Test Pyramid Approach:

* “Test automation is the practice of running tests automatically, managing test data, and utilizing results to improve software quality. It’s primarily a quality assurance measure, but its activities involve the commitment of the entire software production team.”
* The test pyramid has three distinct sections. Unit tests are at the lowest level, integration tests make up the middle tier, and end-to-end tests form the top. The test pyramid helps developers deliver quality software.
* Mocking frameworks, such as Java’s Mockito or JavaScript’s Sinon, are common tools found in the unit test tier.
* I like to think of end-to-end tests from the perspective of the end user. How would a person interact with my app?
* Our best bet is to remember two things from Cohn's original test pyramid:
  + - Write tests with different granularity
    - The more high-level you get the fewer tests you should have
* If you're working in a functional language a unit will most likely be a single function. Your unit tests will call a function with different parameters and ensure that it returns the expected values. In an object-oriented language a unit can range from a single method to an entire class.
* A unit test class should at least test the public interface of the class. Private methods can't be tested anyways since you simply can't call them from a different test class. Protected or package-private are accessible from a test class (given the package structure of your test class is the same as with the production class) but testing these methods could already go too far.
* Conclusion:
* The test automation pyramid is a framework that defines various types of tests and the number of times they should appear.
* Unit tests form the base of the test pyramid. They should be frequent, and they should run fast.
* Integration tests are the middle tier of the pyramid. These tests focus on interactions of your code with the outside world, such as databases and external services.
* End-to-end tests top the test pyramid. They’re written from the perspective of a user and should test that your entire application is functioning from front end to back end.

Advantages of Unit Testing:

1. Any bugs are found easily and quicker
2. Unit testing saves time and money
3. Unit testing is an Integral part of extreme programming
4. Unit testing provides documentation
5. R2: Reusable and Reliable
6. Unit testing helps gauge performance
7. Unit testing improves code coverage
8. Unit testing reduces code complexity

Benefits of Unit Testing in SDLC:

* 1. Ensuring Every System Components benefits to achieving a quality product
  2. Simplifying the debugging process
  3. Allowing for code refactoring and Design Improvements
  4. Validating correcting bugs early in SDLC
  5. Reducing Bug fixes

Six Rules of Unit Testing

1. Write the test first
2. Never write a test that succeeds the first time
3. Start with the null case, or something that doesn't work
4. Don't be afraid of doing something trivial to make the test work
5. Loose coupling and testability go hand in hand
6. Use mock objects

Limitations of Unit Testing:

1. Unit tests can only show the presence of errors; it cannot show the absence of errors
2. It is unrealistic to test all possible input combinations for any non-trivial piece of software
3. It may not catch integration errors, performance problems, or other system-wide issues. Unit testing is more effective if it is used in conjunction with other software testing activities.
4. A rigorous sense of discipline is needed throughout the software development process. It is essential to keep careful records, not only of the tests that have been performed, but also of all changes that have been made to the source code of this or any other unit in the software. Use of a version control system is essential.

Limitations of Unit Testing:

* Unit Testing can’t help to catch all kind of errors.
* Unit testing should be performed with all other testing types.
* Unit testing! = Integration Testing
* Time- Consuming
* It’s hard to create realistic and useful test

Don’t repeat yourself (DRY) principle states that:

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

1. Inheritance Is Not the Right Tool for Reusing Code

*Polymorphism is the provision of a single interface to entities of different types.*

This is not why we use inheritance in our tests. We use inheritance because it is an easy way to reuse code or configuration. If we use inheritance in our tests, it means that

If we want to ensure that only the relevant code is visible to our test classes, we probably have to create a “complex” class hierarchy because putting everything in one superclass isn’t very “clean”. This makes our tests very hard to read.

Our test classes are in the mercy of their superclass(es), and any change which we make to a such superclass can effect its every subclass. This makes our tests “hard” to write and maintain.

1. Inheritance Can Have a Negative Effect to the Performance of Our Test Suite
2. Using Inheritance Makes Tests Harder to Read

Types Of Listeners In TestNG

TestNG provides a bunch of listeners as a part of its testing environment. These listeners are as follows:

1. ITestListener
2. IReporter
3. ISuiteListener
4. IInvokedMethod
5. IHookable
6. IConfigurationListener
7. IConfigurableListener
8. IAnnotationTransformer
   * ITestAnnotation annotation
   * Class testClass
   * Constructor testConstructor
   * Method testMethod
9. IExecutionListener
10. IMethodInterceptor

Code Coverage:

Top 3 Benefits of Code Coverage:

1. Easy maintenance of code base
2. Exposure of bad code
3. Faster time to market

Code Coverage Percentage = (Number of lines of code executed by a testing algorithm/Total number of lines of code in a system component) \* 100.

5 code coverage criteria

To measure the lines of code that are exercised by test runs, various criteria are taken into consideration. We have outlined below a few critical coverage criteria that companies use.

1. Function Coverage – The functions in the source code that are called and executed at least once.
2. Statement Coverage – The number of statements that have been successfully validated in the source code.
3. Path Coverage – The flows containing a sequence of controls and conditions that have worked well at least once.
4. Branch or Decision Coverage – The decision control structures (loops, for example) that have executed fine.
5. Condition Coverage – The Boolean expressions that are validated and that executes both TRUE and FALSE as per the test runs

The analysis should be clubbed with scalable, robust test scripts, covering every functional and non-functional area of the source code.

Maven Tool: -

1. Source Code => .java
2. Compile => .class
3. Building is a process of .class-> into -> .jar/.war/. ear
   1. Jar: java archive
   2. War: Web archive
   3. Ear: Enterprise archive
4. Library -> Collections of .jar files.

Our Project needs jars. without jars we can’t use technologies in our project.

So, these are called as Dependencies.

Maven-> used to Automate the process of Dependencies. It’s also called as Dependency Management Tool.

Dependency contains details like

1. groupId – Provided Name (Company)

2. artifactId – Jar Name

3. version – version

Maven Project get the jars through POM – Project Object Model .xml will access the Jars in Maven Server (Central repository) to Maven Local repository (.m2) then give it to the Maven Project.

POM.xml

1. Parent Maven Project Details
2. \* Our Project Details
3. Properties
4. \* Dependencies
5. Dependency Management
6. Plugins

Maven Folder System:

*Project Name:*

* src/main/java (.java Files)
* src/main/resources (.xml or. properties)
* src/test/java
* src/test/resources
* src (Web Applications)
  + main
    - webapp
      * WEB-INF
        + Web.xml
* JRE System Library
* pom.xml

Dependency Chain: One jar may be depending on another jar then maven provides all set of jars when we request one jar.

Exclusions: Removing jar from maven dependencies in a chain of jars.

Scope:

* compile -> default scope. a jar is used at compile time and runtime
* provided -> a jar is given by Server/API/Container
* runtime -> a jar used only at runtime
* system -> a jar is loaded from System Drives
* test -> a jar is used only for unit testing

Properties: Hold data in Key Value pairs. In maven properties is used to provide version details.

<properties>

<jbehave.core.version>4.6</jbehave.core.version>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

Few Examples for Properties:-

a. Java Version properties:

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

b. No web.xml Application properties

<failOnMissingWebXml>false</failOnMissingWebXml>

Dependency Management:

bom = Bill Of Materials.

=> It is a pom.xml having all set of possible dependencies with version managed.

=> You can use another pom.xml into your application pom.xml as in BOM format.

=> For one f/w or technologies related jars are given as on BOM.

ex: spring-bom, jersey-bom..etc

=> If just we add bom(pom.xml) as <dependencyManagement> in our pom.xml

it will not provide those dependencies to our Build path.

--Ex--

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.core</groupId>

<artifactId>jersey-common</artifactId>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-framework-bom</artifactId>

<version>5.2.6.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<dependency>

<groupId>org.glassfish.jersey</groupId>

<artifactId>jersey-bom</artifactId>

<version>2.25.1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

\*) parent project:

It is a maven project used to provide common setup for all/multiple

child projects (it is like inheritance).

=> Ex: common properties, common dependencies, dependency management, plugin

management etc..

--Steps to create parent projects--

> File > new > Maven Project

> choose checkbox [v] Create simple maven project

> next > Enter details

groupId : in.nareshit.parent

artifactId : MyTestParent

version : 3.2.5

package : pom (must be pom to behave as parent)

> once it is created then add below details in pom.xml of parent project

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.2.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.4.15.Final</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.46</version>

</dependency>

</dependencies>

</dependencyManagement>

--Steps to use parent in child project--

> File > new > Maven Project

> choose checkbox [v] Create simple maven project

> next > Enter details

groupId : in.nit.child

artifactId : MyTestChild

version : 1.2.1

> Next > Finish

> open our project (child project) pom.xml file and add details like

<parent>

<groupId>[parent project groupId]</groupId>

<artifactId>[parent project artifactId]</artifactId>

<version>[parent project version]</version>

</parent>

*Maven archetypes* are project templates which can be generated for your by Maven. In other words, when you are starting a new project you can generate a template for that project with Maven. In Maven a template is called an *archetype*. Each Maven archetype thus corresponds to a project template that Maven can generate.

-----------------------------------------------------------------

Maven Life Cycle:

1. Build Life Cycle: executing set of tasks to convert our code into one final format.

Phase: Phase indicates task done by maven.

3 Types:

1. Clean
2. \*\* Default
3. Site (Document & Website)

\*) Phase is executed by Goals. -> Plugin: goal

\*) If we try to execute one phase (Task) then phases which are connected before in order also executed.

\*) Phase is like a concept. Goal is like a implementation logic

|  |  |
| --- | --- |
| Phase | Goal |
| Package | Jar: jar |

Q) Compile, Test – Compile and Test Phase?

Compile: it will execute goal compile: only on src/main/java

Test-compile: execute goal compile: test-compile: only on src/test/java and src/main/java. Also executes al j Unit test cases. You should have compiler plugin.

Q) jar meta - data?

