**SDLC**

SDLC is overall process by which software applications and systems are developed.

Software Development (SD) is the set of activities that results in software products.

SDLC establishes a logical order of events for conducting system development that is controlled, measured, documented, and ultimately improved.

**Software Development may include:**

**Research**: involves investigating. Research means searching, or a search of established facts and information to determine evidence or build evidence to support/persuade decision making. So a research is basically when you have an idea or you have a concept and you want to learn more about it. You want to explore all of the details what are associated with that idea.

**New development**: means when you build or develop a brand new software or system for scratch from ground up. The software does not exist, therefore software is newly developed.

**Modification**: when an established system or software is changed or modified or/and enhanced for improve functionality, performance, or capability.

**Reuse**: when you reusing a pre-existing system or pre-built system or software code in order to accomplish the task in hand.

**Reengineering**: when you take an established system and re-engineer it to improve it. Modification changes or adds functionality where as re-engineering does not necessarily to change the existing functionality of the system but enhances the performance of the system.

**Maintenance**: is the support and upkeep of the system.

**There are six core SDLC phases:**

1. Planning (initiations & requirements)

2. Design

3. Development

4. Integration & Testing

5. Implementation (production)

6. Operations, Maintenance & Disposition

**Phase I: Planning—initiation phase**

In the initiation phase, the project begins when a business need or an opportunity is identified. To identify a business need, we need to analyze the benefits, return on investment, realized business efficiencies, and affordability.

A project manager should be appointed at this phase for best practice (but due to company/project dynamics this may not be the case.)

It needs to identify and evaluate all aspects of the business for system concept development. Concept development also involves describing how the business will operate once the approved system is implemented.

It needs to ensure the products/services provide the required capability on-time and within budget, project resources, activities, schedules, tools and reviews are also defined in this stage.

System Boundary Document (SBD) identifies the scope and requires senior official approval for funding.

So the concept is developed to describe how the business will operate once the approved system is implemented and to access how the system will impact employee and customer privacy.

In requirements analysis phase:

-all requirements are defined to a level of detail sufficient for system design to proceed.

-all requirements and to be measurable and testable and relate to the business need or the opportunity identified in the initiation phase.

A GOOD requirement should be clear, complete, detailed, cohesive, testable and agreed to by all players (client, development and testing teams etc.).

Requirements need to be assessed if they are testable according to the SMART criterion.

SMART

Specific: clear and unambiguous.

Measurable: quantifiable, computable

Attainable/Achievable: within reach, reasonable

Realistic: practicable, sensible, rational

Timely: appropriate, well-timed, apt, suitable

Requirements Gathering Process:

In BIG project, Business Analysts are dedicated to this activity. In SMALL projects, QA Tester/QA specialist/QA Manager will have to write the requirements themselves, thereby working in a business analyst capacity.

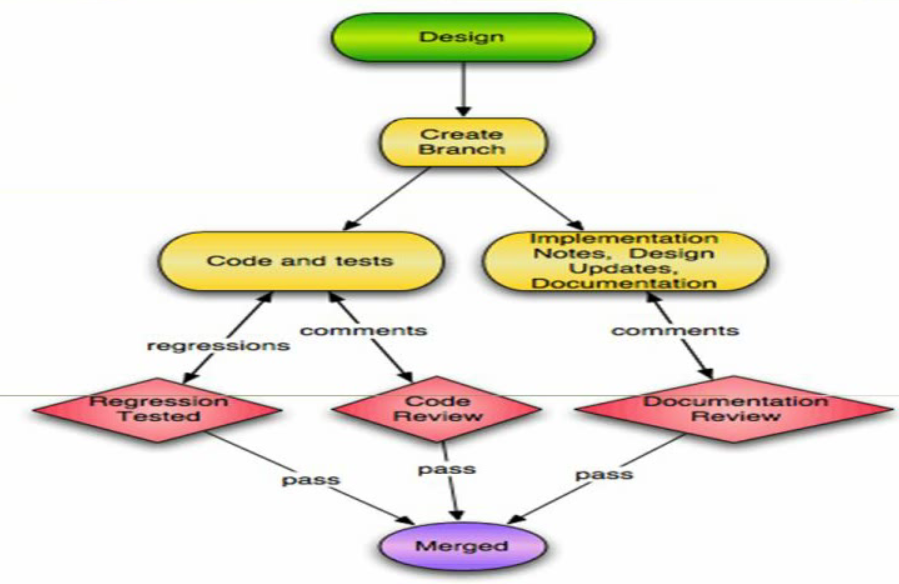
Requirements are gathered by interviewing end users/clients/developers/other team members.

**Phase II: Design Phase**

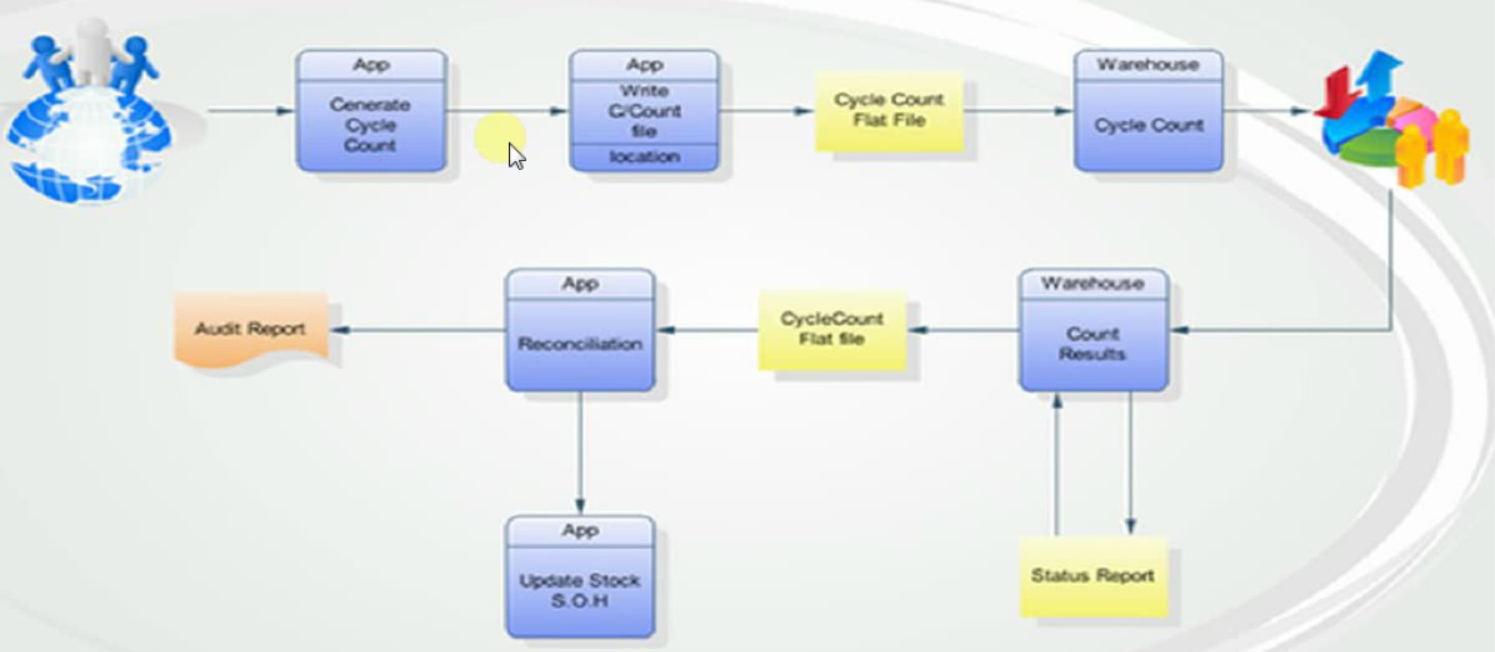
In this phase, models of the solution are created, including workflow and data flow diagrams, module and functionality layouts and any other descriptions required by the solution.

Good practice of the design phase involves vigorous review of the design artifacts to ensure requirement traceability, stability, feasibility, and reliability prior to moving into the development phase.

Workflow Example of Design Phase:



Data Flow Model Diagram: illustrates how data has been transferred.



The operating environment is established, major subsystems and their inputs and outputs are defined and processes are allocated to resources.

**Phase III: Development/Coding**

In this phase, actual software/application is developed (i.e. code is written).

From a tester’s perspective in small projects, testers may have interactive experience with the development team to get an idea for the code logic and code design to brainstorm input/output data for white box testing if required (but this is usually not the case).

Knowing code inputs and outputs still help determine functional testing data inputs and outputs also (but again, this is not really essential).

2 main types of testing performed by developers collectively known as White Box Testing—performed in DEV environment:

Unit Testing: usually test code functionality or small pieces of codes.

Integration Testing: when you integrate various units or modules together, make sure they work together.

According to industry standard, the best practice is for developers to perform Unit and Integration testing upon “code complete” or completion of the development code.

**Phase IV: Integration and Testing**

-The various components of the system are integrated and systematically tested.

-The tester tests the system to ensure that the functional requirements as defined in the functional requirements document are satisfied by the developed or modified system.

There are various environments that test can b performed.

In this stage, a QA engineer/tester will perform the following types of tests in the QA: Alpha, Beta, UAT & Production environments.

Qual-QA/Alpha Environment:

Black box Testing/Grey Box Testing

Smoke Testing/Sanity Testing

Functionality Testing/UI Testing/GUI Testing/Interface Testing/Compliance Testing

Back End Testing/Database Testing

Integration Testing

508 Compliance Testing (JAWS, Windows EYE)

Security Testing

Usability Testing

Regression Testing

Installation Testing

(Quality Assurance or QA Environment is the testing environment, dedicated specifically for testing, where actual tester do their work. In Alpha environment, small group of tester from the client side would come in your company’s site location, and interact with the application as a normal user/tester to find any bug exists.)

