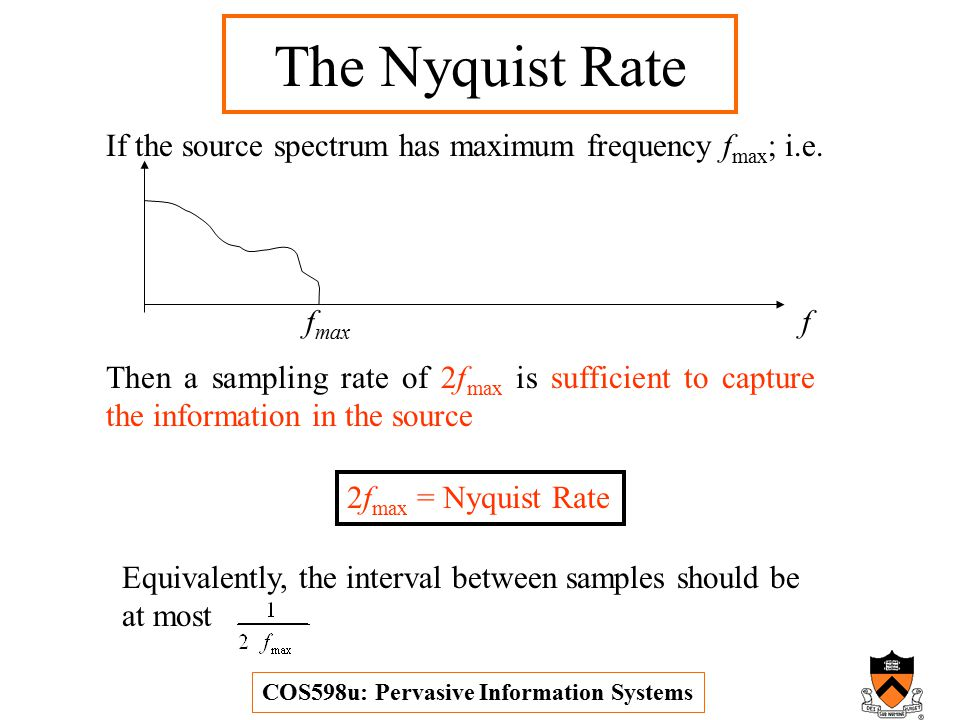
**Testing**

**Q2 .What is Nyquist rate?**

The minimum rate at which a signal can be sampled without introducing errors, which is twice the highest frequency present in the signal.



**Q3. What is sanity Testing,Software Build,somke Testing ,Regression testing,performance testing?**

## **Software Build :**

If you are developing a simple computer program which consists of only one source code file, you merely need to compile and link this one file, to produce an executable file. This process is very simple.

Usually this is not the case. A typical Software Project consists of hundreds or even thousands of source code files. Creating an executable program from these source files is a complicated and time-consuming task.   
You need to use "build" software to create an executable program and the process is called " Software Build"

**Regression testing :** is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.

Regression testing is nothing but full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

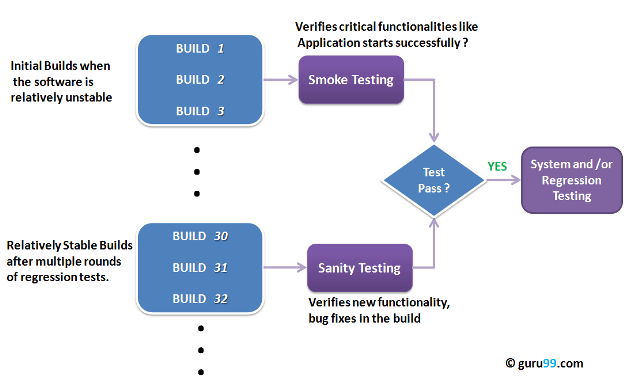
This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that old code still works once the new code changes are done.

**Sanity testing :** is a kind of Software Testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes. The goal is to determine that the proposed functionality works roughly as expected. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.