A) jar information like who created this jar by using which version of JDK which version of jar plugin who is the author…

TestNG:

1. Introduction

TestNG is a testing framework designed to simplify a broad range of testing needs, from unit testing (testing a class in isolation of the others) to integration testing (testing entire systems made of several classes, several packages and even several external frameworks, such as application servers).

1. Annotations:

Execution order of TestNG Annotations:

* Before Suite
  + Before Test
    - Before Class
      * Before Method
        + Test
      * After Method
    - After Class
  + After Test
* After Suite

|  |  |  |
| --- | --- | --- |
| @Test | | Marks a class or a method as part of the test. |
|  | alwaysRun | If set to true, this test method will always be run even if it depends on a method that failed. |
|  | dataProvider | The name of the data provider for this test method. |
|  | dataProviderClass | The class where to look for the data provider. If not specified, the data provider will be looked on the class of the current test method or one of its base classes. If this attribute is specified, the data provider method needs to be static on the specified class. |
|  | dependsOnGroups | The list of groups this method depends on. |
|  | dependsOnMethods | The list of methods this method depends on. |
|  | description | The description for this method. |
|  | enabled | Whether methods on this class/method are enabled. |
|  | expectedExceptions | The list of exceptions that a test method is expected to throw. If no exception or a different than one on this list is thrown, this test will be marked a failure. |
|  | groups | The list of groups this class/method belongs to. |
|  | invocationCount | The number of times this method should be invoked. |
|  | invocationTimeOut | The maximum number of milliseconds this test should take for the cumulated time of all the invocationcounts. This attribute will be ignored if invocationCount is not specified. |
|  | priority | The priority for this test method. Lower priorities will be scheduled first. |
|  | successPercentage | The percentage of success expected from this method |
|  | singleThreaded | If set to true, all the methods on this test class are guaranteed to run in the same thread, even if the tests are currently being run with parallel="methods". This attribute can only be used at the class level and it will be ignored if used at the method level. Note: this attribute used to be called sequential (now deprecated). |
|  | timeOut | The maximum number of milliseconds this test should take. |
|  | threadPoolSize | The size of the thread pool for this method. The method will be invoked from multiple threads as specified by invocationCount. Note: this attribute is ignored if invocationCount is not specified |

1. TestNG.xml:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite1"* verbose=*"1"*>

<test name=*"Nopackage"*>

<classes>

<class name=*"NoPackageTest"* />

</classes>

</test>

</suite>

Or directlty we can mention Package name itsel:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite1"* verbose=*"1"*>

<test name=*"Regression1"*>

<packages>

<package name=*"test.sample"* />

</packages>

</test>

</suite>

You can also specify groups and methods to be included and excluded:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite1"* verbose=*"1"*>

<test name=*"Regression1"*>

<groups>

<run>

<exclude name=*"brokenTests"* />

<include name=*"checkInTests"* />

</run>

</groups>

<classes>

<class name=*"test.IndividualMethodsTest"*>

<methods>

<include name=*"testMethod"* />

</methods>

</class>

</classes>

</test>

You can also define new groups inside testng.xml and specify additional details in attributes, such as whether to run the tests in parallel, how many threads to use, whether you are running JUnit tests, etc...

By default, TestNG will run your tests in the order they are found in the XML file. If you want the classes and methods listed in this file to be run in an unpredictable order, set the preserve-order attribute to false

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name=*"Suite1"* verbose=*"1"*>

<test name=*"Regression1"* preserve-order=*"false"*>

<classes>

<class name=*"test.Test1"*>

<methods>

<include name=*"m1"* />

<include name=*"m2"* />

</methods>

</class>

<class name=*"test.Test2"* />

</classes>

</test>

</suite>

1. Running TestNG

TestNG can be invoked in different ways:

1. Command line
2. Ant
3. Eclipse
4. IntelliJ's IDEA

1. Test methods, Test classes and Test groups

Methods that are annotated with @Test that happened to return a value will be ignored unless you set *allow-return-values* to *true* in your testng.xml:

<suite allow-return-values=*"true"*>

or

<test allow-return-values=*"true"*>

You can define groups at the class level and then add groups at the method level:

Dependencies:

There are two kinds of dependencies:

* + 1. Hard dependencies. All the methods you depend on must have run and succeeded for you to run. If at least one failure occurred in your dependencies, you will not be invoked and marked as a SKIP in the report.
    2. Soft dependencies. You will always be run after the methods you depend on, even if some of them have failed. This is useful when you just want to make sure that your test methods are run in a certain order but their success doesn't really depend on the success of others. A soft dependency is obtained by adding "alwaysRun=true" in your @Test annotation.

Note: as stated before, the order of invocation for methods that belong in the same group is not guaranteed to be the same across test runs.

If a method depended upon fails and you have a hard dependency on it (alwaysRun=false, which is the default), the methods that depend on it are not marked as FAIL but as SKIP. Skipped methods will be reported as such in the final report (in a color that is neither red nor green in HTML), which is important since skipped methods are not necessarily failures.

N5.7.2 - Dependencies in XML

Alternatively, you can specify your group dependencies in the testng.xml file. You use the <dependencies> tag to achieve this:

<test name=*"My suite"*>

<groups>

<dependencies>

<group name=*"c"* depends-on=*"a b"* />

<group name=*"z"* depends-on=*"c"* />

</dependencies>

</groups>

</test>

The <depends-on> attribute contains a space-separated list of groups

Ignoring tests: -

TestNG lets you ignore all the @Test methods:

* In a class (or)
* In a particular package (or)
* In a package and all its child packages

The @Ignore annotation has a higher priority than individual @Test method annotations. When @Ignore is placed on a class, all the tests in that class will be disabled.

To ignore all tests in a particular package, you just need to create package-info.java and add the @Ignore annotation to it. Here's a sample:

@Ignore

package com.testng.master;

import org.testng.annotations.Ignore;

This causes all the @Test methods to be ignored in the package com.testng.master and all of its sub-packages.

Parallel:

<suite name=*"My suite"* parallel=*"methods"* thread-count=*"5"*>

<suite name=*"My suite"* parallel=*"tests"* thread-count=*"5"*>

<suite name=*"My suite"* parallel=*"classes"* thread-count=*"5"*>

<suite name=*"My suite"* parallel=*"instances"* thread-count=*"5"*>

* parallel="methods": TestNG will run all your test methods in separate threads. Dependent methods will also run in separate threads, but they will respect the order that you specified.
* parallel="tests": TestNG will run all the methods in the same <test> tag in the same thread, but each <test> tag will be in a separate thread. This allows you to group all your classes that are not thread safe in the same <test> and guarantee they will all run in the same thread while taking advantage of TestNG using as many threads as possible to run your tests.
* parallel="classes": TestNG will run all the methods in the same class in the same thread, but each class will be run in a separate thread.
* parallel="instances": TestNG will run all the methods in the same instance in the same
* thread, but two methods on two different instances will be running in different threads.

You can also specify that a @Test method should be invoked from different threads. You can use the attribute threadPoolSize to achieve this result:

|  |  |
| --- | --- |
| @Test(threadPoolSize = 3, invocationCount = 10,  timeOut = 10000) | |
| public void testServer() { |

In this example, the function testServer will be invoked ten times from three different threads. Additionally, a time-out of ten seconds guarantees that none of the threads will block on this thread forever.

Running TestNG programmatically

You can invoke TestNG from your own programs very easily:

TestListenerAdapter tla = new TestListenerAdapter();

TestNG testng = new TestNG();

testng.setTestClasses(new Class[] { Run2.class });

testng.addListener(tla);

testng.run();

Dependency Injection:-

TestNG supports two different kinds of dependency injection: native (performed by TestNG itself) and external (performed by a dependency injection framework such as Guice).

The below table summarises the parameter types that can be natively injected for the various TestNG annotations:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Annotation | ITestContext | XmlTest | Method | Object[] | ITestResult |
| BeforeSuite | Yes | No | No | No | No |
| BeforeTest | Yes | Yes | No | No | No |
| BeforeGroups | Yes | Yes | No | No | No |
| BeforeClass | Yes | Yes | No | No | No |
| BeforeMethod | Yes | Yes | Yes | Yes | Yes |
| Test | Yes | No | No | No | No |
| DataProvider | Yes | No | Yes | No | No |
| AfterMethod | Yes | Yes | Yes | Yes | Yes |
| AfterClass | Yes | Yes | No | No | No |
| AfterGroups | Yes | Yes | No | No | No |
| AfterTest | Yes | Yes | No | No | No |
| AfterSuite | Yes | No | No | No | No |

5.22 - Altering suites (or) tests

Sometimes you may need to just want to alter a suite (or) a test tag in a suite xml in runtime without having to change the contents of a suite file.

A classic example for this would be to try and leverage your existing suite file and try using it for simulating a load test on your "Application under test". At the minimum you would end up duplicating the contents of your <test> tag multiple times and create a new suite xml file and work with. But this doesn't seem to scale a lot.

TestNG allows you to alter a suite (or) a test tag in your suite xml file at runtime via listeners. You achieve this by providing a listener that implements IAlterSuiteListener. Please refer to Listeners section to learn about listeners.