Beta environment:

Beta testing

(Beta environment is public environment. Before final publication, clients release beta version to public that peoples can find bugs and report those bugs what small tester teams may not be able to identify.)

UAT environment

User Acceptance Testing (UAT)

Part of UAT team

Coordinating UAT team

UAT testers from client side come in to development site to test all functionalities meet the requirements before finally acceptance the application or product. You could be a part of UAT team if you work for client side or you coordinate with UAT team when they come to your company to test the product.

**Additional environments**

Performance Test Environment:

Performance Testing

Load Testing

Stress Testing

Volume Testing

Endurance Testing

PROD environment

Smoke Testing in Production

Sanity Testing in Production

**Testing Definition:**

**Functional Testing**: verifies that the system accepts the proper data then processes that data based on the appropriate business rules.

Three major functional testing techniques:

**Positive Testing**: is the type of testing whereby each function/test case is executed using valid data parameters

and verified that expected results occur when valid data is used.

**Negative Testing**: is the type of testing whereby each function test case is executed using invalid data

parameters and verified that appropriate error/warning messages are displayed when invalid data is

used.

**Boundary Testing (Equivalence Partitioning)**: is the type of testing to determine that the application in

accepting the expected range of values and rejecting the values outside of that range. In other words, equivalence partitioning is the defining of valid and invalid data parameters and verifying the positive and negative testing scenarios.

For instance, if required password field is minimum 8 characters to maximum 32 characters, to make sure that users able to sign up with only within the range of password.

Functional Testing will also include retesting of the defects identified in prior cycles (or iteration), using similar test scripts and processes. In functional testing, execute ach function using valid and invalid data to verify the following:

a. The expected results occurs when valid data is used.

b. The appropriate error/warning messages are displayed when invalid data is used.

**Smoke Testing**: is also known as Sanity Testing and is done in order to check if the application is ready for further major testing by confirming that critical functionality is working is working properly.

**Graphical User Interface (GUI) Testing**: is the type of testing that validates the value of the display against the written specification.

GUI connects with USERS (interaction of the user with the application—usability, appearance, user friendliness etc.

**Back end/Database Testing**: is ensuring that if you enter data in the frontend or even directly in th database tables, it should b stored properly in the backend.

Similarly if you call the stored record from the front end it should display properly.

**508 compliance Testing (Disability Testing**): is performed in order to ensure that web based applications is accessible to users with disabilities such as vision impaired users.

Tools like JAWS or Windows EYE are used to conduct section 508 Compliance Testing.

**Regression Testing**: ensures that a change or bug fix did not cause faults to appear in unchanged parts of the system.

Two types/elements of regression testing:

Regression (Fix): whenever defects are fixed and deployed to system test, the test scripts directly related to those defects will be run. This effort will only cover selected data paths within the application. The effort and resources will be identified by calculating the time required to run the test scripts.

Regression (Risk): during defect fixes the development tam may unintentionally introduce new errors into the code.

The Test team will conduct an impact and risk analysis for the defects and identify additional test scripts that should be run based on code complexity, defect density, and code priority.

The impact analysis provides the number of test scripts and the level of effort to run the test scripts for the current build as well as for the previous builds.

**User Acceptance Testing**: is generally developed and performed by the client or application specialist and is conducted to determine whether a system satisfies its acceptance criteria by validating requirements have met, including the documentation and business process.

Acceptance is performed before the application is implemented into the production environment.

Based on the results of the Acceptance test, the customers determine whether to accept or decline the system.

**Performance Testing**: determines the runtime performance of the application and support infrastructure under certain conditions and is used measure several system characteristics, such as Processing speed, Response time, Resource consumption, Throughput and Efficiency.

Performance Test is conducted to identify and certify how fast the application responds and at what point of load the application performance starts degrading. The very important expectation on the performance test activity is the ability to run more realistic tests.

**Load Testing**: is testing an application server under heavy loads, such as testing of a web site under a range of loads to determine at what point the systems response time degrades or fails.

**Stress Testing**: is to find server braking point. Stress testing applies testing that measure whether the applications environment is properly configured to handle expected or potentially unexpected high transaction volumes.

**Volume Testing**: is done against the efficiency of the application.

In volume testing, huge amount of data is processed through the application in order to check the extreme limitations of the system.

**Endurance Testing**: is performed to check if the application can sustain the expected load for a prolonged or continuous timeframe. Performance degradation is also checked.

**Phase V: Implementation**

This phase is initiated after the system has been tested and accepted by the users whereby, the system is implemented into the Production environment.

The system or system modifications are installed and made operational in a production environment. This phase continues until the system is operating in production in accordance with the defined user requirements.

**Phase VI: Operation and Maintenance**

The system operation is ongoing. The purpose of this phase is to support and upgrade the software when necessary and also to fix post production bugs and issue HOT FIXES.

The system is also monitored for continued performance in accordance with user requirements and needed system modifications are incorporated.

**Disposition** activities ensure the orderly termination of the system and preserve the vital information about the system so that some or all of the information may be reactivated in the future if necessary.

Emphasis is especially given to proper preservation of the data processed by the system so that data is effectively migrated to another system or archived in accordance with applicable records management and regulations and policies, for potential future access.

Lets say you built an application for an organization. 5 years later that organization no longer needs that application because it is not up to technological standard and application needs to be enhanced or additional requirements are identified. All new system be build, because business is completely transformed to match with market demands and standards. At that point application is going to sunset (or shut down), but it has viral or critical data and you want to store that data, because data is something that can not be lost. Data is something tells company history or process and procedure. So it’s something that very critical and data has to be stored. And the event you need to access it again to make a reference to data occurrence within the system/application or system that be shut down, you can access that data if required. That is the essence of disposition phase.

**My involvement in all phases of SDLC**

I am going to discuss the SDLC scenario in real life.

Knowing theory often isn’t good enough to win over a manager in a job interview.

It’s necessary to apply these ideas to projects you have already dealt with.

The objective of these examples is to provide a visualization of real life, practical scenarios to enable you to formulate your own examples based on your experience.

**Planning: Initiation Phase**

As a Tech Lead/Senior Tester/Test Manager, in BIG projects, I have experience from an observatory and participation perspective. In SMALL projects, I have had the opportunity to b directly involved in the planning/initiation phase.

My company believed that a project in business and science is a collaborative enterprise, frequently involving research or design that is carefully planned to achieve a particular aim. So at the first meting of several projects with which I was involved as a Tech Lead/Senior Tester/Test Manager, I shared my professional view with the project management.

These meetings were a part of the initiation stage. I shared my ideas and recommendations about which objectives should b set in the early stage.

I identified the tasks to be covered by the QA/test tam, pre-determined the time span to define the tasks to b undertaken at a time, what kinds of resources can be used to meet the project objective, such as human resources, hardware, software resources, and budget.

To ensure quality, project objectives and expectations should be defined clearly in an early phase. The risk that might occur in the development phase should be defined, and there should be a strategic mitigation plan to resolve issues.

**Planning: Requirements Analysis Phase**

As a senior QA Analyst, during the requirements phase, I participated in meetings with the client, product development team, developers, designs, DBA and other resources to gather the necessary information for the project and provided my feedback.

Earlier in my career I was often an observer during this phase and began brainstorming test cases from the requirements whereas as I progressed in my career I was an active participant.

I discussed the benchmarking of the project to evaluate how current software is used.

After getting the requirements from either the Business Analyst/QA Manager/Lad or having gathered them myself (depending on phase in my career and project dynamic), I analyze the needs of the end users or customers and performed requirement assessment based on the SMART criteria.

SMART criteria:

Is the requirement **S**pecific, explicit, detailed, complete and unambiguous?

Is the requirement **M**easurable, testable in available test environment?

Is the requirement **A**ttainable, reasonable, factually correct?

Is the requirement **R**ealistic, sensible?

Is the requirement **T**imely, appropriate, suitable?

I assessed the requirement based on SMART criteria.

After performing requirements assessment, I generated the RAR (Requirement Assessment Report). In some projects, requirements were not documented properly, and I was assigned to write the requirements using Reverse Engineering approach.

In my last project which was in an Agile environment, I was involved in writing th stories (requirements) by interviewing clients, Product Management Team, designer, developers etc. in every bi-weekly release.

In the meeting with client, we decided to develop a PoC (Proof of Concept) or prototype or wireframe of the project for client approval.

After final approved requirements, I established the test criteria and began test planning based on the Requirements Document, Design Document, Project Plan, LOE (label of Effort) etc. I have included all areas where testing took place and who was responsible for the testing. I identified the testing environment, what tests were performed, test procedures, and traceability back to the requirements.