The objective is "not" to verify thoroughly the new functionality, but to determine that the developer has applied some rationality (sanity) while producing the software. For instance, if your scientific calculator gives the result of 2 + 2 =5! Then, there is no point testing the advanced functionalities like sin 30 + cos 50.

**Smoke Testing** : is a kind of Software Testing performed after software build to ascertain that the critical functionalities of the program is working fine. It is executed "before" any detailed functional or regression tests are executed on the software build. The purpose is to reject a badly broken application, so that the QA team does not waste time installing and testing the software application.

In Smoke Testing, the test cases chosen cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system is working fine.  
For Example a typical smoke test would be - Verify that the application launches successfully, Check that the GUI is responsive ... etc.



**Difference between Smoke and Sanity Testing**

|  |  |
| --- | --- |
| **Smoke Testing** | **Sanity Testing** |
| Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine | Sanity Testing is done to check the new functionality / bugs have been fixed |
| The objective of this testing is to verify the "stability" of the system in order to proceed with more rigorous testing | The objective of the testing is to verify the "rationality" of the system in order to proceed with more rigorous testing |
| This testing is performed by the developers or testers | Sanity testing is usually performed by testers |
| Smoke testing is usually documented or scripted | Sanity testing is usually not documented and is unscripted |
| Smoke testing is a subset of Regression testing | Sanity testing is a subset of Acceptance testing |
| Smoke testing exercises the entire system from end to end | Sanity testing exercises only the particular component of the entire system |
| Smoke testing is like General Health Check Up | Sanity Testing is like specialized health check up |

**Performance testing** What is fundamental test process in software testing?

: is a type of testing to ensure software applications will perform well under their expected workload.

Features and Functionality supported by a software system is not the only concern. A software application's performance like its response time, reliability, resource usage and scalability do matter. The goal of performance testing is not to find bugs but to eliminate performance bottlenecks.

The focus of Performance testing is checking a software program's

* Speed - Determines whether the application responds quickly
* Scalability - Determines maximum user load the software application can handle.
* Stability - Determines if the application is stable under varying loads

Performance testing is popularly called as “Perf Testing” and is a subset of performance engineering.

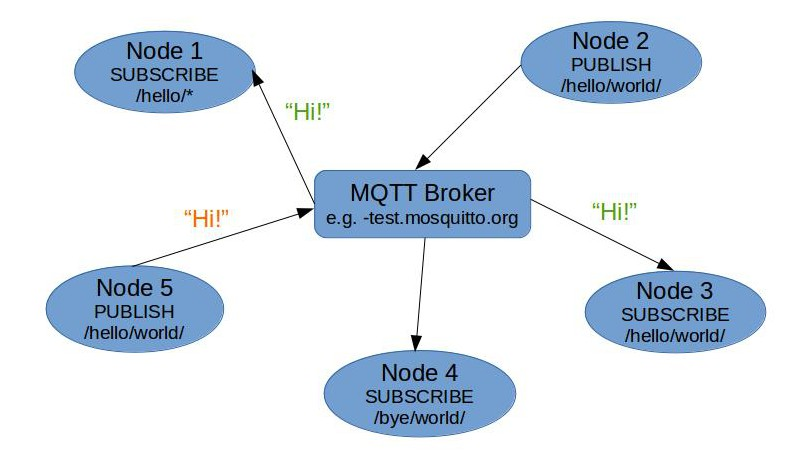
**Q4. Explain Network Protocol(MQTT) and MQTT Architecture. How to reduce Duplication of packet on MQTT broker?**

**Full form :** MQ Telemetry Transport or Message Queue Telemetry Transport

**Definitaion :** MQTT is a lightweight protocol that’s ideal for the Internet of Things.

MQTT is based on the publish-subscribe model. This is quite different from the typical network protocol models where a connection is established, after which the client requests the server for information directly, and the server sends back information to the client’s IP address directly. In case of MQTT, nodes connect to a broker and subscribe to a topic. Then they wait. If a node publishes to the same topic in the node, all the other nodes who had subscribed to it receive it.