##### - TestNG Exit Codes

When TestNG completes execution, it exits with a return code.  
This return code can be inspected to get an idea on the nature of failures (if there were any).  
The following table summarises the different exit codes that TestNG currently uses.

| FailedWithinSuccess | Skipped | Failed | Status Code | Remarks |
| --- | --- | --- | --- | --- |
| No | No | No | 0 | Passed tests |
| No | No | Yes | 1 | Failed tests |
| No | Yes | No | 2 | Skipped tests |
| No | Yes | Yes | 3 | Skipped/Failed tests |
| Yes | No | No | 4 | FailedWithinSuccess tests |
| Yes | No | Yes | 5 | FailedWithinSuccess/Failed tests |
| Yes | Yes | No | 6 | FailedWithinSuccess/Skipped tests |
| Yes | Yes | Yes | 7 | FailedWithinSuccess/Skipped/Failed tests |

### 9 - JVM Arguments in TestNG

| JVM Argument | Comment | Default value |
| --- | --- | --- |
| testng.thread.affinity | A Boolean indicating whether TestNG should resort to running dependent methods on the same thread as the upstream methods. | false |
| testng.mode.dryrun | A Boolean indicating whether TestNG should simulate a real execution. In this mode the test methods are not actually executed. | false |
| testng.test.classpath | A String that represents a list of zip files or jars that need to be added to the TestNG classpath for it to retrieve test classes for execution. | "" |
| skip.caller.clsLoader | A Boolean indicating whether TestNG should skip using the current ClassLoader for loading classes. | false |
| testng.dtd.http | A Boolean indicating whether TestNG should load DTDs from http endpoints. | false |
| testng.show.stack.frames | A Boolean indicating whether TestNG should show detailed stack traces in reports. | false |
| testng.memory.friendly | A Boolean indicating whether TestNG should be memory cognizant and use light weight test method representations. | false |
| testng.strict.parallel | A Boolean indicating that TestNG should attempt to start all test methods simultaneously when there are more than one test tags and parallelism has been set to methods. | false |
| emailable.report2.name | A String indicating the file name into which the emailable reports are to be written into. | emailable-report.html |
| oldTestngEmailableReporter | A Boolean indicating whether TestNG should use the old emailable report listener for building simple html emailable report. | false |
| noEmailableReporter | A Boolean indicating whether TestNG should use the NEW emailable report listener for building simple html emailable report. | true |
| testng.report.xml.name | A String indicating the file name into which the xml reports are to be written into. | testng-results.xml |
| fileStringBuffer | A Boolean indicating whether TestNG should output verbose logs when working with very large text data. | false |
| stacktrace.success.output.level | A String indicating the log levels to be included in the XML Reports (Valid values include : NONE (no stacktraces), SHORT (short stacktrace), FULL (full stacktrace), BOTH (both short and full stacktrace). | FULL |

Requirement Elaboration – Intermediate:

3 Amigo’s:

Three amigos refer to the primary perspectives to examine an increment of work before, during, and after development. Those perspectives are:

1. Business – What problem are we trying to solve?
2. Development – How might we build a solution to solve that problem?
3. Testing – What about this, what could possibly happen?

Also Known As

Story Kick Off huddles or the Triad

Expected Benefits:

* Builds A Shared understanding about the intent of an incremental of work.
* Identifies misunderstanding and confusion early and allows learning.
* Provides a reasonable guard rail for the number of people who should be involved in discussions about any given increment of work to happen sooner in the delivery of an incremental work

Backlog Grooming:

Synonyms: *Backlog Refinement, Story Time*

Objective:

1. Break down large user stories into smaller tasks
2. Confirm your team is on the same page
3. Meeting the Definition of Ready
4. Assign Story Points

4 Key Responsibilities:

* 1. Scheduling the Grooming the session
  2. Keep the meeting on track
  3. Move the conversation forward
  4. Send follow up emails

Meeting Requirements:

1. Facilitator
2. Product Owner
3. Delivery Team
4. QA Representative

Session should be 45mins. To 1 Hr.

Backlog Grooming Session Prep:

* Review objectives
* Talk to stakeholders
* Revisit key metrics

Outcome of grooming session: -

1. We should be having a prioritised list of user stories.
2. Any larger stories at the top should be broken down into smaller, more manageable tasks.

Acceptance Criteria:

* Are we building the right thing?
* Did we build it right?

Effectiveness:

* Testable
* Clear and Concise
* Easy to understand
* Provides end users perspective

Benefits:

* It helps you manage expectations
* Define Scope & Reduce the ambiguity
* Establish testing criteria for QA
* Defend against scope creep mid sprint

Software Requirement Specification (SRS):

1. Correctness of Software Requirement Specification should be checked.
2. Ambiguity should be avoided
3. Requirements should be complete
4. Consistent Requirements
5. Verification and expected result
6. Testing Environment
7. Pre – conditions defined clearly
8. Requirement Id’s
9. Security and Performance criteria
10. Assumptions should be avoided
11. Deletion of irrelevant requirements
12. Freezing the requirements

Requirement Analysis Techniques:

From What to How: Software engineering task bridging the gap between system requirements engineering and software design.

3 Orthogonal Views: Provides software designer with a model of:

1. System information (static view)
2. Function (functional view)
3. Behaviour (dynamic view)

Software Architecture: Model can be translated to data, architectural, and component-level designs.

Iterative and Incremental Process: Expect to do a little bit of design during analysis.

we need ways to accurately capture, interpret, and represent the voice of customers when specifying the requirements for a software product.

Activities for Requirement Analysis: -

1. Eliciting Requirements (Requirements gathering)
2. Analysing Requirements
3. Requirements modelling
4. Review and Retrospective

Business Requirement vs Software Requirements

Business Requirements

Business requirements relate to a business' objectives, vision, and goals. They also provide the scope of a business need or problem that needs to be addressed through a specific activity or project. For example, if a trade association has an objective to promote the services offered by its members, the business requirements for a project might include creating a member directory that increases awareness of members. Good business requirements must be:

* Clear and are typically defined at a very high level.
* Provide enough information and guidance to help ensure that the project fulfils the identified need.
* Understanding an organization's mandate, objectives or goals, a specific business need or problem that is being tackled
* Should be clearly defined and understood before developing business requirements.
* The need or problem can relate to the organization or business in general or focus on a stakeholder group, such as customers, clients, suppliers, employees or another group.

Software Requirements

Software requirements break-down the steps needed to meet the business requirement or requirements.

*Whereas a business requirement states the 'why' for a project, a software requirement outline the 'what'.*

Techniques for Discovering Business Needs

1. Gap Analysis Using BPMN or ArchiMate
2. ArchiMate - Gap Analysis
3. BPMN - As-is and To-be Analysis

As-is Process

1. To-be Process

Use Cases

Use cases specify the expected behaviour (what), and not the exact method of making it happen (how). Use cases once specified can be denoted both textual and visual representation (such as UML). A key concept of use case modelling is that it helps us design a system from end user's perspective. It is an effective technique for communicating system behaviour in the user's terms by specifying all externally visible system behaviour.

* A use case diagram is usually simple. It does not show the detail of the use cases.
* It only summarizes some of the relationships between use cases, actors, and systems.
* It does not show the order in which steps are performed to achieve the goals of each use case.

User Stories

A user story is a note that captures what a user does or needs to do as part of his/her work. Each user story consists of a short description written from user's point of view, with natural language. Unlike the traditional requirement capturing, user story focuses on what the user need instead of what the system should deliver. This leaves room for further discussion of solutions and the result of a system that can really fit into the customers' business workflow, solving their operational problems, and most importantly adding value to the organization. User stories are well compatible with the other agile software development techniques and methods, such as scrum and extreme programming.

Test Case Design Techniques to Ensure High-Quality Software:

Test Case Design Technique is broadly classified into 3 parts:

1. Static Testing Design Technique
   * Manual
     + Walk through – Authors – Document
     + Informal review – Not Documented
     + Technical Review – Technical Expert – Peer Review
     + Audit – External Agent – All Documents
     + Inspection – Trained Moderator
     + Management Review – Top Management
   * With the help of tools
     + Analysis of Coding Standards
     + Analysis of Code Metrics
     + Analysis of Code Structure
       - Control Flow Structure
       - Data Flow Structure
       - Data Structure
2. Dynamic Test Design Technique
   * Specification based technique (Black Box Testing)
     + BVA
     + EP
     + Decision Table: 2 ^ n
     + State Transition
     + Use Case
   * Structure based technique (White Box Testing)
     + Statement Testing & Coverage
     + Decision Testing & Coverage
     + Condition Testing
     + Multiple Condition Testing
     + All path testing
   * Experience based technique
     + Error Guessing
     + Exploratory testing

A Use Case is as a tool for defining the required user interaction and if you are trying to create a new application or make changes to an existing application, several discussions are made.

Use Case Testing is generally a part of a black box testing and that helps developer and testers to identify test scenarios that exercise the whole system on each transaction basis from start to finish. Business experts and developers must have a mutual understanding of the requirement, as it’s very difficult to attain. Use case testing is a functional testing technique which helps in identifying and test scenario on the whole system or doing start to end transactions.

Feature of Use Case Testing:  
There is some feature of a use case testing, which is used to test the software project and provide a better response, these are given below:

* Use case testing is not testing that is performed to decide the quality of the software.
* Although it is a type of end to end testing, it won’t ensure the entire coverage of the user application.
* Use Cases has generally captured the interactions between ‘actors’ and the ‘system’.
* ‘Actors’ represents the user and their interactions that each user takes part in.
* The use case will find out the defects in integration testing.

Benefits of Use Case Testing:  
Use case testing provide some functionality which is used to help to develop a software project. These are given below:

* use case driven analysis is that it helps manage complexity since it focuses on one specific usage aspect at a time.
* Use cases start from a very simple view that a system is built first and foremost for its users.
* Use cases are a sequence of steps that describe the interactions between the actor and the system.
* Use case help to capture the functional requirements of a system.
* Use cases are used to hearten designers to outcomes before attempting to specify outcomes, and thereby they help to make requirements more proactive in system development.

3 simple yet effective tricks to make exploratory testing effective

* Use RIMGEA
  + Replicate it first
  + Isolate it
  + Maximize it
  + Generalize it
  + Externalize it
  + Ans say it clearly and dispassionately
* Keep it simple and yet comprehensive
* Employ cross functional pair testing

Test Case:

Is a Documentation which specifies input values, expected output and the pre-condition for executing the Test.

IEEE:

A set of test inputs, execution conditions, and expected results developed for a particular objective. Such as to exercise a particular program path or to verify compliance with a specific requirement.

