Black-box testing

Black box diagram

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it.[2] Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing and specification-based testing.

Specification-based testing aims to test the functionality of software according to the applicable requirements.[3] This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional.

Specification-based testing may be necessary to assure correct functionality, but it is insufficient to guard against complex or high-risk situations.[4]

One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality. On the other hand, black-box testing has been said to be "like a walk in a dark labyrinth without a flashlight."[5] Because they do not examine the source code, there are situations when a tester writes many test cases to check something that could have been tested by only one test case, or leaves some parts of the program untested.

This method of test can be applied to all levels of software testing: unit, integration, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

Boundary Value Analysis

Boundary value analysis, BVA, tests the behavior of a program at the boundaries. When checking a range of values, after selecting the set of data that lie in the valid partitions, next is to check how the program behaves at the boundary values of the valid partitions. Boundary value analysis is most common when checking a range of numbers.

For each range, there are two boundaries, the lower boundary (start of the range) and the upper boundary (end of the range) and the boundaries are the beginning and end of each valid partition. We should design test cases which exercises the program functionality at the boundaries, and with values just inside and just outside the boundaries.

Here, the assumption is that if the program works correctly for these extreme cases, then it will work correctly for all values in between the valid partition. Testing has shown that defects that arise when checking a range of values the most defects are near or at the boundaries.

Boundary Value Analysis Example

A program which accepts an integer in the range **-100** to **+100**, there would be three sets of valid equivalent partitions: these are -10 to -1, the negative range 0, Zero and 1 to 10, the positive range.

For each range, there are minimum and maximum values at each boundary. For the negative range the lower boundary is -10 and the upper boundary is -1. At each boundary, three conditions should be checked.

-101, **-100**, -99 … … … … … … -2, **-1**,

0, -1, **0**, +1

0, **1**, 2 … … … … … … 99, **100**, 101

You might have noticed that by selecting values at the boundaries for each partition there are some values that overlap. That is they would appear in our test conditions when we check the

boundaries. We should of course make redundant those values that overlap to eliminate unnecessary test cases.

Another worthy note to consider is that because the range goes from -100 to +100, then effectively, the "boundary values" -2, -1 and -99 are considered as equivalent. That is, the behavior of the system is the same (or should be the same) when testing with values -99, -2 and -1.

However, because -99 is close to the upper limit boundary, and is most likely to reveal defects, we keep the value -99 in our test case and scrap conditions for -2 and -1. The same applies to the positive range boundaries, that is, +1, +2 and +99 are expected to give the same behavior. However, because 99 is close to the upper range boundary, we keep 99 in our list of data to test and get rid of values 1 and 2.

Therefore, our list of data to check at the boundaries become

-101, **-100**, -99 … … … … … … 0 … … … … … … 99, **100**, 101

If our range of data was from 0 to +10 then the boundary value analysis would give us the following values to test:

-1, **0**, 1 … … … … … …9, **10**, 11

Here, 1 and 9 are both part of the test condition because each are at the boundary of the whole range and can reveal defects.

It is important to note that that boundary values analysis can be applied to float numbers as well as integers. The only difference is that when analyzing boundaries for float numbers, we should test to the closest decimal point.

Example, if we have a range from 5.5 to 9.9, then the set of data at boundaries becomes

5.4, **5.5**, 5.6 … … … … … … 9.8, **9.9**, 10.0

Like Equivalence Partitioning Test Technique, Boundary Value Analysis is a common black box testing technique and is one which must be applied when checking a range of values. Together

with equivalence partitioning and negative testing, it can be a very powerful black box testing technique to find defects when testing a range of values.

INTEGRATION TESTING

Integration Test Case

Integration Test Case differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**. Here priority is to be given for the **integrating links**rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page.

Similarly Mail Box: Check its integration to the Delete Mails Module.

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mailbox select the email and click a delete button	Selected email should appear the Deleted/Trash folder

Approaches/Methodologies/Strategies of Integration Testing:

Software Engineering defines variety of strategies to execute Integration testing, viz.

- Big Bang Approach :
- Incremental Approach: which is further divided into the following

- Top Down Approach
- Bottom Up Approach
- Sandwich Approach - Combination of Top Down and Bottom Up

Below are the different strategies, the way they are executed and their limitations as well advantages.

