -> Newtonsoft.Json