In the diagram below, Node 5 publishes “Hi!” to the topic “hello/world/”, and Node 1 and Node 3 receive the message since they are subscribing to the topic.



### Client

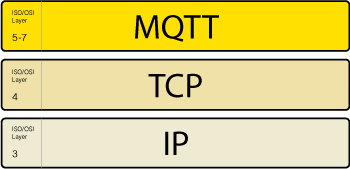
When talking about a client it almost always means an MQTT client. This includes publisher or subscribers, both of them label an MQTT client that is only doing publishing or subscribing. (In general a MQTT client can be both a publisher & subscriber at the same time). A MQTT client is any device from a micro controller up to a full fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. This could be a really small and resource constrained device, that is connected over a wireless network and has a library strapped to the minimum or a typical computer running a graphical MQTT client for testing purposes, basically any device that has a TCP/IP stack and speaks MQTT over it. The client implementation of the MQTT protocol is very straight-forward and really reduced to the essence. That’s one aspect, why MQTT is ideally suitable for small devices. MQTT client libraries are available for a huge variety of programming languages, for example Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, .NET.

### Broker

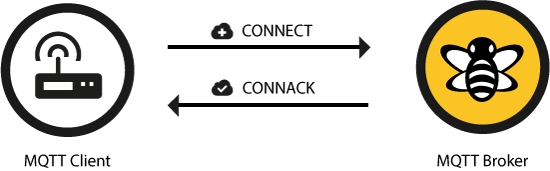
The counterpart to a MQTT client is the MQTT broker, which is the heart of any publish/subscribe protocol. Depending on the concrete implementation, a broker can handle up to thousands of concurrently connected MQTT clients. The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients.

## MQTT Connection

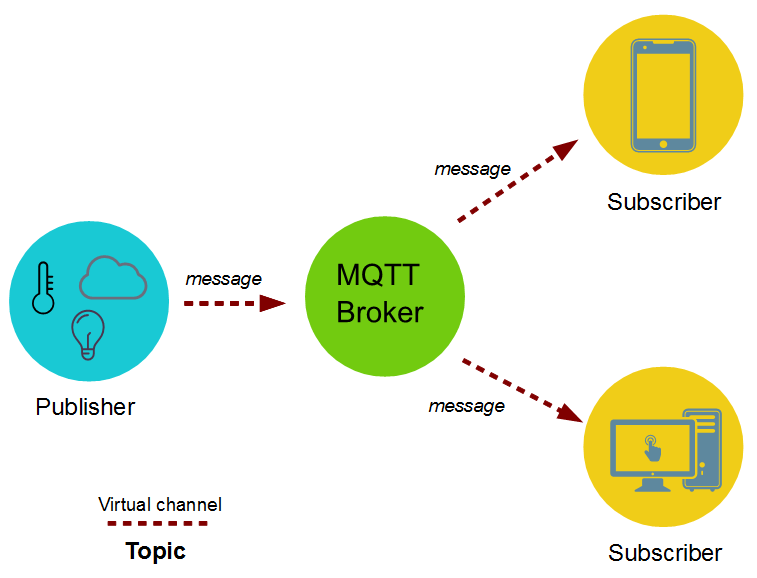
The MQTT protocol is based on top of TCP/IP and both client and broker need to have a TCP/IP stack.



The MQTT connection itself is always between one client and the broker, no client is connected to another client directly. The connection is initiated through a client sending a CONNECT message to the broker. The broker response with a CONNACK and a status code. Once the connection is established, the broker will keep it open as long as the client doesn’t send a disconnect command or it looses the connection.