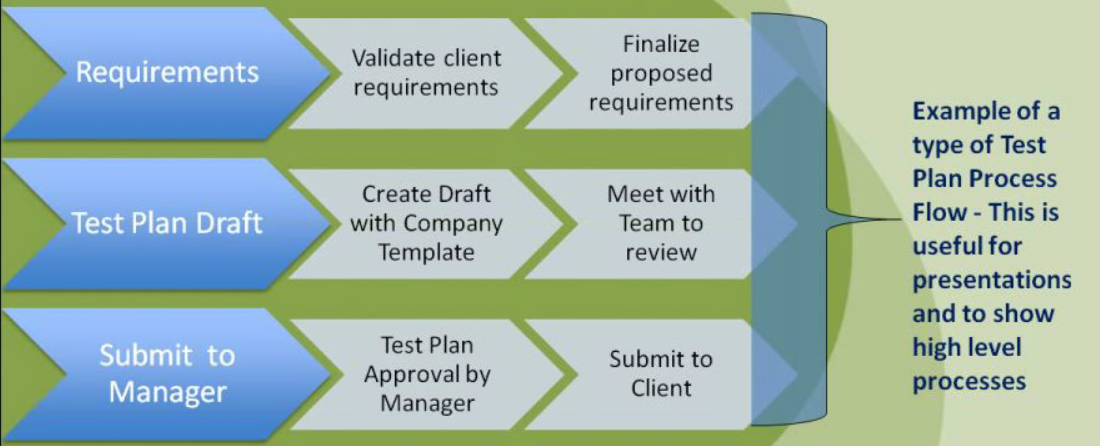
I called Test Plan review meetings: based on the feedback from the team members, I finalized the Test Plan and sent to the Project Manager for approval.

After the Project Manager’s approval, I sent the Test Plan for the Client’s approval. After incorporating the client’s feedback, the Test Plan was sent for final client acceptance.

When Test Plan got approved by the client, I “checked it in” to the shared repository and all the tam members started following the Test Plan.

Then I entered the requirements in Quality Center, and developed Test Cases in Quality Center.

Test Plan work flow:



**Design Phase**

As a QA, I know that in th software design, functions and operations are described in detail, including screen layouts, business rules, process diagrams and other documentation. The output of this stage will describe the nw software as a collection of modules.

Software design is the process or the art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements.

Designers will produce one or more ‘models’ of what they see a system eventually looking like, with ideas from the analysis section either used or discarded.

A document will be produced with a description of the system, but nothing is specific—thy might say ‘touch screen’ or ‘GUI operating system’, but do not mention any specific brands to leave options open ended.

So as a QA engineer, I participated in the CDR (Critical Design Review) process whereby I provided feedback regarding the design architecture and concepts.

**Development Phase**

Developers perform UNIT and INTGRATION test as soon as coding is complete.

Since I did Masters/Bachelor in Computer Science, I had to develop some college projects as part of my course using VB, C++, Java, HTML, so I am pretty clear on development stage.

Also since I was involved in Agile methodology, I hade to closely work with the developers.

After finalization of the design document, the developing stages begin. In the development stages, I found countless activities behind th graphics, links and functions.

In this stage I worked on software, implementation map, on line help, test plan, updated the project plan and schedule and updated the requirements traceability matrix.

The development stage takes as its primary input the design elements described in the approved design document.

For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, data management forms, data reporting formats, and specialized procedures and functions.

Appropriate test cases will be developed for each set of functionality related software artifacts, and an online help system will be developed to guid3e users in their interactions with the software.

**Integration and Testing Phase**

In the testing phase, I often performed a DRY RUN of test cases to make sure the test cases were written accurately. I also checked the actual test data and files that were provided to execute test cases.

Many times I created the test data, for instance, creating .doc, .xls, etc.

I executed all tedst cases and reported defects that were encountered. I used Quality Center or the Rational Clear Quest Tracking system to log, manage and report all the defects to the appropriate developers.

When the developer fixed the bug, the fixes should be re-tested and made sure that fixes did not create problems in other parts of the application. This leads us to regression testing.

In preparation of regression testing, I performed the required analysis to find the right candidates for regression testing by risk analysis.

As we all know, retesting a whole application is never possible because of budget, time, and resource constraints. Therefore we need to select/choose the right candidates for regression testing so that there are no adverse effects in any other parts of the application.

Sometimes bugs hide within a bug. So if we do regression testing we can eliminate more bugs and ensure the quality of the product.

I selected the right candidates for regression testing by considering ALL of the FOLLOWING criteria:

Which functionality is most important to the project intended?

Which functionality is most visible to the user?

Which functionality has the largest safety impact?

Which functionality has the largest financial impact on users?

Which aspects of the application are most important to the customer?

Which parts of the code are most complex, and thus most subject to errors?

Which part of the application were developed in rush and panic mode?

Which part of the requirements and design are unclear or poorly thought out?

What do the developers think are the highest-risk aspects of the application?

What kinds of problems would cause the most customer service complaints?

What kinds of tests could easily cover multiple functionalities?

Regression testing can be done manually but if possible, automated test scripts are used to run regression testing.

The regression test scripts are performed for each cycle within a build as well as for previous builds.

The final cycle of regression testing before production covers the test scripts that are not touched in any of the regression cycles to ensure that all the test scripts are run at least twice before production deployment.

As soon as the right regression testing candidates were selected, I consolidated all requirements to multiple scenarios, created test cases and executed the test cases manually.

Some companies that I worked for used automated testing tools. In these environments I developed the automated scripts from the scenarios using UFT/QTP/Rational Functional Tester and executed the automated scripts under AUT (Application Under Test).

We used these automated scripts for the future release also. When the new release came, we manually tested the new functionalities and used these automated scripts to check the existing functionalities.

**UAT Phase**

For a number of projects, I was art of UAT team, acted as a user, created scenarios, executed them under UAT, and logged the defects in Quality Center.

Scenario: Lets say we have 2000 requirements and we are testing PeopleNTech forum application. Some scenario or requirements are:

Users should be able to log in

Users should be able to access their profile

Users should be able to access in their profile and change their password.

If you go to PeoplNTech forum, under modify profile, we have account related settings and we can modify each of these fields. So requirement will be developed in a way that the modification of ach considered is one requirement. Like this, we have two thousands requirements. Now a scenario would be grouped requirements. Lets say first 100 requirements would be considered in one scenario. And in those requirements, what’s covered? The login functionality, the users will be able to login, and users able to modify their profiles. So when you say modify the profile, it includes modification or updating each of the profile fields. Whereas when you look up requirement separately, one requirement try to update or modification of each field.

So, Scenario creation is when you are basically requirements and grouping them and consolidating them.

In some other projects, being a Functional Tester, I also coordinated the UAT team, providing test data and necessary support.

**Post UAT Phase**

After User Acceptance Testing, I was involved in Performance testing using LoadRunner that load tests applications by emulating an environment in which multiple users work concurrently. LoadRunner replaces real users with Virtual users.

I was involved picking up th business processes that we did performance testing on. The performance testing candidates were picked for the following business processes:

Mission critical

Heavy throughput

Dynamic content

As soon as the business processes were picked, I used the standard performance testing life cycle, which follows the following:

Gathering requirements

Crating Performance Test Plan

Once the test environment (for performance testing) becomes available the following activities are performed:

Validate the test environment

Start scripting

Script shakeout (enhancement)

Crate scenarios using the LR Controller: both Manual and Goal Oriented scenarios.

Scenario execution (Baseline, Stress, Volume and Endurance testing)

Establish Monitors to monitor the test

Analyze the test:

Hit/sec

Throughput

Finally I worked on closeout documents and the TAR (Test Analysis Report).

**Implementation Phase**

I also worked on release notes and user manuals for the users.

The user manual contains all essential information for the user to make full use of the information system. This manual includes a description of the system functions and capabilities, contingencies and alternate modes of operation, and step-by-step procedures for system access and use.

After drop (Push) to the Production, I also involved performing Smoke Testing on the Production to make sure the major functionalities were working properly and there was no configuration error.

**Operation and Maintenance Phase**

I also was on the support team. Sometime I replied to the user emails and also provided them with an on line solution. Based on this support interaction with uses, I can claim that I was a part of the application maintenance team.

When users found issues, they sent an email via our feedback section. Reading users email with the complaints or feedback I replied describing/showing how to navigate the system properly or asked them to see our HELP section if they experiences navigation confusion.

This is a diagram of Bug Cycle:

(29:50 in 2nd video)

It means, if you find bug, report then, resolve the bug, and analyze the bug/issue.

SLA stands for Service Level Agreement, which is basically a contractual agreement between the contractor and the client to ensure that services perform according to contractual agreement.

If we found a issue and issue was a defect, I logged the defect using Quality Center and assigned to the developers. If the defect was a SHOW STOPPR/High Visibility defect, then the developer fixed it right away and the DBA pushed it to Production as a HOT FIX. If the defect did not have high severity, then the Developer would address it in the upcoming release.

This is a comprehensive example of how a Quality Assurance Engineer would b involved in all phases of SDLC. But in interview you would have own story to tell.

**SDLC** (Software/System Development Life Cycle)

SDLC is an overall process by which software applications and/or systems are developed.

There are five core SDLC phases:

1. Planning & Requirement Analysis

3. Design

4. Development

5. Integration and Testing

6. Operation and maintenance

1a. Planning

i. Initiation

a. Identification of business need: analyze the benefits/return on investment, realized business efficiencies, affordability.

b. Acquire upper management approval: funding is required before proceeding.

ii. System Concept Development

a. Develop the concept of the business need by identifying and evaluating all aspects of the concept/business need: Concept development also involves describing how the business will operate once the approved system is implemented.

b. To ensure the products/services provide the required capability on-time and within budget, project resources, activities, schedule, tools and reviews are also defined in this stage.

(Comment: As a Tech Lead, Senior Tester or Manger, I have experience from an observatory and participation perspective.)

1b. Requirements Analysis

i. A GOOD Requirement follows the S.M.A.R.T. criteria:

a. Specific (clear and unambiguous)

b. Measurable (quantifiable, computable)

c. Attainable/Achievable (within reach, reasonable)

d. Realistic (practicable, sensible, rational)

e. Timely (appropriate, well-timed, apt, suitable)

ii. As a tester/test lead/QA specialist, I wrote requirements:

a. In BIG projects, Business Analysts are dedicated to this activity. In SMALL projects, QA tester/specialist/manager will have to write the requirements themselves thereby working in a business analyst capacity

1. Requirements are developed by interviewing end users/clients/developers/other team members.

2. Employ Reserve Engineering approach

(Comment (2): Performed requirement assessment analysis as a Tester/Test Lead by evaluating each requirement according to SMART criteria.