Big Bang Approach:

Here all component are integrated together at **once** and then tested.

Advantages:

- Convenient for small systems.

Disadvantages:

- Fault Localization is difficult.
- Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.
- Since the Integration testing can commence only after "all" the modules are designed, the testing team will have less time for execution in the testing phase.
- Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

Incremental Approach

In this approach, testing is done by joining two or more modules that are ***logically related***. Then the other related modules are added and tested for the proper functioning. The process continues until all of the modules are joined and tested successfully.

Incremental Approach, in turn, is carried out by two different Methods:

- Bottom Up

- Top Down

What is Stub and Driver?

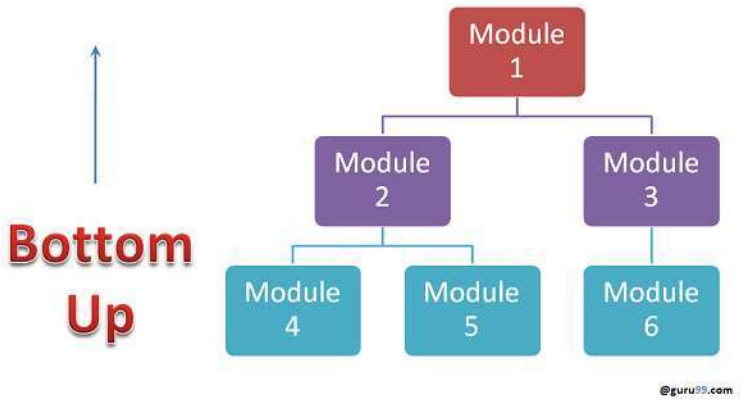
Incremental Approach is carried out by using dummy programs called **Stubs and Drivers**. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.

Stub: Is called by the Module under Test.

Bottom-up Integration

In the bottom-up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing

Diagrammatic Representation:



Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.

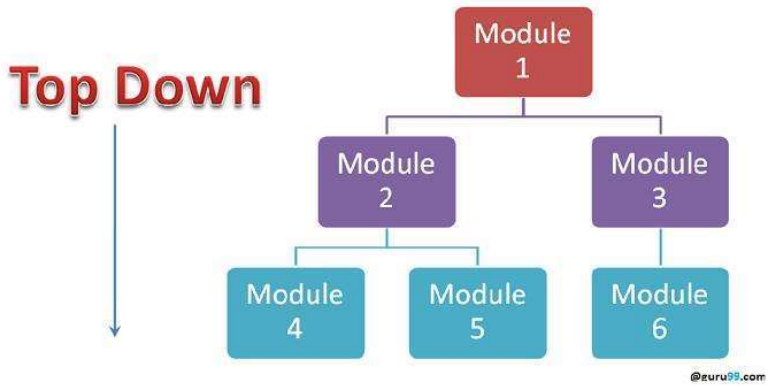
- An early prototype is not possible

Top-down Integration:

In Top to down approach, testing takes place from top to down following the control flow of the software system.

Takes help of stubs for testing.

Diagrammatic Representation:



Advantages:

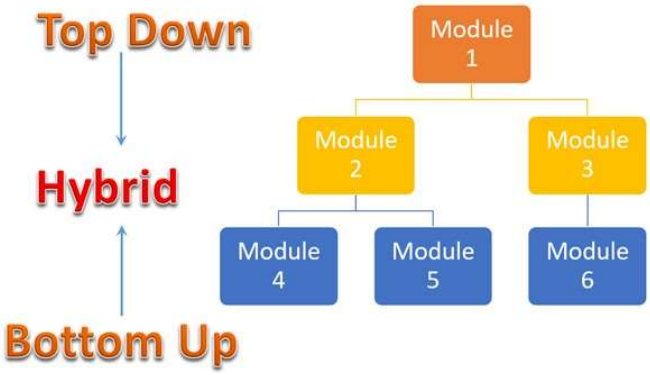
- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

Hybrid/ Sandwich Integration

In the sandwich/hybrid strategy is a combination of Top Down and Bottom up approaches. Here, top modules are tested with lower modules at the same time lower modules are integrated with top modules and tested. This strategy makes use of stubs as well as drivers.



Branch Coverage

In the branch coverage, every outcome from a code module is tested. For example, if the outcomes are binary, you need to test both True and False outcomes.