Goals:

1. Plan
2. Test Before Execution
3. Accelerate the regression
4. Pass info to new testers
5. Satisfy customer requirements
6. Store Information
7. Track Testing InProgress

Outcome of TC:

1. How many user scenarios passed/failed?
2. How many features are stable?
3. Does the application confirm to the specification?
4. Is the software ready to deliver or not?
5. Which features need to rework?
6. What is the quality of the application?

Test Case: are a set of valid and invalid steps of Test Scenario.

Test Scenario: are the high-level classification of test requirements grouped by module functionality.

Checklist: Is a list of statements that we ned to test. In short, it’s “what to test”.

Test Case Fields: -

1. Pre-Condition
2. Title/Summary
3. Module to be tested
4. Description
5. Test steps
6. Priority
7. Expected Results

Software Estimation Techniques:

1. Planning poker
2. Story Points:

Story points are a unit of measure for expressing an estimate of the overall effort that will be required to fully implement a product backlog item or any other piece of work.

When we estimate with story points, we assign a point value to each item. The raw values we assign are unimportant. What matters are the relative values. A story that is assigned a 2 should be twice as much as a story that is assigned a 1. It should also be two-thirds of a story that is estimated as 3 story points.

* Amount of work you do
* Complexity of work
* Risk and uncertainty

1. Delphi Technique
2. Work Breakdown Structure (WBS)
3. Three Point Estimation
   1. Optimistic Estimate (A)
   2. Most Likely Estimate (M)
   3. Pessimistic Estimate (B)

Value = [ A + (4 \* M ) + B ] / 6

SD = [ B – A ] / 6

1. Functional Point Method

Total Effort Estimate = Total Function Points \* Estimate defined per Functional Point

How to Select Correct Test Cases For Automation Testing (And Ultimately Achieve A Positive Automation ROI):

(ROI – Return on Investment – It is a calculation of benefits in terms of cost-saving, increased efficiency, and quality)

Parameters for Automation Candidate:

* 1. TC executed with different set of data
  2. TC executed with different browsers
  3. TC executed with different environments
  4. TC executed with complex business logic
  5. TC executed with different set of users
  6. TC involves a large amount of data
  7. TC has any dependency
  8. TC requires special data

Attributes which form the basis for determining the ROI:

* Purchasing and licensing cost of the tool
* Time to develop the scripts
* Time to analyse the results manually and automatically
* Time and cost to train the resources
* Management overheads

Manual To Automation Testing – What Are The Process Challenges?

1. Automation Need: -
2. Automating the complete application
3. Manual Vs Automation mentality: “We normally automate which is important and repetitive, but we prefer to test the important functionality manually”.
4. Deciding on the framework
5. Knowledge of the team
6. Reporting
7. Trust

Use these top tips to ensure that your software testing is successful and you get the maximum return on investment (ROI):

1. Decide what Test Cases to Automate
2. Select the Right Automated Testing Tool
3. Divide your Automated Testing Efforts
4. Create Good, Quality Test Data
5. Create Automated Tests that are Resistant to Changes in the UI

## Tests that should be automated:

* Tests that need to be run against every build/release of the application, such as smoke test, sanity test and regression test.
* Tests that utilize the same workflow but different data for its inputs for each test run (data-driven and boundary tests).
* Tests that need to gather multiple information during runtime, like SQL queries and low-level application attributes.
* Tests that can be used for performance testing, like stress and load tests
* Tests that take a long time to perform and may need to be run during breaks or overnight. Automating these tests maximizes the use of time.
* Tests that involve inputting large volumes of data.
* Tests that need to run against multiple configurations — different OS & Browser combinations, for example.
* Tests during which images must be captured to prove that the application behaved as expected.

Important: Remember that the more repetitive the test run, the better it is for automation

## Tests that should not be automated:

* User experience tests for usability (tests that require a user to respond as to how easy the app is to use).
* Tests that you will only run one-time. (This is a general rule. I have automated one–time tests for data population situations in which the steps can be automated quickly, and when placing in a loop can produce thousands of records, saving a manual tester considerable time and effort)
* Test that need to run ASAP.
* Tests that require ad hoc/random testing based on domain knowledge/expertise.
* Tests without predictable results. For automation validation to be successful, it needs to have predictable results in order to produce pass and fail conditions.
* If a test needs to be manually “eyeballed” to determine whether the results are correct.
* Test that cannot be 100% automated should not be automated at all — unless doing so will save a considerable amount of time.

What should a test case be automated?

1. Repetitive Test Runs
2. High-Risk Test Cases
3. Critical Parts of A Web Application
4. Extensive Tests
5. Evaluate the Pros and Cons
6. Three Golden Tests
7. If You Can Answer the Magical Question “Why?”
8. Complex Cases
9. Long Testing
10. Test has significant downtime between steps

Life cycle of Software Test Automation

Life Cycle in layman definition is the series of steps/changes that happens in a process to get the desired result.

Life cycle denotes the subsequent actions in a timely fashion. Below are the lists of things to be considered for test automation life cycle.

1. Feasibility study
2. Tool identification
3. Framework identification
4. Proof of concept
5. Test scripts design
6. Test scripts execution
7. Test scripts review

Tips to Improve Test Automation Effectiveness & ROI:

Use these top tips to ensure that your software testing is successful, and you get the maximum return on investment (ROI):

* Decide what Test Cases to Automate
  + Repetitive tests that run for multiple builds.
  + Tests that tend to cause human error.
  + Tests that require multiple data sets.
  + Frequently used functionality that introduces high risk conditions.
  + Tests that are impossible to perform manually.
  + Tests that run on several different hardware or software platforms and configurations.
  + Tests that take a lot of effort and time when manual testing.
* Select the Right Automated Testing Tool
* Divide your Automated Testing Efforts
* Create Good, Quality Test Data
* Create Automated Tests that are Resistant to Changes in the UI

Let’s break it down...

The 4 Agile Values:

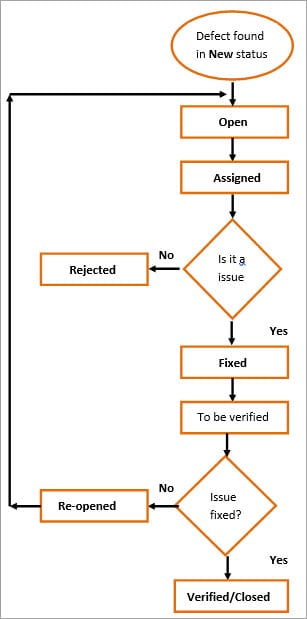
* Individuals and interactions over processes and tools.
* Working software over comprehensive documentation.
* Customer collaboration over contract negotiation.
* Responding to change over following a plan.

The 12 Agile Principles:

1. The highest priority is to satisfy the customer through early, and continuous, delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Clients and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to, and within a development team, is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development — the sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity — the art of maximizing the amount of work not done — is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Goals of Defect Management Process (DMP):

1. Prevent the Defect
2. Early Detection
3. Minimize the impact
4. Resolution of the Defect
5. Process Improvement



Defect Management Process:

* 1. Defect prevention
     1. Identify critical risk
        + 1. Missing a key requirement
          2. Critical application software that does not function properly
          3. Vendor supplied software does not function properly
          4. Performance is unacceptably poor
          5. Hardware malfunction
          6. Hardware and/or software does not integrate properly
          7. Hardware new to installation site
          8. Hardware not delivered on-time
          9. Users unable or unwilling to embrace new system
          10. User's inability to actively participate in project
          11. Etc.
     2. Estimate expected impact
     3. Minimize the expected impact
  2. Deliverable baseline
  3. Defect discovery:
     1. Find a defect
     2. Report a defect
     3. Acknowledge a defect
  4. Defect Resolution
     1. Prioritize the risk
     2. Fix the defect’
     3. Report the resolution
  5. Process improvement

Techniques to Minimize Impact:

* *Quality Assurance*
* *Training and Education (work force)*
* *Training and Education (customers)*
* *Methodology and Standards*
* *Defensive Design*
* *Defensive Code*

Bug Report Details:

* Summary
* Severity
* Priority
* Description
* Steps to Description
* Actual and Expected Results
* Attachment
* *Optional Fields:*
  + Issue Types
  + Product component
  + Found/Fix Version
  + Environments
  + Sprint/Release

6 Ways to Make Test Automation Successful in Agile Environment:

1. Conduct Parallel Testing
2. Develop Quality Test Scripts
   1. Correctness
   2. Versioning
   3. Maintainability
   4. Portability
   5. Integrity
   6. Performance
3. Adopt DevOps
4. Get the right Automation Tool
   1. The tool should be OS friendly in that the number of operating systems it supports, the better it is for teams to perform testing.
   2. It should be capable of meeting end-to-end testing requirements.
   3. The tool should be intuitive, meaning it should come with easy-to-understand UI, navigation, and functionality. The user-friendliness of the tool will determine how easily a user can get up and ready with test automation.
   4. The tool should offer integration support with cross-functional tools like bug tracking tools, test management tools, continuous integrations, etc.
   5. It should be within the company’s budget and should offer features as per its pricing.
   6. The tool you choose should be robust and provide quick results so that testers can respond to the test results quickly
5. Keep a check on Development Environment
6. Keeps your tests small and & lean

**Challenges in implementing In-sprint test automation:**

1. The biggest challenge is finding a right tool or a test automation framework that allows for in-sprint test automation
2. High quality test automation engineers conversant with multiple frameworks and scripting languages to own the automation agility to achieve In-Sprint Automation

Here are five key considerations to successfully achieve in-sprint test automation.:

1. Developers must check in the code periodically with in the sprint
2. Automation engineers should be a part of the development scrum team.
3. Automation engineers are well trained and hard to find.
4. Written test cases should be descriptive, concise and easier to understand.
5. Pick the right test automation framework and tool that can aid in in-sprint automation.