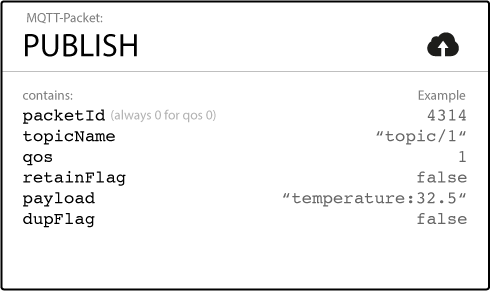


The schema below describes the MQTT architecture:



## **Publish**

After a MQTT client is connected to a broker, it can publish messages. MQTT has a topic-based filtering of the messages on the broker (check [part 2](http://www.hivemq.com/mqtt-essentials-part2-publish-subscribe/) for more details on that), so each message must contain a topic, which will be used by the broker to forward the message to interested clients. Each message typically has a payload which contains the actual data to transmit in byte format. MQTT is data-agnostic and it totally depends on the use case how the payload is structured. It’s completely up to the sender if it wants to send binary data, textual data or even full-fledged XML or JSON. A MQTT publish message also has some more attributes, which we’re going discuss in detail:



Topic Name  
A simple string, which is hierarchically structured with forward slashes as delimiters. An example would be “myhome/livingroom/temperature” or “Germany/Munich/Octoberfest/people”. More details about topics can be found in [part 5 of the MQTT Essentials](http://www.hivemq.com/mqtt-essentials-part-5-mqtt-topics-best-practices/).

Q6.**What Is QOS (quality of service) ?**

QoS  
A Quality of Service Level (QoS) for this message. The level (0,1 or 2) determines the guarantee of a message reaching the other end (client or broker). More details about QoS can be found in [part 6 of the MQTT Essentials](http://www.hivemq.com/mqtt-essentials-part-6-mqtt-quality-of-service-levels/).

Retain-Flag  
This flag determines if the message will be saved by the broker for the specified topic as last known good value. New clients that subscribe to that topic will receive the last retained message on that topic instantly after subscribing. More on retained messages and best practices in one of the next posts.

Payload  
This is the actual content of the message. MQTT is totally data-agnostic, it’s possible to send images, texts in any encoding, encrypted data and virtually every data in binary.

Packet Identifier  
The packet identifier is a unique identifier between client and broker to identify a message in a message flow. This is only relevant for QoS greater than zero. Setting this MQTT internal identifier is the responsibility of the client library and/or the broker.

DUP flag  
The duplicate flag indicates, that this message is a duplicate and is resent because the other end didn’t acknowledge the original message. This is only relevant for QoS greater than 0 and more details are in [part 6, which is about QoS levels](http://www.hivemq.com/mqtt-essentials-part-6-mqtt-quality-of-service-levels/). This resend/duplicate mechanism is typically handled by the MQTT client library or the broker as an implementation detail.

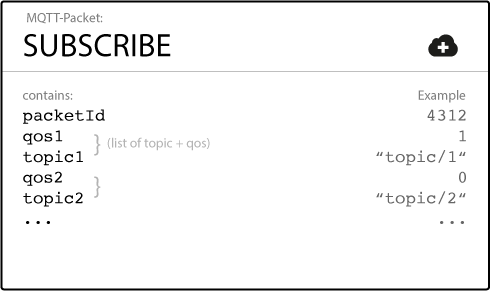
So when a client sends a publish to a MQTT broker, the broker will read the publish, acknowledge the publish if needed (according to the QoS Level) and then process it. Processing includes determining which clients have subscribed on that topic and then sending out the message to the selected clients which subscribe to that topic.



The client, who initially published the message is only concerned about delivering the publish message to the broker. From there on it is the responsibility of the broker to deliver the message to all subscribers. The publishing client doesn’t get any feedback, if someone was interested in this published message and how many clients received the message by the broker.

## **Subscribe**

Publishing messages doesn’t make sense if no one ever receives the message, or, in other words, if there are no clients subscribing to any topic. A client needs to send a [SUBSCRIBE](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html" \l "_Toc398718063) message to the MQTT broker in order to receive relevant messages. A subscribe message is pretty simple, it just contains a unique packet identifier and a list of subscriptions.