Comment (2ia2): You have the application but documented requirements are not available. Application is developed in agile type environment where features were added functionality enhancement )

2. Design (phase is really performed by a collective efforts with the developers and the IT architects who are basicly the visionary of the projects.)

i. Models of the solutions are crated: including workflow & data flow diagrams, module and functionality layouts and any other descriptions required by the solution. Good practices of the design phases involves vigorous review of the design artifacts to ensure requirement traceability, stability, feasibility, and rliability prior to moving into the development phase.

3. Development

i. Actual software application is developed i.e. code is written.

ii. Testers are not involved in this phase, but many testers may have development experience. BUT from a tester’s perspective in small projects testers may have interactive experience with the development tam to get an idea for the code logic and code design to brainstorm input/output data for white box testing if required (but this is usually not the case), but knowing code inputs & outputs still help determine functional testing data inputs & outputs also (but again this is not really essential).

4. Integration & Testing

i. White Box Testing: usually performed by Developers or highly skilled testers on large projects.

a. Advantages:

1. As the knowledge of inyernal coding structure is pre-requisite, it becomes very easy to find what types of input data can help in testing the application effectively.

2. Helps in optimizing the code

3. Helps in removing th extra lines of cod which can introduce hidden defects.

b. Disadvantages:

Since knowledge of code and internal structure is a pre-requisite, a skilled tester is required to perform this type of testing which can be quite costly, hence this is usually done by the developers, who test their own units of code.

It’s also nearly impossible to look into every bit of code to find hidden errors that can cause critical application failures, hence functional testing is more practical.

ii. Unit Testing

a. Testing small units of code functionality: inputs and outputs are tested.

iii. Integration Testing

a. Testing several modules integrated together.

Example: Banking Application

- Login Module

- Current Balance Module

i. Deposit Module

ii. Withdraw Module

iii. Transfer Module

b. Dev: integration, QA Integration Test Environment, Beta, UAT, PROD Integration & Testing.

iv. Black Box Testing

a. Smoke Testing: testing major functionality to determine if further in depth testing can be performed.

b. Functional Testing/UI Testing/GUI Testing/Compliance Testing: testing all functional rquirements.

1. Positive Testing

2. Negative Testing

3. Boundary Testing/Equivalence Partitioning

- Testing outside the boundaries of the acceptable/defined data parameters.

4. GUI (Graphical User Interface) Testing

i. validates the value of the display against the written specification.

ii. GUI connects with the USERS (interaction of the user with the application: usability, appearance, user friendliness etc.

v. Back End/Database Testing

a. Database Testing is ensuring that if you enter data in the front end or even directly in the database tables, it should be stored properly in the back end.

b. Similarly if you call the stored record from the front end it should display properly.

- Examples:

- Google Search

- Username and password authentication

- Mail application

vi. 508 Compliance Testing (JAWS, Windows EYE): disability testing

vii. Regression Testing

a. Regression (fix): re-test after defects are fixed.

b. Regression (Risk): during bug fixes, developers may unintentionally introduce new bugs into the system, i.e. fixing one part of the application breaks another part—evaluate areas of the application that have regression risk potential for consideration during regression testing. Development in “rush & panic” mode also introduces unintentional bugs.

viii. Alpha Testing

A group of users are invited to test the application with th presence of developers who not very particular input or action carried out by the user. Example for a banking application: a group of employees are selected to interact /test the application with the in house developer.

ix. Beta Testing

Application is released to a simple group of users to test the application for bugs before actual release in production.

x. User Acceptance Testing

a. Part of UAT team: you test as functional/UAT/user tester.

b. Coordinating UAT group: support foreign UAT group.

xi. Smoke testing in production

Testing major/critical functionality in the production environment to ensur that the application is operational in production.

xii. Performance Testing

a. Testing with the intent of determining how quickly a product handles a request. Performance tests are conducted in a production-like test environment with full-volume data. These tests validate the application, hardware, and network behavior under a user load that is based on production estimates.

b. Performance tests are used to measure several system characteristics, such as processing speed, response time, resource consumption, throughput, and efficiency. Performance testing is sometimes coupled with stress testing.

xiii. Stress/Load Testing

a. Team often used interchangeably with “load” and “performance” testing. Used to measure how well a product/application performs when a load (data volume/user requests) is placed on the system resources that nears that then exceeds capacity.

b. The application is tested against heavy load such as complex numerical values, large number of queries etc. which checks for the stress/load the application can withstand.

xiv. Volume/Endurance Testing

Volume testing is done against the efficiency of the application. Huge amounts of data are processed through the application in order to determine the extreme limitations of the system.

5. Production/Implementation: when software has been thoroughly tested, you had some User Acceptance Testing, and the development environment where users agreed the software beta test and alpha test, users comment to you development environment non-production and they sit with your development team and testing team and test the application to make sure everything is perfect.

When this is done, it goes to implementation. It means you take the developed application and you implement in clients’ existing infrastructure, and you put it in their IT architecture. This application is now fit in their existing overall business process and overall flow. Since, in most cases you are not developing standalone application. Even if it is a standalone application, still need to fit in their system.

i. Operational Maintenance

a. The system operation is ongoing.

b. The system is monitored for continued performance in accordance with user requirements, and needed system modifications are incorporated.

c. The operational system is periodically assessed through in-process review to determine how the system can be made more efficient and effective.

d. This phase continues as long as the system can be effectively adapted to respond to an organization’s needs.

e. When modifications or changes are required, the system may re-enter the planning phase.

ii. Disposition

a. Disposition activities ensure the orderly termination of the system and preserve the vital information about the system so that some or all of the information may be reactivated in the future if necessary.

b. Emphasis is especially given to proper preservation of the data processed by the system so that data is effectively migrated to another system or achieve in accordance with applicable records management regulations and policies, for potential future access.

(Comment (6ii): You might have question that you developed an application, then why do you need to terminate it. This concept is called sun-setting of an application.

Lets say you developed an application and five years later, that application is no longer valid, because technology is ever evolving, new requirements always growing, new needs are identified. Technology become absolute.

For example, your five years old computer might not compatible for all new software you need to install due to insufficient RAM, and older version of operating system, so you buy a new computer with upgraded RAM and updated version of Operating System. But you do not want loose data. Either you want to keep that data or transfer data over to your new system. So you can continue work with that data. Data is most important part of life, data is something we accumulate. Data is something we cannot effort to loose, specially for large scale projects or organizations. This is the importance of disposition phase.)

**Test Methodology**

**Waterfall**

Popular with new Managers because it facilitates project estimation—everything is orderly. Everything is planned upfront, making it easy to determine expected costs, timelines/schedule.

Requirement must be clearly established from the beginning, no options to change. Requirements are validated and the exit criteria and measurements for success are clearly defined before code is developed.

This method assumes that it’s possible to capture all requirements and complete analysis before design starts. This is highly impractical for most projects (as requirements are always evolving) i.e. by the time all of the rquirements are gathered, they may have changed.

Work is performed in stages so only one team at a time manages the project which simplifies the manager’s job.

**Iterative**

Also known as spiral life cycle.

In the iterative/spiral method analysis is done in the same way as in the Waterfall method.

In this method we define the software output through separating requirements, analysis, developing and testing phases (so the core SDLC phase—Requirement Analysis through Testing are repeated each iteration).

Iterative development is an approach in building software or applications in which the overall lifecycle is composed of several iterations in sequence. Each iteration is a self-contained mini project composed of activities outlined in the above step. The goal for the end of an iteration is an iteration release, a stable, integrated and tested partially complete system.

In modern iterative methods, the recommended length of one iteration is between one and six weeks.

Advantages:

1. The design phase goes much faster, as designs are only done on the items in the current release.

2. Coding & Testing goes much faster because there are less items to code and test. If major design flaws are found, re-work is much faster since the Functional areas have been greatly reduced.

**Agile**

Agile projects represent a complete paradigm shift from other methodologies. All agile complaint life cycle require early software delivery, minimal management, minimal documentation, complete test platforms, and heavy customer involvement throughout the process. Unlike most processes, changing requirements are welcomed and encouraged, even late in the development cycle.

**Scrum**

**Deliverables**

FRD (Functional Requirements Documents)/BAD (Business Analyst Documents)

Business Rules (for Business Analysts)

Design Documents (for Developers)

Design Mock Up (for Developers)

Prototype (Demo using main/major requirements)

PoC (Proof of Concept)

Wire frame

Use Case

A description of a system’s behavior as it responds to a request that originate externally from the system—a use case describes interaction of users with the application or system—i.e. “who” can do “what” with the system application. The use case techniques is used to capture a system’s behavioral requires detailing scenario-driven threads through the functional requirements.

RTM (Requirement Traceability Matrix)

A document used to check to see if the current project requirements are being met. Additionally traceability is a mapping of test cases to the requirements—i.e. which test case covers which system requirement.

Test Matrix (Test Report)

User Manual

Software/application user guide

Master Test Plan

Global Test Plan: not specific to any particular projects—but applicable organization wide for ALL projects.

Test Plan

Planning of testing effort.

Some have manual (functional test plan).

Some have automation test plan.

Some have regression test plan.