Test Strategy Document:

1. Scope and Objectives
2. Business issues
3. Roles and responsibilities
4. Communication and status reporting
5. Test deliverables
6. Industry standards to follow
7. Test automation and tools
8. Testing measurements and metrices
9. Risks and mitigation
10. Defect reporting and tracking
11. Change and configuration management
12. Training plan

Test Strategy: <BRS

1. Scope & Overview
   1. What to test and why to test
2. Test Approach
   1. Test Levels
   2. Test Types
   3. Roles & Responsibilities
   4. Environment Requirements
3. Testing Tools
4. Industry Standards to follow
5. Test Deliverables
6. Testing Metrics
7. RTM
8. Risk & Mitigation
9. Reporting Tool
10. Test Summary

Test Plan Document: <SRS

Components of the Test Plan document

1. Test Plan id
2. Introduction
3. Test items
4. Features to be tested
5. Features not to be tested
6. Test techniques
7. Testing tasks
8. Suspension criteria
9. Features pass or fail criteria
10. Test environment (Entry criteria, Exit criteria)
11. Test deliverables
12. Staff and training needs
13. Responsibilities
14. Schedule

Test Plan:

1. Test Plan Identifier
2. Introduction
3. Test Items
4. Software Risk Issues
5. Features to be tested
6. Features not to be tested
7. Approach
8. Item Pass/Fail Criteria (or) Acceptance Criteria
9. Suspension Criteria and Resumption Requirements
10. Test Deliverables
11. Test Tasks
12. Environmental Requirements
13. Staffing and Training needs
14. Responsibilities
15. Schedule
16. Approvals

### Test plan

Think of the test plan as a super document that lists everything that will be required in the project. According to Software Testing Help, this will often include all of the activities to be executed, the scope, roles, entry and exit criteria and test objectives. This deliverable acts as a roadmap for teams to follow and measure their progress against. It's a critical part in knowing how they are performing and if projects are staying on track with scheduling and user expectations. Test plan will also include features that need to be tested, testing tools and environment requirements, which are all essential to plan out prior to executing any [testing processes.](http://searchsoftwarequality.techtarget.com/answer/Software-testing-processes-and-development-methodologies)

### Test strategy

The finer details of testing are included in the test strategy. This deliverable is often a part of the test plan, and outlines the testing approach that will be used to fulfill quality standards. The test strategy gives clear direction as to what types of tests will be used, but leaves some room to ensure that anyone is able to execute the test at any time. This type of information will give clear direction about what tasks need to be completed and allows teams to determine what tools will be the most effective in helping to achieve their goals.

### Test case

Unlike test plan and test strategy, a test case is far more specific. It's a sequence of steps that helps teams perform a test in the project. This document often includes conditions, environment, expected results, actual results and whether it passed or failed, according to Software Testing Guide. In many cases, an [enterprise test management software](https://smartbear.com/test-management/zephyr-enterprise/) can manage these elements and keep track of testing progress.

Testing processes have a number of deliverables that are essential, but many organizations are unable to utilize them effectively due to misunderstanding about what each asset entails. With the knowledge of the differences between these terms, quality assurance teams can create better plans and ensure that they are providing the right guidance to meet stakeholder objectives.

|  |  |
| --- | --- |
| **Test** **Plan** | **Test** **Strategy** |
| Test plan defines the scope, objective and method of software testing. | Test strategy defines how testing will be done. |
| The test plan has components like test plan id, test environment, features to be tested/not tested, entry/exit criteria, status, types of testing, a brief intro. | The test strategy has components like scope and objective, business issues, testing approach, test deliverables, defect tracking approach, training, automation, risks. |
| The test plan is carried out by the test team manager or lead. | Test strategy is carried out by the project manager. |
| Test plan changes frequently. | Test strategies do not change or change less frequently. |
| The test plan is based on test strategies and defined at the project level. | Test strategy is based on based standards and defined at the organizational level. |
| The test plan narrates how the defined approach will be followed. | The test strategy defines the approach. |

Agile Test Strategy Document:

1. Test Approach
   1. Testing Strategy
   2. Roles & Responsibilities
   3. Test Types
2. Testing Environments
3. Testing Automation & Tools
4. Risk Analysis
5. Test Planning & Execution
6. Review & Approval

Code Quality:

1. Complexity
   1. What’s your method is doing?
      1. Cognitive Complexity: Count the Statements that can make code hard to understand
      2. Cyclomatic Complexity: Count the Number of Distinct Code Paths
2. Hotspots & Churn:
   1. Churn 🡪 A count of number of times file has changed. Can be measured at different periods of time.
   2. Hotspots🡪The files that have higher than average complexity and have changed recently.
3. Code Coverage
4. Duplicate
5. Dependency Security
6. Coding Style

|  |  |  |  |
| --- | --- | --- | --- |
| 200 OK | 300 Multiple Choice | 400 Bad Request | 500 Internal Server Error |
| 201 Created | 301 Moved Permanently | 401 Unauthorized | 501 Not implemented |
| 202 Accepted | 302 Temporarily | 402 Payment Required | 502 Bad Gateway |
| 203 Non-Authoritative Information | 303 Sent Directly | 403 Forbidden | 503 Service unavailable |
| 204 No Content | 304 Not Modified | 404 Not Found | 504 Gateway Time Out |
|  | 307 Temporary Redirect | 405 Method Not Allowed |  |
|  |  | 406 Not Acceptable |  |
|  |  | 412 Pre-Condition Failed |  |
|  |  | 415 Unsupported Media Type |  |

Git:

git init

git status

git log

git add .

git commit -m “Commit Message”

git push

git log -n 5

clear

git log --author="kumar"

git log --grep='init'

ls -la

cat .git/HEAD

$ cat .git/HEAD

ref: refs/heads/master

cat

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ cat .git/refs/heads/master

77b4d3c48f68288255f6efd789322a3cc713ec51

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git log

commit 77b4d3c48f68288255f6efd789322a3cc713ec51 (HEAD -> master)

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Fri May 21 16:15:06 2021 +0530

initial Commit

$ git add "New Text Document.txt"

$ git diff

**diff --git a/New Text Document.txt b/New Text Document.txt**

**index 1a7c0e2..1ad9b69 100644**

**--- a/New Text Document.txt**

**+++ b/New Text Document.txt**

@@ -1 +1 @@

-acscxc

\ No newline at end of file

+acscxc aghdvasbdasndb

\ No newline at end of file

e10

$ git diff HEAD

**diff --git a/New Text Document.txt b/New Text Document.txt**

**index 1a7c0e2..1ad9b69 100644**

**--- a/New Text Document.txt**

**+++ b/New Text Document.txt**

@@ -1 +1 @@

-acscxc

\ No newline at end of file

+acscxc aghdvasbdasndb

\ No newline at end of file

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git diff

**diff --git a/asadsad.txt b/asadsad.txt**

**deleted file mode 100644**

**index 1ad9b69..0000000**

**--- a/asadsad.txt**

**+++ /dev/null**

@@ -1 +0,0 @@

-acscxc aghdvasbdasndb

\ No newline at end of file

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git diff --staged

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

deleted: File1.txt

no changes added to commit (use "git add" and/or "git commit -a")

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git rm File1.txt

rm 'File1.txt'

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

deleted: File1.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git rm File.txt

fatal: pathspec 'File.txt' did not match any files

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git rm File2.txt

rm 'File2.txt'

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

deleted: File2.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git commit -m "sample DeleredF"

[master a2d5cea] sample DeleredF

1 file changed, 1 deletion(-)

delete mode 100644 File2.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

nothing to commit, working tree clean

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

deleted: File1.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

File.txt

no changes added to commit (use "git add" and/or "git commit -a")

git

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git add File1.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git add File.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git rm File1.txt

fatal: pathspec 'File1.txt' did not match any files

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

renamed: File1.txt -> File.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

deleted: File1.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

File.txt

no changes added to commit (use "git add" and/or "git commit -a")

git

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git add File1.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git add File.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git rm File1.txt

fatal: pathspec 'File1.txt' did not match any files

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

renamed: File1.txt -> File.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

no changes added to commit (use "git add" and/or "git commit -a")

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git restore Sample.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

nothing to commit, working tree clean

git st

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: Sample.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

Sample - Copy.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git reset HEAD

Unstaged changes after reset:

M Sample.txt

git

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

Untracked files:

(use "git add <file>..." to include in what will be committed)

Sample - Copy.txt

no changes added to commit (use "git add" and/or "git commit -a")

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git log

commit 94593399aa68269f4ff0fd276c9c21e7c91c2442 (HEAD -> master)

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 10:22:15 2021 +0530

Sample

commit 9ff777194a2d89d6d8709b11431dd041277ae681

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 10:10:58 2021 +0530

Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$ git show 9ff77

commit 9ff777194a2d89d6d8709b11431dd041277ae681

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 10:10:58 2021 +0530

Sample

**diff --git a/Sample.txt b/Sample.txt**

**new file mode 100644**

**index 0000000..7c485fa**

**--- /dev/null**

**+++ b/Sample.txt**

@@ -0,0 +1 @@

+Sample Text

\ No newline at end of file

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample (master)

$

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master|REVERTING)

$ git status

On branch master

You are currently reverting commit cdde683.

(all conflicts fixed: run "git revert --continue")

(use "git revert --skip" to skip this patch)

(use "git revert --abort" to cancel the revert operation)

Untracked files:

(use "git add <file>..." to include in what will be committed)

Sample - Copy.txt

nothing added to commit but untracked files present (use "git add" to track)

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master|REVERTING)

$ git clean -i

**Would remove the following item:**

Sample - Copy.txt

**\*\*\* Commands \*\*\***

1: clean 2: filter by pattern 3: select by numbers

4: ask each 5: quit 6: help

What now> Y

Huh (Y)?

**\*\*\* Commands \*\*\***

1: clean 2: filter by pattern 3: select by numbers

4: ask each 5: quit 6: help

What now> 1

Removing Sample - Copy.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master|REVERTING)

$ git status

On branch master

You are currently reverting commit cdde683.