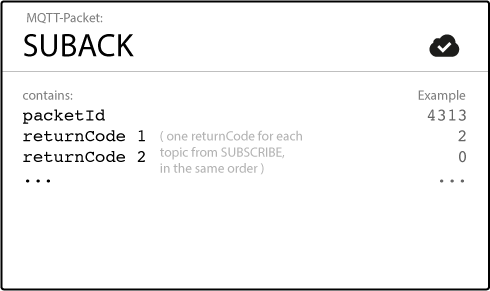


Packet Identifier  
The packet identifier is a unique identifier between client and broker to identify a message in a message flow. Setting this MQTT internal identifier is the responsibility of the client library and/or the broker.

List of Subscriptions  
A SUBSCRIBE message can contain an arbitrary number of subscriptions for a client. Each subscription is a pair of a topic topic and QoS level. The topic in the subscribe message can also contain wildcards, which makes it possible to subscribe to certain topic patterns. If there are overlapping subscriptions for one client, the highest QoS level for that topic wins and will be used by the broker for delivering the message.

## Suback

Each subscription will be confirmed by the broker through sending an acknowledgment to the client in form of the [SUBACK](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html" \l "_Toc398718068) message. This message contains the same packet identifier as the original Subscribe message (in order to identify the message) and a list of return codes.

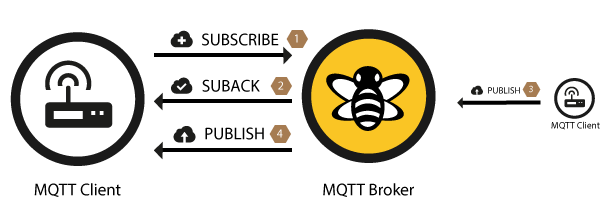


Packet Identifier  
The packet identifier is a unique identifier used to identify a message. It is the same as in the SUBSCRIBE message.

Return Code  
The broker sends one return code for each topic/QoS-pair it received in the SUBSCRIBE message. So if the SUBSCRIBE message What is fundamental test process in software testing?

had 5 subscriptions, there will be 5 return codes to acknowledge each topic with the QoS level granted by the broker. If the subscription was prohibited by the broker (e.g. if the client was not allowed to subscribe to this topic due to insufficient permission or if the topic was malformed), the broker will respond with a failure return code for that specific topic.

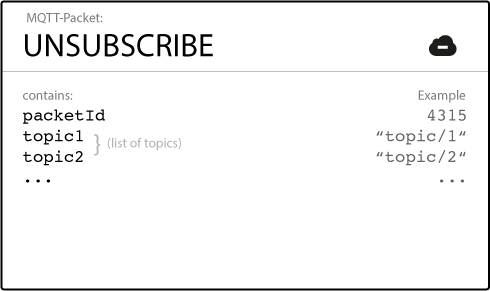
| Return Code | Return Code Response |
| --- | --- |
| 0 | Success – Maximum QoS 0 |
| 1 | Success – Maximum QoS 1 |
| 2 | Success – Maximum QoS 2 |
| 128 | Failure |



After a client successfully sent the SUBSCRIBE message and received the SUBACK message, it will receive every published message matching the topic of the subscription.

## Unsubscribe

The counterpart of the SUBSCRIBE message is the [UNSUBSCRIBE](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html" \l "_Toc398718072) message which deletes existing subscriptions of a client on the broker. The UNSUBSCRIBE message is similar to the SUBSCRIBE message and also has a packet identifier and a list of topics.

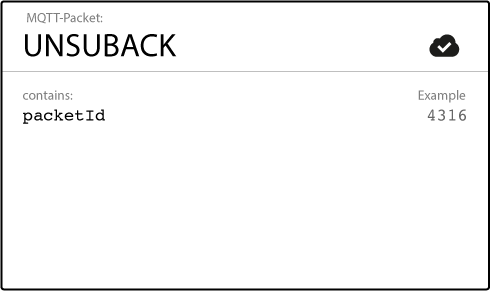


Packet Identifier  
The packet identifier is a unique identifier used to identify a message. The acknowledgement of an UNSUBSCRIBE message will contain the same identifier.