Automation Test Plan

Test plan created to automates testing (i.e. script development, automated script running schedules, etc)

Performance Test Plan

Test Plan specific to performance testing

Test Strategy

High level view of test plan (i.e. which methodology)

Test Case

A test case in software engineering is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not.

Example: Login test case, Transfer balance test case, etc.

Test Script

Automated code required to.

Test Scenario

Consists of multiple requirements.

Example: transferring funds in banking application includes

a. login requirements

b. account balance requirements

RAR (Requirements Assessment Report)

TAR (Test Analysis Report)

A formalized report that documents the outcome of the testing activities. It summarizes the test results and the defects found during the process.

Executive Summary

A short document or section of a formalized final reports i.e. Test Plan that summaries a longer report, proposal or group of related reports in such a way that radrs can rapidly become acquinted with a large body of material without having to read it all.

LOE

Post Mortem Report

A post mortem meeting occurs at the end of a project or full testing iteration to discuss lessons learnd, areas for improvement and activities that were successfully executed and good standard for future projects. A post mortem report is a short summary (can be a list).

Meeting Minutes

Notes/action items from team meeting

Release Notes

Responsibility of development team.

Release notes are documents that are distributed with software products, often when the product is still in the development or test state (e.g. a beta release). For products that have already been in use by clients, the release note is a supplementary document that is delivered to the customer when a bug is fixed or an enhancement is made to the product.

Fish Bone Diagram

**Validation and Verification**

**CMM/CMMI** (Capability Maturity Model/ Capability Maturity Model Integration)

Standard process invented by Software Engineering Institute (SET).

Very few companies in the world hold a CMM level-5 certification

Five Maturity Levels (characteristics)

Level I: Initial

The software process is characterized as ad hoc, and occasionally even chaotic. Few process are defined, and success depends on individual

Level II: Repeatable

Basic project management processes are established to track cost, schedule and functionality. The necessary process discipline is in place to repeat earlier success on projects with similar applications.

Level III: Defined

The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization’s standard software process for developing and maintaining software.

Level IV: Managed

Detailed measures of the software process and product quality are collected. Both the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

Level V: Optimizing

Continuous process improvement is enabled by quantative feedback from the process and from piloting innovative ideas and technologies.

**Other concepts:**

Six Signs: implemented in commercial product development projects/organizations

Configure Management (CM) tools

Rational Clear Case is a CM tool

Version Control Tools

MS Sharepoint

Private Network Drives

Organizational Project Tam Rooms: IBM Lotus Team room

**SDLC (SOFTWARE DEVELOPMENT LIFE CYCLE)**

1. PLANNING/REQUIREMENTS GATHERING
2. DESIGN
3. DEVELOPMENT
4. TESTING
5. PRODUCTION
6. MAINTENANCE

**Environment:**

1. DEV Environment
2. QUAL/QA/ALPHA Environment
3. BETA/UAT
4. PROD

**TYPES OF TESTING:**

1. **UNIT TESTING**-**WHITE BOX TESTING**-DONE BY DEVELOPER-Dev Environment

-- USER SHALL BE ABLE TO LOGIN TO PEOPLENTECH FORUM.

1. **INTEGRATION TESTING**- **WHITE BOX TESTING**-DONE BY DEVELOPER - Dev Environment

-AN ADMIN SHALL BE ABLE TO CREATE A USER ACCOUNT.

-THE CREATED USER SHALL BE AVAILABLE IN THE MEMBERS SECTION.

1. **SMOKE/SANITY TESTING**- QUAL/QA/ALPHA Environment

-Log into PeopleNTech forum

-Click on the followings:

Home tab

Help tab

Search tab, then Search a string like ‘QTP’

Admin tab

Moderate tab

Profile tab

My Messages tab

Calendar tab

Members tab

Logout tab

1. **FUNCTIONAL TESTING**-QUAL/QA/ALPHA Environment

USER SHALL BE ABLE TO LOGIN TO PEOPLENTECH FORUM.-PASS

USER SHALL BE ABLE TO CREATE A TOPIC USING PEOPLENTECH FORUM. -PASS

USER SHALL BE ABLE TO MODIFY HIS/HER OWN TOPIC USING PEOPLENTECH FORUM. -PASS

USER SHALL BE ABLE TO DELETE HIS/HER OWN TOPIC USING PEOPLENTECH FORUM. -PASS

1. **POSITIVE TESTING**-QUAL/QA/ALPHA Environment

USER SHALL BE ABLE TO LOGIN TO PEOPLENTECH FORUM.

-ABLE TO LOGIN WITH AUTHORIZED/VALID USERID AND PASSWORD

--Enter valid user id and valid password, click on Login—User logs in and welcome page displays

1. **NEGATIVE TESTING**-QUAL/QA/ALPHA Environment

USER SHALL BE ABLE TO LOGIN TO PEOPLENTECH FORUM.

-UNABLE TO LOGIN WITH UNAUTHORIZED/INVALID USERID AND PASSWORD

-Leave blank user id and password fields, click on Login—Error displays

-Enter invalid user id and wrong password, click on Login—Error displays

-Enter valid user id and wrong password, click on Login—Error displays

-Enter valid user id and no password, click on Login—Error displays

1. **EQUIVALENCE PARTITIONING/BOUNDARY TESTING**-QUAL/QA/ALPHA Environment

USER SHALL BE ABLE TO REGISTER TO YAHOO.

-In the password, the condition is, password must be between 8 to 32 characters.

Then boundary testing conditions are:

1. Create a password with any number of characters which is in between 8 to 32 characters like 10 characters.-POSITIVE TESTING
2. Create a password with less than 8 characters like 7 characters.-NEGATIVE TESTING
3. Create a password with more than 32 characters like 33 characters. -NEGATIVE TESTING
4. **Role Based Security Testing**-QUAL/QA/ALPHA Environment

-User has access to certain functionalities, does not have admin privs

-A USER SHALL NOT BE ABLE TO CREATE A USER ACCOUNT.

-USER SHALL BE ABLE TO MODIFY HIS/HER OWN TOPIC USING PEOPLENTECH FORUM. -

-USER SHALL BE ABLE TO DELETE HIS/HER OWN TOPIC USING PEOPLENTECH FORUM. -

-Admin has access to certain functionalities that user should not have, like Members Registration, Banning members, changing members password, deleting members etc.

- AN ADMIN SHALL BE ABLE TO CREATE A USER ACCOUNT.

-An Admin SHALL BE ABLE TO MODIFY ANY TOPIC USING PEOPLENTECH FORUM. -

- An Admin SHALL BE ABLE TO DELETE ANY TOPIC USING PEOPLENTECH FORUM. -

1. **INTEGRATION TESTING**-QUAL/QA/ALPHA Environment

-AN ADMIN SHALL BE ABLE TO CREATE A USER ACCOUNT.

-THE CREATED USER SHALL BE AVAILABLE IN THE MEMBERS SECTION.

1. **DATABASE TESTING**-QUAL/QA/ALPHA Environment

SELECT \* FROM table\_name WHERE condition;

SELECT \* FROM `smf\_members`

WHERE `member\_name`='crazytester';

**What is Testing?**

- Software testing is to identify and correct/fix any bugs/problems of an application or software.

- Software Testing is an activity to check whether the actual results match the expected results and to ensure the software system is bug free.

Tester plays the role of an end user while testing.

**Manual Testing**

- Manual testing is the process of manually

The goal of manual testing is to ensure that the application is error free and it is working in conformance to the specified functional requirements.

Manual testing is done to make sure that reported defects are fixed by developers and re-testing (regression testing) has been performed by a tester on the fixed defects. Basically, this testing checks the quality of the system and delivers bug-free product to the customer.

Automation is done on a more stable application.

**There are different types of manual testing:**

**Unit Testing**: This initial stage of testing normally carried out by the developer who wrote the code and sometime by a peer/co-worker using the white-box testing technique.

Unit testing is done to test the codes by the developers.

**Integration testing**: Integration testing is done by the developers by combining the codes together to see if they are working together (as one application) and meet the requirements.

After the integration testing the manual tester or automator perform software testing on complete software build. This software testing consists of two types of testing:

1. **Functional Testing** is done to check whether Software under testing (SUT) is working as per the functional requirements. i.e. Database Testing.

2. **Non-Functional Testing** is to check whether Software under testing (SUT) is working as per the non-functional requirements. I.e usability, compatibility, configuration.

**White-box testing**: Also known as glass box, structural, clear box and open box testing. White-box testing is done by testers/developers who has internal knowledge of the coding. This type of testing uses specific knowledge of programming code to examine outputs.

**Black-box testing**: black box testing is a software testing techniques in which functionality of the software under test (SUT) is tested without looking at the internal code structure. This type of testing is based entirely on the software requrements and specifications.

**Gray Box Testing**: Gray box testing is a combination of white-box testing and black-box testing. The goal of this testing is to serch for the defects if any due to improper structure of improper usage of application.

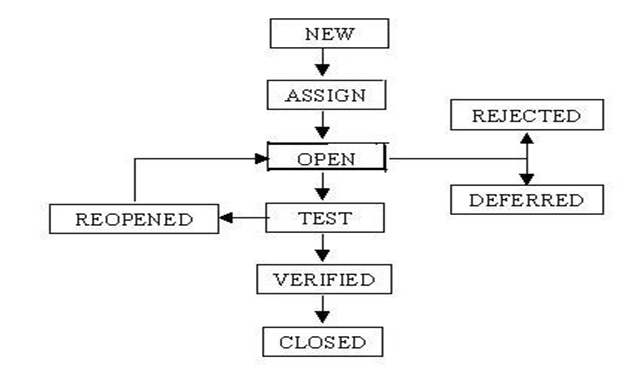
**Regression Testing**: regression testing is done after code fixes, upgrade or any other system maintenance to check the new code/fix has not affected the existing code.