(all conflicts fixed: run "git revert --continue")

(use "git revert --skip" to skip this patch)

(use "git revert --abort" to cancel the revert operation)

nothing to commit, working tree clean

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master|REVERTING)

$

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git status

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: Sample.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git reset HEAD Sample.txt

Unstaged changes after reset:

M Sample.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

no changes added to commit (use "git add" and/or "git commit -a")

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git clean -n

Would remove Sample - Copy - Copy.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git clean -f

Removing Sample - Copy - Copy.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git status

On

* git reset changes the staging index and git checkout changes the working directory.

Feedback

git reset HEAD unstages files, or removes the changes from the staging index, while git checkout pulls the last commit's version of a file from the repository and replaces the working directory version with it.

Why might Git track a file in .gitignore?

**You are correct!**

* The file was added to .gitignore after Git began tracking it.

Feedback

A common error when .gitignore does not seem to properly work for a file is when the file has already been tracked by Git. Removing it with git rm --cached <filename> will remove the tracked copy.

What is the conventional file name to ensure an otherwise empty directory is included in the repository?

**You are correct!**

* .gitkeep

Feedback

Using .gitkeep for the filename is the convention, though any file name is acceptable.

Each commit represented by one value that is Hash Value: 40 long character string.

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash -m "Awesome Feature"

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

no changes added to commit (use "git add" and/or "git commit -a")

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash

Saved working directory and index state WIP on master: 4c77fa1 Kumar

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git log

commit 4c77fa13d4808cdf639b3b81364b6d96f8d45cf8 (HEAD -> master)

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 12:32:36 2021 +0530

Kumar

commit 3cf61b0b5029b1289e971c21310aedd52005fa3c

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 12:18:09 2021 +0530

Sample

commit 7fad4490debfb1c77b58d181a3a2da15c5ad758f

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 10:47:38 2021 +0530

Sanoke giles

commit a6bbc9424b6951575accc89d405ac093c394e22a

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 10:38:46 2021 +0530

Sample Tet

commit cdde683f23f5799ea042a3dddc2e409273558fad

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 10:38:08 2021 +0530

Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash apply

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

no changes added to commit (use "git add" and/or "git commit -a")

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash

Saved working directory and index state WIP on master: 4c77fa1 Kumar

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash

No local changes to save

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: WIP on master: 4c77fa1 Kumar

stash@{1}: WIP on master: 4c77fa1 Kumar

stash@{2}: WIP on master: 3cf61b0 Sample

stash@{3}: WIP on master: 3cf61b0 Sample

stash@{4}: WIP on master: 3cf61b0 Sample

stash@{5}: WIP on master: 3cf61b0 Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash apply o

error: o is not a valid reference

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash apply 0

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

no changes added to commit (use "git add" and/or "git commit -a")

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash apply 1

error: Your local changes to the following files would be overwritten by merge:

Sample.txt

Please commit your changes or stash them before you merge.

Aborting

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: WIP on master: 4c77fa1 Kumar

stash@{1}: WIP on master: 4c77fa1 Kumar

stash@{2}: WIP on master: 3cf61b0 Sample

stash@{3}: WIP on master: 3cf61b0 Sample

stash@{4}: WIP on master: 3cf61b0 Sample

stash@{5}: WIP on master: 3cf61b0 Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash apply 5

On branch master

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

new file: access - Copy.log

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash

Saved working directory and index state WIP on master: 4c77fa1 Kumar

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git status

On branch master

nothing to commit, working tree clean

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git list

git: 'list' is not a git command. See 'git --help'.

The most similar commands are

bisect

rev-list

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: WIP on master: 4c77fa1 Kumar

stash@{1}: WIP on master: 4c77fa1 Kumar

stash@{2}: WIP on master: 4c77fa1 Kumar

stash@{3}: WIP on master: 3cf61b0 Sample

stash@{4}: WIP on master: 3cf61b0 Sample

stash@{5}: WIP on master: 3cf61b0 Sample

stash@{6}: WIP on master: 3cf61b0 Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash apply 2

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

no changes added to commit (use "git add" and/or "git commit -a")

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash

Saved working directory and index state WIP on master: 4c77fa1 Kumar

git

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash lisr

fatal: unknown subcommand: lisr

usage: git stash list [<options>]

or: git stash show [<options>] [<stash>]

or: git stash drop [-q|--quiet] [<stash>]

or: git stash ( pop | apply ) [--index] [-q|--quiet] [<stash>]

or: git stash branch <branchname> [<stash>]

or: git stash clear

or: git stash [push [-p|--patch] [-k|--[no-]keep-index] [-q|--quiet]

[-u|--include-untracked] [-a|--all] [-m|--message <message>]

[--pathspec-from-file=<file> [--pathspec-file-nul]]

[--] [<pathspec>...]]

or: git stash save [-p|--patch] [-k|--[no-]keep-index] [-q|--quiet]

[-u|--include-untracked] [-a|--all] [<message>]

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: WIP on master: 4c77fa1 Kumar

stash@{1}: WIP on master: 4c77fa1 Kumar

stash@{2}: WIP on master: 4c77fa1 Kumar

stash@{3}: WIP on master: 4c77fa1 Kumar

stash@{4}: WIP on master: 3cf61b0 Sample

stash@{5}: WIP on master: 3cf61b0 Sample

stash@{6}: WIP on master: 3cf61b0 Sample

stash@{7}: WIP on master: 3cf61b0 Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git status

On branch master

nothing to commit, working tree clean

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: WIP on master: 4c77fa1 Kumar

stash@{1}: WIP on master: 4c77fa1 Kumar

stash@{2}: WIP on master: 4c77fa1 Kumar

stash@{3}: WIP on master: 4c77fa1 Kumar

stash@{4}: WIP on master: 3cf61b0 Sample

stash@{5}: WIP on master: 3cf61b0 Sample

stash@{6}: WIP on master: 3cf61b0 Sample

stash@{7}: WIP on master: 3cf61b0 Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash apply 0

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

no changes added to commit (use "git add" and/or "git commit -a")

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash -m "Awesome Feature"

Saved working directory and index state On master: Awesome Feature

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: On master: Awesome Feature

stash@{1}: WIP on master: 4c77fa1 Kumar

stash@{2}: WIP on master: 4c77fa1 Kumar

stash@{3}: WIP on master: 4c77fa1 Kumar

stash@{4}: WIP on master: 4c77fa1 Kumar

stash@{5}: WIP on master: 3cf61b0 Sample

stash@{6}: WIP on master: 3cf61b0 Sample

stash@{7}: WIP on master: 3cf61b0 Sample

stash@{8}: WIP on master: 3cf61b0 Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ ^C

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ ^C

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash apply

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

no changes added to commit (use "git add" and/or "git commit -a")

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash push -m "final feature"

Saved working directory and index state On master: final feature

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: On master: final feature

stash@{1}: On master: Awesome Feature

stash@{2}: WIP on master: 4c77fa1 Kumar

stash@{3}: WIP on master: 4c77fa1 Kumar

stash@{4}: WIP on master: 4c77fa1 Kumar

stash@{5}: WIP on master: 4c77fa1 Kumar

stash@{6}: WIP on master: 3cf61b0 Sample

stash@{7}: WIP on master: 3cf61b0 Sample

stash@{8}: WIP on master: 3cf61b0 Sample

stash@{9}: WIP on master: 3cf61b0 Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash drop 2

Dropped refs/stash@{2} (d36985bfed11c0747bceaec4e4607b0a6c565cc6)

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: On master: final feature

stash@{1}: On master: Awesome Feature

stash@{2}: WIP on master: 4c77fa1 Kumar

stash@{3}: WIP on master: 4c77fa1 Kumar

stash@{4}: WIP on master: 4c77fa1 Kumar

stash@{5}: WIP on master: 3cf61b0 Sample

stash@{6}: WIP on master: 3cf61b0 Sample

stash@{7}: WIP on master: 3cf61b0 Sample

stash@{8}: WIP on master: 3cf61b0 Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash drop 2

Dropped refs/stash@{2} (9310d26bbca86cd724778fe654a35dd77544a5b0)

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash drop 2

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash drop 2

Dropped refs/stash@{2} (3e6a3db6f1509dac84addfb58aa48c0188c5822b)

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash drop 2

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: On master: final feature

stash@{1}: On master: Awesome Feature

stash@{2}: WIP on master: 3cf61b0 Sample

stash@{3}: WIP on master: 3cf61b0 Sample

stash@{4}: WIP on master: 3cf61b0 Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash drop 2

Dropped refs/stash@{2} (8d24307342eb11416ee14bba665bf0d515e5e3bb)

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash drop 2

Dropped refs/stash@{2} (610eb0da9d82e711a83006fec1d80a5e1c03c43b)

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash drop 2

Dropped refs/stash@{2} (850bbbf853e64b4720fc9a5c82114ee883a65e39)

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: On master: final feature

stash@{1}: On master: Awesome Feature

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git status

On branch master

nothing to commit, working tree clean

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash pop 0

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Sample.txt

no changes added to commit (use "git add" and/or "git commit -a")

Dropped refs/stash@{0} (3200cc800d708baf8c6614c9a0b35c2cd30c6537)

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: On master: Awesome Feature

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git add .

git co

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git commit -m "Final Feature"

[master a49f7ee] Final Feature

1 file changed, 7 insertions(+), 1 deletion(-)

git

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git log

commit a49f7eef61f9a4fcdd8cdbb1291ebb9bd5a1c8e0 (HEAD -> master)

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 12:45:48 2021 +0530

Final Feature

commit 4c77fa13d4808cdf639b3b81364b6d96f8d45cf8

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 12:32:36 2021 +0530

Kumar

commit 3cf61b0b5029b1289e971c21310aedd52005fa3c

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 12:18:09 2021 +0530

Sample

commit 7fad4490debfb1c77b58d181a3a2da15c5ad758f

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 10:47:38 2021 +0530

Sanoke giles

commit a6bbc9424b6951575accc89d405ac093c394e22a

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 10:38:46 2021 +0530

Sample Tet

commit cdde683f23f5799ea042a3dddc2e409273558fad

Author: e104399 <Kumar.Gorla2@mastercard.com>

Date: Sat May 22 10:38:08 2021 +0530

Sample

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$

git

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

stash@{0}: On master: Awesome Feature

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash clear

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$ git stash list

e104399@GH-B3QD1N2 MINGW64 /c/GK\_Work/codebase/Sample Project (master)