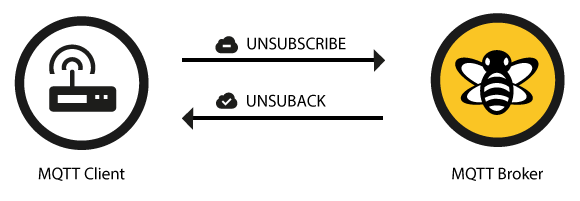
List of Topic  
The list of topics contains an arbitrary number of topics, the client wishes to unsubscribe from. It is only necessary to send the topic as string (without QoS), the topic will be unsubscribed regardless of the QoS level it was initially subscribed with.

## Unsuback

The broker will acknowledge the request to unsubscribe with the [UNSUBACK](http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html" \l "_Toc398718077) message. This message only contains a packet identifier.



Packet Identifier  
The packet identifier is a unique identifier used to identify a message. It is the same as in the UNSUBSCRIBE message.



After receiving the UNSUBACK from the broker, the client can assume the subscriptions in the UNSUBSCRIBE message are deleted.

**Q32.White box Testing ?**

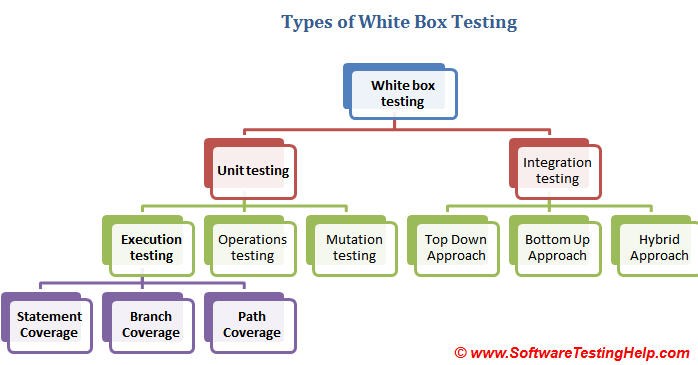
**White-box testing** (also known as **clear box testing**, **glass box testing**, **transparent box testing**, and **structural testing**) : is a method of testing [software](https://en.wikipedia.org/wiki/Software) that tests internal structures or workings of an application, as opposed to its functionality (i.e. [black-box testing](https://en.wikipedia.org/wiki/Black-box_testing)).

In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. [in-circuit testing](https://en.wikipedia.org/wiki/In-circuit_test) (ICT). White-box testing can be applied at the [unit](https://en.wikipedia.org/wiki/Unit_testing), [integration](https://en.wikipedia.org/wiki/Integration_testing) and [system](https://en.wikipedia.org/wiki/System_testing) levels of the [software testing](https://en.wikipedia.org/wiki/Software_testing) process. Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements.

White-box test design techniques include the following [code coverage](https://en.wikipedia.org/wiki/Code_coverage) criteria:

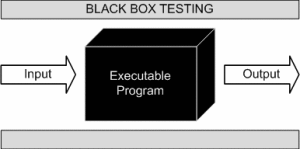
* [Control flow](https://en.wikipedia.org/wiki/Control_flow) testing
* Data flow testing
* Branch testing
* Statement coverage
* Decision coverage
* [Modified condition/decision coverage](https://en.wikipedia.org/wiki/Modified_condition/decision_coverage)
* Prime path testing
* Path testing

**Types of White Box Testing**



**Q33.Black Box Testing ?**

Black Box Testing, also known as Behavioral Testing, is a [software testing method](http://softwaretestingfundamentals.com/software-testing-methods/) in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.



This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see.This method attempts to find errors in the following categories:

* Incorrect or missing functions
* Interface errors
* Errors in data structures or external database access
* Behavior or performance errors
* Initialization and termination errors

Black Box Testing method is applicable to the following levels of software testing:

* [Integration Testing](http://softwaretestingfundamentals.com/integration-testing/)
* [System Testing](http://softwaretestingfundamentals.com/system-testing/)
* [Acceptance Testing](http://softwaretestingfundamentals.com/acceptance-testing/)

The higher the level, and hence the bigger and more complex the box, the more black box testing method comes into use.