**Bug Life Cycle**

Definitions:

1. **Bug**: Bugs are errors/flaws that are found prior to production.
2. **Defect**: Defects are errors/flaws that are found after the production.

Stages of bug life cycle:



1. **New** : When a bug/defect is detected for the first time
2. **Assign**: After the tester has logged in or posted the defect, the lead of the tester approves that the bug is genuine and he assigns the bug to corresponding developer and the developer team. And changes its status to assigned.
3. **Open**: At this stage the developer has started to analyzing and working on the defect.
4. **Rejected**: if the developer feels/sees that the bug is not genuine or he can not regenerate the bug in his environment, he rejects the bug and changes the status to rejected.
5. **Deferred:** the bug, changed to deferred state means the bug is expected to be fixed in next release. And this could happen for many factors, it might happen because the bug might have low impact on the application.
6. **Fixed:** when developer makes necessary code changes the he/she can make bug status as fixed and the bug is passed to the testing team again to retest.
7. **Verified:** The tester retests the bug again after it got fixed by the developer. If the bug is not present in the software, the tester approves that the bug is fixed and changes its status to verified.
8. **Closed:** Once the bug is fixed, it is tested by tester again and if the tester can not reproduce the bug/defect in the application anymore. He changes the status to closed.

**Unit Testing**

This is a white box testing technique where the developer has to look into the code and find out which unit/statement/chunk of the code is malfunctioning and hence is needed to possess knowledge of coding and logic i.e. internal working of the code.

|  |  |
| --- | --- |
| Test Objective: | The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. The test is accurate only if the tester knows what the program is supposed to do. |
| Technique: | Execute each function, using valid and invalid data, to verify the following:   * The expected results occur when valid data is used. * The appropriate error / warning messages are displayed when invalid data is used. * Each business rule is properly applied. * Each requirement is properly implemented. |
| Completion Criteria: | * All planned tests have been executed. * All identified defects have been addressed and reconciled. |
| Special Considerations: |  |

Table 1: Unit Testing

### Integration Testing

Integration testing is a logical extension of unit testing. In its simplest form, two units that have already been tested are combined into a component and the interface between them is tested.

Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests those aggregates, and delivers as its output the integrated system ready for system testing.

|  |  |
| --- | --- |
| Test Objective: | Integration testing identifies problems that occur when units are combined. To ensure the viability of each before combining units, developers know that any errors discovered when combining units are likely related to the interface between units. |
| Technique: | * Testing in which software components are combined and tested progressively until the entire system has been integrated. * The expected results occur when valid data is used. * The appropriate error / warning messages are displayed when invalid data is used. * Each business rule is properly applied. |
| Completion Criteria: | * All planned tests have been executed. * All identified defects have been addressed and reconciled. |
| Special Considerations: |  |

Table 2: Integration Testing

### Smoke Testing

Smoke is the initial level of testing effort to determine if the new software version is performing well enough for its major level of testing effort.

|  |  |
| --- | --- |
| Test Objective: | The Test Team will navigate through the main application functionality to ensure build stability and to accept the build for additional system testing efforts. |
| Technique: | * It verifies the major functionality at high level in order to determine if further testing is possible. * The Smoke test scenarios should emphasize breadth more than depth. * All components should be touched, and every major feature should be tested briefly. * If test fails, the build is returned to developers un-tested. |
| Completion Criteria: | * All planned tests have been executed. * All identified defects have been addressed and reconciled. |
| Special Considerations: |  |

Table 3: Smoke Testing

### Functional Testing

Functional testing verifies that the system accepts the proper data and processes and retrieves the data based on the appropriate business rules. Function Testing will also include retesting of the defects identified in prior cycles, using similar test scripts and processes.

|  |  |
| --- | --- |
| Test Objective: | Ensure proper functionality, including navigation, data entry, processing, and retrieval. |
| Technique: | Execute each function, using valid and invalid data, to verify the following:   * The expected results occur when valid data is used. * The appropriate error / warning messages are displayed when invalid data is used. * Each business rule is properly applied. * Data integrity rules are followed. * Data access rules are followed. * There is a logical progression from one module to the next. * Standardization of screen layouts, buttons, and data fields is present. * Each requirement is properly implemented. * Whenever defects are fixed and deployed to system test, the test scripts directly related to the corresponding defects will be run. |
| Completion Criteria: | * All allocated requirements listed in the RTM is properly executed. * All identified defects have been addressed and reconciled. |
| Special Considerations: | Prior to the testing, make sure that the Database Management System is running and the IDA system application functions without any severity level ‘High’ problems. |

Table 4: Functional Testing

### Role Based Security Testing

Role based security ensures that IDA system users are restricted to specific functions based on the roles that are available to them. These tests will be done based on the IDA system functional requirements and design documents.

|  |  |
| --- | --- |
| Test Objective: | Application level Security: Verify that IDA system users can access only those functions / data for which their user type is provided permissions based on the functional and design documents. |
| Technique: | * Identify and list each IDA system user type and the functions / data each type has permissions for. * Create tests for each IDA system user type and verify the each permission by creating transactions specific to each user actor. * Modify user type and re-run tests for the same users. In each case verify those additional functions / data are correctly available or denied. |
| Completion Criteria: | For each known IDA system user type, the appropriate function / data are available and all transactions function as expected and run in prior function tests. |
| Special Considerations: |  |

Table 5: Role Based Security Testing

### Regression Testing

Regression testing ensures that a change or fix has not caused faults to appear in unchanged parts of the system.

**How do you perform regression testing?**

Regression testing verifies the changes applied to the software during development do not have a negative impact on existing functionality. Regression test cases will be developed in such a way that the test covers overall functional scenarios of the application. There are two types of regression testing that need to be performed for each build of the code deployed to system test environment.

Regression (Fix): Whenever defects are fixed and deployed to system test, the test scripts directly related to those defects will be run. This effort will only cover selected data paths within the application. The effort and resources will be identified by calculating the time required to run the test scripts.

Regression (Risk): During defect fixes the development team may unintentionally introduce new errors into the code. The Test team will conduct an impact and risk analysis for the defects and identify additional test scripts that should be run based on code complexity, defect density, and code priority. The impact analysis provides the number of test scripts and the level of effort to run the test scripts for the current build as well as for the previous builds. The listing of files and versions of design artifacts and code included in each build will be designated in the Version Description Document (VDD).

For both of the above regression tests, current test cases may be re-used and new regression test cases will be developed and added to the regression deck, if needed. Wherever possible, automated test scripts will be used to run regression testing. The above mentioned regression tests will be performed for each cycle within a build as well as for previous builds. The final cycle of regression testing before production will cover the test scripts that are not touched in any of the regression cycles to ensure all the test scripts are run at least twice before production deployment.

|  |  |
| --- | --- |
| Test Objective: | Regression testing verifies the changes applied to the software during development do not have a negative impact on existing functionality. Regression test cases will be developed in such a way that the test covers overall functional scenarios of the application. |
| Technique: | During defect fixes the development team may unintentionally introduce new errors into the code. The Test team will conduct an impact and risk analysis for the defects and identify additional test scripts that should be run based on code complexity, defect density, and code priority.   * The right candidates for regression testing can be selected by considering: * Which functionality is most important to the project intended? * Which functionality is most visible to the user? * Which functionality has the largest safety impact? * Which functionality has the largest financial impact on users? * Which aspects of the application are most important to the customer? * Which parts of the code are most complex, and thus most subject to errors? * Which part of the application were developed in rush and panic mode? * Which part of the requirements and design are unclear or poorly thought out? * What do the developers think are the highest-risk aspects of the application? * What kinds of problems would cause the most customer service complaints? * What kinds of tests could easily cover multiple functionalities? * Wherever possible, automated test scripts will be used to run regression testing. * The regression test scripts will be performed for each cycle within a build as well as for previous builds. * The final cycle of regression testing before production will cover the test scripts that are not touched in any of the regression cycles to ensure all the test scripts are run at least twice before production deployment. |
| Completion Criteria: | * All planned tests have been executed. * All identified defects have been addressed and reconciled. |
| Special Considerations: |  |

Table 7: Regression Testing

### 

**508 Compliance Testing**

|  |  |
| --- | --- |
| Test Objective: | Section 508 Compliance Testing ensures that the software meets 508 Compliance requirements. |
| Technique: | The Section 508 Compliance requirements will be verified using a two-step approach to assessing compliance of web pages.  **Step 1.** Consists of a manual assessment, using prepared 508 test scripts and Section 508 Web Standards Compliant Evaluation Forms.   * Test scripts and evaluation forms will prompt the tester to review the web page for each of the 16 Section 508 Web Site standards. * 508 test scripts will have a combination of user and code checkpoints. * The user checkpoints assist in the assessment of a site on the web from a user’s perspective.   The code checkpoints assist in the assessment of a web page’s source code from a developer’s perspective.  After running each 508 test script, results will be entered in the Section 508 Web Standards Compliant Evaluation Forms ([Appendix C](#_Appendix_C._Section)). Results will be reported as Fully Compliant (FC), Partially Compliant (PC), Non-Compliant (NC) or Not Applicable (N/A).   * Fully Compliant indicates that every instance of the check is compliant. For example, if every image on the page has alt tags associated with it, check 1 would receive a Fully Compliant. * Partially Compliant indicates that only some of the instances of the check are compliant. Using the example from above, if only some of the images have alt tags, check 1 would receive a Partially Compliant. * Non-Compliant indicates that none of the instances of the check are compliant. Again using the example from above, if none of the images have alt tags, check 1 would receive a Non-Compliant. If no instances of the check exist on the page, the check would receive a Not Applicable.   **Step 2.** Involves evaluating the web page with assistive technology such as a screen reader (JAWS 4.5) and noting any issues and comments related to the compatibility on the evaluation form and within TestDirector. |
| Completion Criteria: | All planned tests have been executed.  All identified defects have been addressed and reconciled. |
| Special Considerations: |  |

Table 6: 508 Compliance Testing

### Performance/Load/Stress Testing

Performance tests are conducted in a production-like test environment with full-volume data. These tests validate the application, hardware, and network behavior under a user load that is based on production estimates.