$



Spring Framework:

* + - 1. Simplicity
      2. Testability
      3. Loose Coupling
* Web Layer
* Common Layer
* Service Layer
* Data Layer

Spring Modules:

1. Core Container
   1. Core
   2. Bean
   3. SpEL
   4. Context
2. Data Access/ Integration
   1. JDBC
   2. ORM
   3. Transaction
   4. JMS
   5. OXM
3. Web
   1. Web
   2. Web MVC
   3. Web Portlet
   4. Web Socket
4. Misc.
   1. AOP
   2. Instrumentation
   3. Test
   4. Aspects
   5. Messaging

* IoC : Inversion of Control : Configured by loading the XML files or by detecting specific java annotation on configuration classes
  + Bean Factory
  + Application Context
* Dependency Injection
  + By Constructor
  + By Setter Method
* AOP: Aspect Oriented Programming: Cross Cutting Concerns
* MVC: User – Presentation Layer (View) 🡪 Controller 🡪 Model 🡪 View 🡪 User

Advantages of Java Framework:

* Efficiency
* Security
* Expense
* Support
* Code is public
* Custom build features
* Restriction
* Bean Class
  + XML File
    - Demo Class
      * Loading Jar Files
* Run

IoC : you don’t use create Objects. Object will be created by Spring by use of 2 API’s

* Application Context: Constructs objects directly
* Bean Factory won’t create until u mention Bean Class.

Bean Object:

* Beans are objects that form the backbone of our application and are managed by the Spring IoC Controller
* Spring IoC container instances, assembles and manages the bean object
* The configuration meta data that are supplied to the container are used to create the bean object

MetaData🡪 IoC Container 🡪 Bean Object

XML Based Configuration File

Annotation Based Configuration

Java bases Configuration

Life Cycle:

Instantiate 🡪 Populate Properties 🡪 Call BeanName Aware’s setBeanName() 🡪 Call BeanFactoryAware’s setBeanFactory() 🡪 Call ApplicationCOntextAware’s setApplicationCOntext() 🡪 Preinitialize(BeanPostProcessor) 🡪 Call Initializing Bean’s after Properies Set() 🡪 Call custom -init() 🡪 post initialization(BeanPOstProcessr)

Container Shutdown -> Cal Disposabledesttroy 🡪 Call custom destroy()

Dependency Injection: Makes code loosely coupled

* Constructor
* Setter
* Interface

AOP:

Advice 🡪 Join Point 🡪 Point Cut

Spring AOP :

1. Maven Dependencies
2. Aspect & Point Cut Expression
3. Methods (Joint Points)
4. Run the Application

Types of Advices:

* Before Advice
* After Advice
* After Returning Advice
* Around Advice

Spring AOP ASPECTJ Annotations:

1. Project Dependencies
2. Spring Beans
3. Enable AspectJ
4. Apply Advices

What is Dependency Injection:

The Process of creating an object for some other class and let the class directly using the dependency is called Dependency Injection.

Scope in Spring;

* Singleton
* Prototype
* Request
* Session
* Global session

Annotations

1. Component : Generic Stereotype
2. Controller : Presentation : Business Part
3. Service : Back end Layer
4. Repository

Java Lambda:

* Enables Functional Programming
* Readable and Concise Code
* Easier to use API and Libraries
* Enable support for parallel processing

Code in OOP:

* Everything is an object
* All code blocks are “associated” with classes and objects .

Lambda Expression:

aBlockOfCode = ***public*** ***void*** *perform*() {

System.***out***.println("Greeting.....!");

}

Next:

aBlockOfCode = **() ->** {

System.***out***.println("Greeting.....!");

}

Next : if only one line:

LambdaExpression :

aBlockOfCode = **() ->** System.***out***.println("Greeting.....!");

1. greetingFunction = () -> System.out.print(“Greeting…”);
2. greet(greetingFunction);
3. greet(()->System.out.print(“Greeting…”));
4. doubleNumberFunction = public int double(int a){

return a \* 2;

}

doubleNumberFunction = (int a) -> a \* 2;

1. addFunction = (int a, int b) -> a + b;
2. safeDivideFunction = (int a, int b) -> {

if(b==0)

return 0;

return a / b;

};

1. stringLengthCountFunction = (String str) -> str.length();

Functional Interface:

Only One Abstract Method + Any no of Default Methods

**Selenium 3 and 4 features:**

Selenium 4 is packed with excellent features like

* Relative locators
* Improved Selenium Grid architecture
* Super-useful Selenium IDE
* W3C compliance of WebDriver APIs
* Enhanced Documentation
* Simplification to open new browser Windows and Tabs

driver.get("https://www.google.com/");

// Opens a new window and switches to new window

driver.switchTo().newWindow(WindowType.WINDOW);

// Opens LambdaTest homepage in the newly opened window

driver.navigate().to("https://www.lambdatest.com/");

driver.get("https://www.google.com/");

// Opens a new window and switches to new window

driver.switchTo().newWindow(WindowType.TAB);

// Opens LambdaTest homepage in the newly opened tab

driver.navigate().to("https://www.lambdatest.com/");

|  |  |
| --- | --- |
| **RELATIVE LOCATOR** | **DESCRIPTION** |
| above | The required WebElement is ‘above’ a specified (or particular) element. |
| below | The required WebElement is ‘below’ a specified (or particular) element. |
| to\_left\_of | The required WebElement is ‘to the left of’ a specified (or particular) element. |
| to\_right\_of | The required WebElement is ‘to the right of’ a specified (or particular) element. |
| near | The required WebElement is ‘at most 50 pixels’ away from the specified (or particular) element. |

**Monitoring**

Processes related to logging and request tracing are quite optimized in Selenium 4. These optimizations help in accelerating the debugging process, which eventually aid in delivering a better quality test script.

**TakeElementScreenshot**

In Selenium 3, there was a provision to capture a screenshot of the entire web page. Selenium 4 onwards, there is a new option to capture screenshots of a particular WebElement. Hence, there is no need to use third-party tools like Shutterbug, Ashot, etc. (like in Selenium 3) for capturing a screenshot of WebElement.

WebElement element = driver.findElement(By.cssSelector(".btn"));

File scrFile = ((TakesScreenshot)element).getScreenshotAs(OutputType.FILE);

File dstFile = new File("./loginbutton.png");

FileUtils.copyFile(scrFile, dstFile);

Thread.sleep(2000);

* Chrome DevTools:
* Deprecation of DesiredCapabilities

In Selenium 3, we make extensive use of the DesiredCapabilities when working with a RemoteWebDriver. This is required for setting the browser capabilities so that tests can be run on a cloud-based Selenium Grid like LambdaTest. With Selenium 4, we bid adieu to DesiredCapabilities.

Capabilities objects are now replaced with Options, and we need to create an Options object to use the Driver class. With Selenium 4, we need to set the necessary test requirements (i.e., browser and OS combinations) and pass the object to the Driver constructor. Henceforth, the following Options object would be used for setting the browser-specific capabilities:

Chrome – ChromeOptions

Firefox – FirefoxOptions

Internet Explorer (IE) – InternetExplorerOptions

Safari – SafariOptions

Microsoft Edge – EdgeOptions

* New additions to the Actions Class
  1. moveToElement().click() 🡪 Click(elemtntToClick)
  2. moveToElement(element).doubleClick() 🡪 doubleClick(element)
  3. moveToElement(element).contextClick() 🡪 ContextClick(element)
  4. moveToElement(Element).clickAndHold()  🡪 clickAndHold(element)

The native support for Opera and PhantomJS is removed in Selenium 4, as their WebDriver implementations are no longer under development. The Opera browser is based on Chromium, and users looking to test their implementation on Opera can opt for testing on the Chrome browser. PhantomJS users can opt for testing on Firefox and Chrome in the headless mode.

**Stale Element Reference Exception:**

What is StaleElementReferenceException in Selenium Webdriver:

Stale means old, decayed, no longer fresh. Stale Element means an old element or no longer available element. Assume there is an element that is found on a web page referenced as a WebElement in WebDriver. If the DOM changes then the WebElement goes stale. If we try to interact with an element which is staled then the StaleElementReferenceException is thrown.

What are the Causes of StaleElement Exception:

We face this stale element reference exception when the element we are interacting with is destroyed and then recreated again. When this happens the reference of the element in the DOM becomes stale. Hence we are not able to get the reference to the element.

Cause 1: The referenced web element has been deleted completely.

Assume that there is an element in a web page and the page that the web element was part of has been refreshed or the selenium script navigated away to another web page. This causes any subsequent interaction with the element to fail with the stale element reference exception.

Cause 2: The referenced element is no longer attached to the DOM

Assume a document node is removed from the DOM, its element reference will be invalidated. This causes any subsequent interaction with the element to fail with the stale element reference exception.

Out of the above two causes, the first case is more common than the second case.

* **Solution 1: Refreshing the web page**

You could refresh the page and try again for the same element.

Assume you are trying to click on a link and getting the stale element exception.

Sample code to overcome the issue

driver.navigate().refersh();

driver.findElement(By.xpath("xpath here")).click();

* **Solution 2: Using Try Catch Block**

If an element is not attached to DOM then you could try using ‘try-catch block’ within ‘for loop’

// Using for loop, it tries for 3 times.