**Q34 . Gray Box Testing**

Gray Box Testing is a [software testing method](http://softwaretestingfundamentals.com/software-testing-methods/) which is a combination of [Black Box Testing](http://softwaretestingfundamentals.com/black-box-testing/) method and [White Box Testing](http://softwaretestingfundamentals.com/white-box-testing/) method.

In Black Box Testing, the internal structure of the item being tested is unknown to the tester and in White Box Testing the internal structure in known.

In Gray Box Testing, the internal structure is partially known. This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level.



Gray Box Testing is named so because the software program, in the eyes of the tester is like a gray/ semi-transparent box; inside which one can partially see.

**Q8**.**Describe about the performance testing using QXDM tool.**

**What is QXDM tool:** The **Qualcomm eXtensible Diagnostic Monitor** Professional (QXDM Pro) tool provides a diagnostic client for rapid prototyping of new clients and protocol packets. It utilizes a graphical user interface to display data transmitted to and from the subscriber station.

**Q9.Difference in Application testing and Device testing ?**

Mobile **application testing** is a process by which **application** software developed for handheld mobile devices is tested for its functionality, usability and consistency. Mobile **application testing** can be an automated or manual type of **testing**

**Mobile-device testing** involves a set of activities from monitoring and trouble shooting mobile application, content and services on real handsets. Testing includes **verification** and **validation** of **hardware devices** and **software applications**. Developing applications for the Android platform is a complicated business.

**Q29.What is Re-testing? And Difference between Re – testing and Regression testing.**

**Re-testing** : Re-testing is a type of testing performed to check the test cases that were unsuccessful in the final execution are successfully pass after the defects are repaired .

**Regression testing :** is a type of software testing executed to check whether a code change has not unfavorably disturbed current features & functions of an Application .

### Difference between Retesting and Regression Testing

|  |  |
| --- | --- |
| **Regression Testing** | **Re-testing** |
| * Regression testing is carried out to confirm whether a recent program or code change has not adversely affected existing features | * Re-testing is carried out to confirm the test cases that failed in the final execution are passing after the defects are fixed |
| * The purpose of regression testing is that new code changes should not have any side effects to existing functionalities | * Re-testing is done on the basis of the defect fixes |
| * Defect verification is not the part of regression testing | * Defect verification is the part of re-testing |
| * Based on the project and availability of resources, regression testing can be carried out parallel with Re-testing | * Priority of re-testing is higher than regression testing, so it is carried out before regression testing |
| * You can do automation for regression testing, manual testing could be expensive and time consuming | * You cannot automate the test cases for Retesting |
| * Regression testing is known as a generic testing | * Re-testing is a planned testing |
| * Regression testing is done for passed test cases | * Retesting is done only for failed test cases |
| * Regression testing checks for un-expected side-effects | * Re-testing makes sure that the original fault has been corrected |
| * Regression testing is only done when there is any modification or changes become mandatory in existing project | * Re-testing executes a defect with same data and the same environment with different inputs with new build |
| * Test cases for regression testing can be obtained from the functional specification, user tutorials and manuals, and defect reports in regards to corrected problems | * Test cases for retesting cannot be obtained before start testing. |

**Q28.Real Time Testing ?**

**Real-time testing is the process of** [**testing**](https://en.wikipedia.org/wiki/Software_testing)[**real-time computer systems**](https://en.wikipedia.org/wiki/Real-time_computing)**.**

Software testing is performed to detect and help correct [bugs](https://en.wikipedia.org/wiki/Computer_bug) (errors) in computer software. Testing involves ensuring not only that the software is error-free but that it provides the required functionality to the user. Static and conventional methods of testing can detect bugs, but such techniques may not ensure correct results in real time software systems.

Real-time software systems have strict timing constraints and have a deterministic behavior. These systems have to schedule their tasks such that the timing constraints imposed on them are met. Conventional static way of analysis is not adequate to deal with such timing constraints, hence additional real-time testing is important.

**Q35.What are the Software Development Life Cycle (SDLC) phases?**

Software Development Life Cycle, or SDLC is a process used to develop software.