Load testing provides flexible user scenarios and realistic load types to help application load testing. The load tests can help simulate real time user activities and load test hundreds or even thousands of concurrent users under dynamic load conditions.

Stress testing is seen as part of the process of performance testing which tries to identify the breaking point in a system under test by overwhelming its resources or by taking resources away from it. This helps to ensure that the system fails and recovers gracefully.

|  |  |
| --- | --- |
| Test Objective: | The IDA application will be tested for consistent performance against expected data volumes and user load by simulating the load conditions using HP’s LoadRunner. This will help to identify issues or defects that cannot be measured during the functional testing efforts, such as application functionality issues, system configuration issues, memory leaks in the code, or other issues that may go unnoticed with a simulated system user load. |
| Technique: | * In order to create a well-rounded performance testing effort, the test team will identify highly utilized critical business paths a user would take in the system. The list of paths will be captured in the Performance Test Scenario. * Performance testing tool LoadRunner allows to perform Load testing/ Stress testing of web applications/web sites with accuracy and ease. Comprehensive reports helps to identify critical performance issues and optimize the user experience of the web applications/web sites before it goes into deployment. * Stress Testing is done to uncover memory leaks, bandwidth limits, transactional problems, resource locking, hardware limitations and synchronization problems that occur when an application is loaded beyond the limits determined by the performance statistics. * Stress testing tools generate extremely high load on the Web server by simulating multiple client connections. * LoadRunner is able to simulate this load by using a “virtual user (vUser)”. LoadRunner only sends and receives the CALL between the server and client, never actually displaying them in the browser. * After each phase is completed, the test team will generate a Performance Test Report based on the data collected during the test. |
| Completion Criteria: | * + All planned tests have been executed.   + All identified defects have been addressed and reconciled. |
| Special Considerations: | Prior to the testing, make sure that the Database Management System is running and the IDA system application functions without any problems. |

Table 8: Performance/Load/Stress Testing

### Government Acceptance Testing (GAT)

Acceptance testing is designed to allow the customer to validate that the functionality not only meets requirements but also that the application performs as intended. The objective is to provide confidence that the delivered system meets the business requirements of both sponsors and users. The acceptance phase may also act as the final quality gateway, where any quality defects not previously detected may be uncovered.

|  |  |
| --- | --- |
| Test Objective: | The Government Acceptance Test (GAT) is testing that client performs to verify compliance with the IDA functional requirements. This method is the last level of testing to identify and eliminate fatal errors and other system functionality problems. |
| Technique: | * Testing is conducted in context of the entire system and is carried out in an environment that closely resembles the real world or true production environment. * BSD, CI Research, NOC, and CI designated user representatives will design and conduct User Acceptance Testing. They will be supported by the IBM Test Team as needed. * It is expected that a scenario-based approach will be used. This approach is one in which the testers create and execute test cases/scripts that are based on real life situations. * The goal of this testing is to identify any and all application errors prior to deployment of the System to the production environment. |
| Completion Criteria: | * + All planned tests have been executed.   + All identified defects have been addressed and reconciled. |
| Special Considerations: | Prior to the testing, make sure that the Database Management System is running and the IDA system application functions without any severity level ‘High’ problems. |

Table 9: Government Acceptance Testing (GAT)

**SDLC (Software Development Life Cycle)**

Step1 User groups/SME (Subject matter expert) identifies and documents the need for an IT solution with URS

(User Requirement Specifications)

Step2 When BA (Business Analyst)/RA (Report Analyst)/FA (Function Analyst) receive URS, they make SRS

(System Requirement Specifications)/FRS (Functional Requirement Specifications)/BRS (Business

requirement Specifications)

Step3 Design group/SA (System Analyst) make SDD (Software Design Documents)/TDD (Technical Design

Documents). Design includes detailed functional and technical to support development and testing in later

phase.

Step4 Development: Developer/Programmer/SW Engineer write codes and assemble software components to

create the system.

UT (Unit Test) and IT (Integration Test) are executed.

Step5 System Testing: the completed system is independently tested by TE (Test Engineer) that system meets

the SRS requirement.

Step6 UAT (User Acceptance Test): Users perform acceptance activities to confirm that the system, used in

conjunction with operational procedures, meets the documented user requirements.

Step7 Deployment: Sites that will receive the system are prepared. Activities may include infrastructure

installation, end user training, modifications of operational procedures, preparation of data for migration,

etc.

Step8 Maintenance

**STLC (Software Testing Life Cycle)**

Step1 Analyze SRS/FRS/URS/SDD: go through SRS, try to figure out background, make sure all requirements are unique, bounded, and create RAR (Requirement Analysis Report)

Step2 Create Test Plan: (Test Plan is a document that describes the entire test process including objectives, scopes, approach, and focus of a software testing effort.)

Step3 Create (S)TCP (Software Test Case and Procedures): (TCP is a document that describes necessary steps to execute test.)

Step4 Prepare Test data: (basically business owner/DBA give you test data)

Step5 Create (S)RTM (Software Requirement Trace-ability Matrix/mapping)

Step5 Set up test environment: (OS, HW, SW)

Step6 Execute test

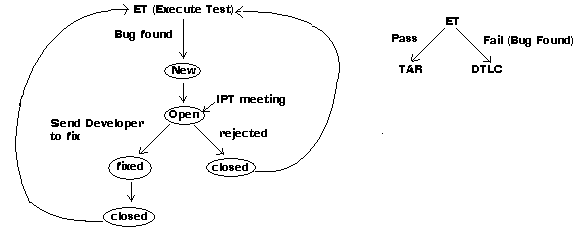
Step7 Meetings: PEER meeting

IPT (Integrated Product Team: BAs, Designers, Developers, Testers) meeting

Step8 TAR (Test Analyst Report): The overall report of testing, includes how many requirements covered, how

many bug found, meeting, bug fixed/not fixed, etc.)

**DTLC (Defect Tracking Life Cycle)**



**What is a bug/error?**   
A bug or error in software product is any exception that can hinder the functionality of either the whole software or part of it.   
**How do I find out a BUG/ERROR?**   
Basically, test cases/scripts are run in order to find out any unexpected behavior of the software product under test. If any such unexpected behavior or exception occurs, it is called as a bug.   
**What do I do if I find a bug/error?**   
In normal terms, if a bug or error is detected in a system, it needs to be communicated to the developer in order to get it fixed.   
Right from the first time any bug is detected till the point when the bug is fixed and closed, it is assigned various statuses which are New, Open, Postpone, Pending Retest, Retest, Pending Reject, Reject, Deferred, and Closed.   
(Please note that there are various ways to communicate the bug to the developer and track the bug status)   
  
**Statuses associated with a bug:**   
**New:** When a bug is found/revealed for the first time, the software tester communicates it to his/her team leader (Test Leader) in order to confirm if that is a valid bug. After getting confirmation from the Test Lead, the software tester logs the bug and the status of ‘New’ is assigned to the bug.   
**Assigned:** After the bug is reported as ‘New’, it comes to the Development Team. The development team verifies if the bug is valid. If the bug is valid, development leader assigns it to a developer to fix it and a status of ‘Assigned’ is assigned to it.   
**Open:** Once the developer starts working on the bug, he/she changes the status of the bug to ‘Open’ to indicate that he/she is working on it to find a solution.   
**Fixed:** Once the developer makes necessary changes in the code and verifies the code, he/she marks the bug as ‘Fixed’ and passes it over to the Development Lead in order to pass it to the Testing team.   
**Pending Retest:** After the bug is fixed, it is passed back to the testing team to get retested and the status of ‘Pending Retest’ is assigned to it.   
**Retest:** The testing team leader changes the status of the bug, which is previously marked with ‘Pending Retest’ to ‘Retest’ and assigns it to a tester for retesting.   
**Closed:** After the bug is assigned a status of ‘Retest’, it is again tested. If the problem is solved, the tester closes it and marks it with ‘Closed’ status.   
**Reopen:** If after retesting the software for the bug opened, if the system behaves in the same way or same bug arises once again, then the tester reopens the bug and again sends it back to the developer marking its status as ‘Reopen’.   
**Pending Reject:** If the developers think that a particular behavior of the system, which the tester reports as a bug has to be same and the bug is invalid, in that case, the bug is rejected and marked as ‘Pending Reject’.   
**Rejected:** If the Testing Leader finds that the system is working according to the specifications or the bug is invalid as per the explanation from the development, he/she rejects the bug and marks its status as ‘Rejected’.   
**Postponed:** Sometimes, testing of a particular bug has to be postponed for an indefinite period. This situation may occur because of many reasons, such as unavailability of Test data, unavailability of particular functionality etc. That time, the bug is marked with ‘Postponed’ status.   
**Deferred:** In some cases a particular bug stands no importance and is needed to be/can be avoided, that time it is marked with ‘Deferred’ status.

**SW Test methodology**

Two basic types of testing: manual testing, automated testing.