// If the element is located for the first time then it breaks from the for loop and comeout of the loop

for(int i=0; i<=2;i++){

try{

driver.findElement(By.xpath("xpath here")).click();

break;

}

catch(Exception e){

Sysout(e.getMessage());

}

* **Solution 3: Using ExpectedConditions.refreshed**

Wait for the element till it gets available

wait.until(ExpectedConditions.presenceOfElementLocated(By.id("table")));

Use ExpectedConditions.refreshed to avoid StaleElementReferenceException and retrieve the element again. This method updates the element by redrawing it and we can access the referenced element.

wait.until(ExpectedConditions.refreshed(ExpectedConditions.stalenessOf("table")));

* **Solution 4: Using POM**

We can handle Stale Element Reference Exception by using POM.

Must Read: Page Object Model Design Pattern

We could avoid StaleElementException using POM. In POM, we use initElements() method which loads the element but it won’t initialize elements. initElements() takes latest address. It initializes during run time when we try to perform any action on an element. This process is also known as Lazy Initialization.

In Conclusion:

The StaleElementReferenceException is a very common exception that we face in our Selenium tests.

**Diff between Data Provider and Factory Annotations in TestNG:**

<https://howtodoinjava.com/testng/difference-factory-dataprovider/>

@DataProvider – A test method that uses @DataProvider will be executed multiple number of times based on the configuration provided in it. The test method will be executed using the same instance of the test class to which the test method belongs.

@Factory – A factory will execute all the test methods present inside a test class using separate instances of the class.

TestNG @Factory is used to create instances of test classes dynamically. This is useful if you want to run the test class any number of times.

For example, if you have a test to login to a site and you want to run this test multiple times, then it is easy to use the TestNG factory where you create multiple instances of test class and run the tests (maybe to test any memory leak issues).

Whereas, @DataProvider is used to provide parameters to a test. If you provide the data provider to a test, the test will be run taking a different set of values each time. This is useful for a scenario like where you want to login to a site with a different set of username and password each time.

|  |
| --- |
| public class DataProviderClass  {      @BeforeClass      public void beforeClass() {          System.out.println("Before class executed");      }        @Test(dataProvider = "dataMethod")      public void testMethod(String param) {          System.out.println("The parameter value is: " + param);      }        @DataProvider      public Object[][] dataMethod() {          return new Object[][] { { "one" }, { "two" } };      }  } |

Let’s run the above test.

|  |
| --- |
| Before class executed  The parameter value is: one  The parameter value is: two  PASSED: testMethod("one")  PASSED: testMethod("two") |

|  |
| --- |
| public class SimpleTest  {      private String param = "";        public SimpleTest(String param) {          this.param = param;      }        @BeforeClass      public void beforeClass() {          System.out.println("Before SimpleTest class executed.");      }        @Test      public void testMethod() {          System.out.println("testMethod parameter value is: " + param);      }  }    public class SimpleTestFactory  {      @Factory      public Object[] factoryMethod() {          return new Object[] {                                  new SimpleTest("one"),                                  new SimpleTest("two")                              };      }  } |

Let’s run the above test.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Before SimpleTest class executed.  testMethod parameter value is: two  Before SimpleTest class executed.  testMethod parameter value is: one  PASSED: testMethod  PASSED: testMethod  **Page Object Model & Page Factory:**  **POM:-**  Page Object Model, also known as POM, is a design pattern in Selenium that creates an object repository for  storing all web elements. It is useful in reducing code duplication and improves test case maintenance.  In Page Object Model, consider each web page of an application as a class file. Each class file will contain  only corresponding web page elements. Using these elements, testers can perform operations on the  website under test.  **Page Factory:**  Page Factory is a class provided by Selenium WebDriver to support Page Object Design patterns.  In Page Factory, testers use @FindBy annotation. The initElements method is used to initialize web elements.  @FindBy: An annotation used in Page Factory to locate and declare web elements using different locators.  Below is an example of declaring an element using  @FindBy  @FindBy(id="elementId") WebElement element;  Similarly, one can use @FindBy with different location strategies to find web elements and  perform actions on them. Below are locators that can be used:  ClassName  CSS  Name  Xpath  TagName  LinkText  PartialLinkText  initElements(): initElements is a static method in Page Factory class. Using the initElements method,  one can initialize all the web elements located by @FindBy annotation.  lazy initialization: AjaxElementLocatorFactory is a lazy load concept in Page Factory.  This is used to identify web elements only when they are used in any operation or activity.  The timeout of a web element can be assigned to the object class with the help of the AjaxElementLocatorFactory.   |  |  | | --- | --- | | Page Object Model | Page Factory | | Finding web elements using By | Finding web elements using @FindBy | | POM does not provide lazy initialization | Page Factory does provide lazy initialization | | Page Object Model is a design pattern | PageFactory is a class that provides the implementation of the Page Object design pattern | | In POM, one needs to initialize every page object individually | In PageFactory, all page objects are initialized by using the initElements() method | |

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |

Rebase and Merge:

rebase “reapplies commits on top of another base branch”,

whereas merge “joins two or more development histories together”.

In other words, the key difference between merge and rebase is that while merge preserves history as it happened, rebase rewrites it.

**Branching Strategy in GIT:**

1. Git Flow
2. GitHub Flow
3. GitLab Flow
4. Trunk Based

**Important points to remember about throws keyword:**

* throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
* throws keyword is required only to convince compiler and usage of throws keyword does not prevent abnormal termination of program.
* By the help of throws keyword we can provide information to the caller of the method about the exception.

**Throwable Class in Java with Examples:**

The Throwable class is the superclass of every error and exception in the Java language. Only objects that are one of the subclasses this class are thrown by any “Java Virtual Machine” or may be thrown by the Java throw statement. For the motives of checking of exceptions during compile-time, Throwable and any subclass of Throwable which is not also a subclass of either Error or RuntimeException are considered as checked exceptions.

Throwable class is the root class of Java Exception Heirarchy and is inherited by two subclasses:

1.Exception

2.Error

The throwable class implements Serializable Interface and the direct known classes to Throwable are Error and Exception.

Throwable contains a snapshot of the execution stack of its thread at the time it was created. It can also contain a message string that gives more information about the error. It can also suppress other throwables from being propagated.

If a user wants to create his own , custom throwable, then he/she can extend Throwable class.

Example:

Class MyThrowable extends Throwable{

//Here the user can create his own custom throwable

}

public class Throwable

extends Object

implements Serializable

Constructors: Any class can have any one of the three or all the three types of constructors. They are default, parameterized and non-parameterized constructors. This class primarily has the following constructors defined:

Public Constructors

* Throwable(): It is a non-parameterized constructor which constructs a new Throwable with null as its detailed message.
* Throwable(String message): It is a parameterized constructor which constructs a new Throwable with the specific detailed message.
* Throwable(String message, Throwable cause): It is a parameterized constructor which constructs a new Throwable with the specific detailed message and a cause.
* Throwable(Throwable cause): It is a parameterized constructor which constructs a new Throwable with the specific cause and a detailed message of the cause by converting the case to the String using toString() method.

Methods:

* addSuppresed(Thorwable Exception)
* fillInStackTrace()
* getCause()
* getLocalizedMessage()
* getMessage()
* getStackTrace()
* getSuppressed()
* initCause()
* printStackTrace()
* printStackTrace(PrintWriter s)
* setStackTrace(StackTraceElement[] stakcTrace)
* toString()

String concat and + Operator:

* The concat() method concatenates only immutable String objects and it must be called on immutable String object and its parameter must be immutable String object.
* But the + operator can concatenate the any non-string object like Primitive, object. It converts the non-string object to String and then append it.

class ExampleOfString

{

public static **void** main(String args[])

{

String str1 = "Java";

String str2 = "Goal";

System.out.println(str1.concat(str2));

String str3 = "Hi";

System.out.println(str3 + 123);

}

}

***Output:****JavaGoal  
Hi123*

* As we said the concat() method concatenates only the string objects, if any object has a null reference it will throw NullPointerException.
* But the + operator concatenates the null reference as a “null” string.

class ExampleOfString

{

public static **void** main(String args[])

{

String str1 = "Java";

String str2 = **null**;

System.out.println("By + operator: "+(str1+str2));

System.out.println("By concat(): "+str1.concat(str2));

}

}

***Output:****By + operator: Javanull  
Exception in thread “main” java.lang.NullPointerException at java.base/java.lang.String.concat(String.java:1937) at ExampleOfString.main(ExampleOfString.java:9)*

Time taken by different String :

Time taken to concatenate by + operator : 132484

Time taken to concatenate by concat method: 78762

Time taken to concatenate by StringBuffer: 1167

Time taken to concatenate by StringBuilder: 570

public **class** ExampleJoinMethod

{

public static **void** main(String args[])

{

String[] strings = {"Java", "Goal", "website", "for", "Java", "Learning"};

String joinedString = String.join(" ", strings);

System.out.println(joinedString);

}

}

***Output:****Java Goal website for Java Learning*

**The Synchronized Keyword:**

The synchronized keyword can be used on different levels:

* Instance Methods
* Static Methods
* Code Blocks

When we use a synchronized block, internally Java uses a monitor also known as monitor lock or intrinsic lock, to provide synchronization. These monitors are bound to an object, thus all synchronized blocks of the same object can have only one thread executing them at the same time.

**public** **synchronized** **void** **synchronisedCalculate**() { setSum(getSum() + 1); }

**public** **static** **synchronized** **void** **syncStaticCalculate**() { staticSum = staticSum + 1; }

**public** **void** **performSynchronisedTask**() { **synchronized** (**this**) { setCount(getCount()+1); } }

**Continue and Break:**

|  |  |
| --- | --- |
| **Break** | **Continue** |
| The break statement is used to terminate the loop immediately. | The continue statement is used to skip the current iteration of the loop. |
| break keyword is used to indicate break statements in java programming. | continue keyword is used to indicate continue statement in java programming. |
| We can use a break with the switch statement. | We cannot use a continue with the switch statement. |
| The break statement terminates the whole loop early. | The continue statement brings the next iteration early. |
| It stops the execution of the loop. | It does not stop the execution of the loop. |