SDLC creates a structure for the development teams to be able to design, create and deliver high quality software by defining various tasks that need to happen.The life cycle defines a methodology for improving the quality of software and the overall development process.

The intent of a SDLC process it to help produce a product that is cost-efficient, effective, and of high quality.

There are following six phases in every Software development life cycle model:

1. Requirement gathering and analysis
2. Design
3. Implementation or coding
4. Testing
5. Deployment
6. Maintenance



### Requirement Analysis

Software Development Life Cycle begins with Requirement Analysis phase, where the stakeholders discuss the requirements of the software that needs to be developed to achieve a goal. The aim of the requirement analysis phase is to capture the detail of each requirement and to make sure everyone understands the scope of the work and how each requirement is going to be fulfilled.

### Design

The next stage of Software Development Life Cycle is the Design phase. During the design phase, developers and technical architects start the high-level design of the software and system to be able to deliver each requirement.

The technical details of the design is discussed with the stakeholders and various parameters such as risks, technologies to be used, capability of the team, project constraints, time and budget are reviewed and then the best design approach is selected for the product.

The selected architectural design, defines all the components that needs to be developed, communications with third party services, user flows and database communications as well as front-end representations and behaviour of each components. The design is usually kept in the Design Specification Document (DSD).

### Implementation

After the requirements and design activity is completed, the next phase of the Software Development Life Cycle is the implementation or development of the software. In this phase, developers start coding according to the requirements and the design discussed in previous phases.

Database admins create the necessary data in the database, front-end developers create the necessary interfaces and GUI to interact with the back-end all based on guidelines and procedures defined by the company.

Developers also write unit tests for each component to test the new code that they have written, review each other’s code, create builds and deploy software to an environment. This cycle of development is repeated until the requirements are met.

### Testing

Testing is the last phase of the Software Development Life Cycle before the software is delivered to customers. During testing, experienced testers start to test the system against the requirements.

The testers aim to find defects within the system as well as verifying whether the application behaves as expected and according to what was documented in the requirements analysis phase.

Testers can either use a test script to execute each test and verify the results, or use [exploratory testing](http://www.testingexcellence.com/exploratory-testing-tips-best-practices/) which is more of an experience based approach.

It is possible that defects are identified in the testing phase. Once a defect is found, testers inform the developers about the details of the issue and if it is a valid defect, developers will fix and create a new version of the software which needs to be verified again.

This cycle is repeated until all requirements have been tested and all the defects have been fixed and the software is ready to be shipped.

### Deployment and Maintenance

Once the software has been fully tested and no [high priority issues](http://www.testingexcellence.com/severity-and-priority-difference/) .remain in the software, it is time to deploy to production where customers can use the system.

Once a version of the software is released to production, there is usually a maintenance team that look after any post-production issues.

If an issue is encountered in the production the development team is informed and depending on how severe the issue is, it might either require a hot-fix which is created and shipped in a short period of time or if not very severe, it can wait until the next version of the software.

**Q36.What is fundamental test process in software testing?**

Testing is a process rather than a single activity. This process starts from test planning then designing test cases, preparing for execution and evaluating status till the test closure.

Test Planning and Control

Test Plan documents the strategy that will be used to verify and ensure that a product or system meets its design specifications and other requirements.

Test control is about guiding and corrective actions to try to achieve the best possible outcome for the project.

Test Analysis and Design

Test analysis is the process of looking at something that can be used to derive test information.

Test design is the act of creating and writing test suites for testing a software.

Test Implementation and Execution

During execution & implementation, we take the test conditions into test cases and procedures and other testware such as scripts for automation, the test environment and any other test infrastructure.

Evaluating Exit criteria and Reporting

Based on the risk assessment of the project we will set the criteria for each test level against which we will measure the “enough testing”. These criteria vary from project to project and are known as exit criteria.

Exit criteria come into picture, when:

* Maximum test cases are executed with certain pass percentage.
* Bug rate falls below certain level.
* When achieved the deadlines.