**Manual Testing:**

**Functionality Testing:** to determine all modules and its properties are working properly.

**Regression Testing:** after fixing bug(s) by developer, to make sure that fixation bug has not caused any new problems to any part of the software.

**Integration Testing:** to determine that every single piece of application such as back-end data, front-end operating system, hardware, software, networking connectivity and all subsystems are interacting to each other as per requirement.

**Transaction Testing:** to make sure that the inserted data into the front-end, has been transacted to the database properly. (SQL queries)

**System Testing:** we perform this testing after all the modules have been developed and connected to each other. In this point, we test the whole functionality to make sure all the systems and subsystems to the application is working properly.

**Black-box Testing:** testing without knowing any program designed and source code.

**White-box Testing:** testing with detailed knowledge of program design and source code.

**Unit Testing:** to verify the smallest pieces of codes to determine if the actual structure is correct and if the code defines operates correctly.

**Security Testing:** to determine how well the system protects against unauthorized internal or external access. (userid, password)

**Positive Testing:** testing with valid data.

**Negative Testing:** testing with invalid data.

**Alpha Testing:** when an application development is close to complete but still has to make some minor design change due to testing result that’s called alpha testing. (Usually done by end users or others, not by the developer or tester.)

**Beta Testing:** when development and testing are essentially completed and final bugs or problem need to be found before the final release that is Beta Testing.

**UAT** (User Acceptance Testing): is the final testing, based on specifications of the end user or customer before the software go for the production.

**Recovery Testing:** how well the system recovers from crashes and hardware failure or other major problems.

**Usability Testing:** how friendly or simple way the software can use by the end user. User Interview, surveys, video recording of user sessions can be used to perform this testing. (Usually done by end-users or others, not by the developers or testers.)

**User Interface Testing:** is to check a user’s interaction with the software.

**Compatibility Testing:** verifies that the SW application to be installed and executed in the production environment doesn’t prevent or preclude the execution of other computer programs that co-exist within the installed SW environment.

**Smoke Testing:** to determine whether it is possible to continue testing.

**Sanity Testing:** to determine whether it is reasonable to proceed with further testing.

**Automated Testing**:

**Performance Testing**: determines how fast a system performs under a particular load.

**Load Testing**:

**Volume Testing**: determine whether the program can handle the required volumes of data or large volume of data, requests, etc.

(or) determine how long application takes to load how large amount of data into the database.

**Stress Testing**: determine application’s breaking point (bottleneck).

**Soak Testing**: long hour testing, usually 24 hours, determines database leaking, system endurance or stability.

**Test plan**: is the document that describes the entire test process including objectives, scope, approach, and focus of a software testing effort.

Test plan outline:

01. Background: this item summarizes the functions of the application system and the tests to be performed.

02. Introduction

03. Assumption: indicates any anticipated assumptions which will be made while testing the application.

04. Test items: List each of the items (programs) to be tested.

05. Feature to be tested:

06. Feature not to be tested:

07. Approach: describes the data flow and test philosophy.

08. Item pass/fail criteria: itemized list of expected output.

09. Suspension/Resumption criteria: must the test run from start to completion?

under what circumstances it may be resumed in the middle?

establish check-points in long tests.

10. Test deliverables: what, besides software, will be delivered (Test plan, TCP, RTM, TAR, Test script, etc.)

11. Testing task: functional tasks (e.g., equipment set up)

administrative tasks

12. Environmental needs: security clearance

office space and equipment

hardware and software requirements

13. Responsibilities: Who does the tasks in section 10?

What does the user do?

14. Staffing and training

15. Schedule

16. Resources

17. Risk and Contingencies

18. Approval

**Test case**: is an individual test that corresponds of step by step description to test the specific scenario which in turn maps back to the assertion(s) and finally the specifications.

In order to write test case, we need: user requirement, function specification, test plan, application under test.

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

| Test Case no: |

| Version: |

| Tester: |

| Test Author: |

| Test date: |

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

| Step no | Description | Expected Result | Actual Result | Pass/fail criteria |

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

| | | | | |

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

**Traceability Matrix**: is a document which maps requirements with test cases. By preparing RTM, we can ensure that we have covered all functionalities in our test cases.

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

| Serial no | Requirement ID | Test case ID |

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

| | | |

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

**Test Analysis Report**: documents software testing unit/module, subsystem integration, system, user acceptance, and security - as defined in the test plan. The Test Analysis Report records results of the tests, presents the capabilities and deficiencies for review, and provides a means of assessing software progression to the next stage of development or testing. Results of each type of test are added to the software development document for the module or system being tested. Reports are created as required in the remaining phases.

The set of Test Analysis Reports provides a basis for assigning responsibility for deficiency correction and follow up, and for preparation of a statement of project completion.

**How many members are in your team?** 6

**How many scripts you created in QTP?** (multiply week with 10)

**How long takes to create a test plan?** 2 weeks

**How long takes to create Test case?** initially 3 weeks. After that review and approval.

**Automation Testing:** is a process where test cases run with test scripts using automation testing tools, which can be executed repeatedly and doesn’t require any manual intervention.

**Benefits of Automation Testing:**

Automation testing:

* Supports execution of repeated test cases
* Aids in testing a large test matrix
* Enables parallel execution
* Encourages unattended execution
* Improves accuracy thereby reducing human generated errors
* Saves time and money

**What percent automation and what percent manual test in your last project?**

30 percent automation and 70 percent manual. (But in my job 10% manual, 90% automation)

**Who gonna decide you gonna start testing manually or automation?**

Usually we decide while develop a test plan

**When do you need automation testing?**

Tests that need to run for every build

Tests that use multiple data values for same action

Identical tests that need to be executed for different browsers

**When do you not need automation testing?**

Test cases that only executed once

Test cases used for ad-hoc or random testing

Test cases that are infrequently selected for execution

**When to stop testing?**

Release or testing deadline

Test cases completed with certain percentages passed

Test budget depleted

Coverage of functionality or requirements reaches a specific point

The risk of the project is under acceptable limit

**If I give you thousand tests to execute in 2 days, what do you do?**

I'd try to finish all ad-hoc testing. It covers to least basic functionalities to verify that the system is working fine.

**Do you have any requirement that you don't have to test?**

Yes, I found some requirement hardware related.

**What is the major bug you found in your last project?**

Users input the accident report, but data were not stored in the database.

**Do you start testing in development period?** Yes

**Where do you log in bug?** QC/ALM

**What is Risk Analysis?**

**Risk Analysis** attempts to identify all the risks and then quantify the severity of the risk.

A **risk** is a potential for loss or damage to an organization from materialized threats.

A **threat** as we seen is a possible damage event.

Risk identification:

**Software risks**: most common risks associated with software development, and the platform you're working on.

**Business risks**: most common risks associated with business using the software.

**Testing risks**: most common risks associated with software testing for the platform you are working on, tools being used, and test methods being applied.

**Risk methods**: strategies and approaches for identifying risks or problems associated with implementing and operating IT, product and process; assessing their likelihood, and initiating strategies to test those risks.

**What are exit and entry criteria?**

**Entry criteria**: is a set of decision-making guidelines used to determining whether a system under test is ready to move into a particular phase of testing.

**Exit criteria**: is a set of decision-making guidelines used to determining whether a system under test is ready to exit a particular phase of testing.

**What is your strength? (or Why should I hire you?)**

I’m a hard working person.

I can take the challenge with job related any action.

I’m expert in both manual and automation testing.

Quick learner, quick starter, willing to learn.

Good Interpersonal and Communication Skills.

Ability to work either in a team or independently.

Ability to pay close attention in detail

I can quickly absorb new technologies.

I am an honest and friendly person.

Finally, your requirements match with my qualification. (for why should I hire you?)

**What is your weakness? ??**

I’m detailed oriented. A lot of people don’t like it.

I cannot stop until I finished. Most of people don’t wait when office hours over.

(or)

My weakest attribute is when I can’t solve out a problem. I can’t give it up. It keeps hovering on my head. Whatever I do, my conscious mind comes back to the previous problem and seeks for the solution. I think it is not always good for work. (I may make mistakes on serious task while I’m thinking on previous problem).

**What your immediate supervisor says about you?** I’m a valuable asset in their team.

**What** **is the negative thing your boss says about you???** He is a very nice guy and never say anything negative.

**What is the leadership quality?**

Lead at least three testers in a team. Distribute their jibs and make sure they are within timeframe.

**Day to day activities?**

Depends.

In the beginning: review the documents, create test plan and test cases.

In the middle: execute test, tracking bug

In the end: generating report.

**Are you in the market?** I’m at the last phase of my project.

**Why did you leave your recent job?** It is a contract position.

**When are you able to start?** In two weeks.

**Salary?**  $100K or $50 /hr

**Is salary negotiable?**

Depends. If this is a permanent position, what kind of benefit are you providing? $5000 less if benefit.

**Tax files?**

(never say that you’re a salary based employee. Say that you’re a contractor and take 1099 or Corp to Corp)

**W 2**: cutting tax.

**Corp to Corp**: you work for a company to another company. (H1, ..)

**1099**: for contract work, Companies pay you all amount without deducting tax. Your responsibilities to file tax.

(if they ask you to show: I cannot show, because they did not send me anything yet.)

**What is the quality of a good tester? (or what is the difference between a tester and a good tester?)**

A good tester should:

Be familiar with the software development process.

Be able to promote cooperation between analyst, developers, and QA tester.

Be able to promote teamwork to increase productivity.

Be able to maintain time management.

**What is the challenge of a good tester?**

Ensure that the application met all the user requirements.

**What is the release schedule? How often a company has a release schedule?**

A document that contains the dates when all releases will be rolled out into the live environment.

Usually 3 months.