It helps you to ensure that every possible branch from each decision condition is executed at least a single time.

By using Branch coverage method, you can also measure the fraction of independent code segments. It also helps you to find out which is sections of code don't have any branches.

The formula to calculate Branch Coverage:

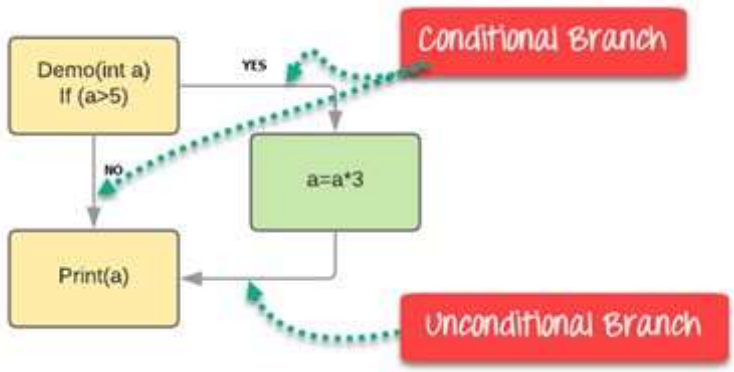
$$Branch\ Coverage = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches}$$

Example of Branch Coverage

To learn branch coverage, let's consider the same example used earlier

Consider the following code

```
Demo(int a) {
    If (a> 5)
            a=a*3
    Print (a)
   }
```



Branch Coverage will consider unconditional branch as well

Test Case	Value of A	Output	Decision Coverage	Branch Coverage
1	2	2	50%	**33%**
2	6	18	50%	**67%**

Advantages of Branch coverage:

Branch coverage Testing offers the following advantages:

- Allows you to validate-all the branches in the code
- Helps you to ensure that no branched lead to any abnormality of the program's operation

- Branch coverage method removes issues which happen because of statement coverage testing
- Allows you to find those areas which are not tested by other testing methods
- It allows you to find a quantitative measure of code coverage
- Branch coverage ignores branches inside the Boolean expressions

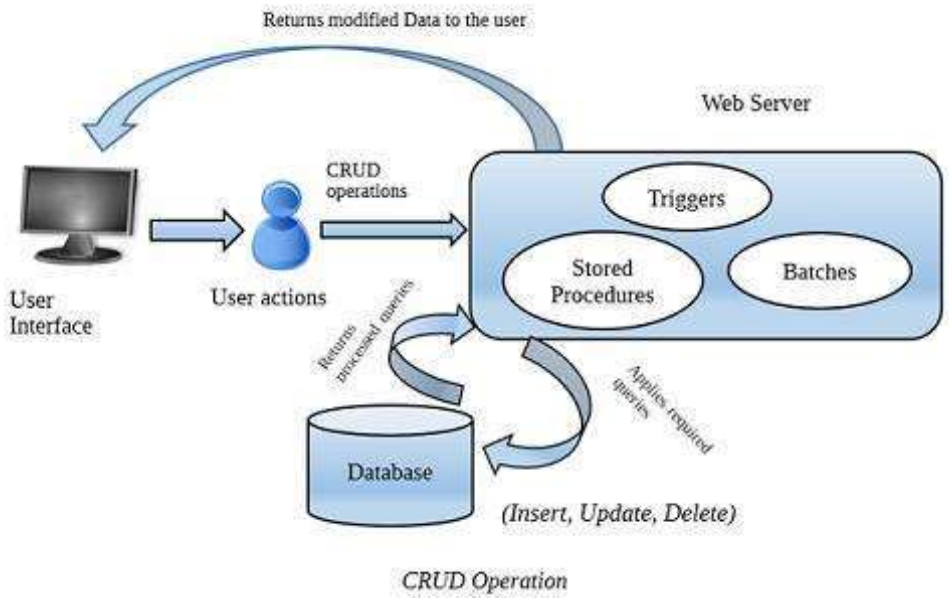
CRUD Testing

CRUD testing is a black box testing. CRUD is an acronym for Create, Read, Update, Delete. CRUD testing is another term for database testing. Database forms an inevitable part of a software. Database forms the backbone of any application- web or desktop, data is stored somewhere. A user feeds in the information and the data gets stored in some form. An organisation may use any of the available databases available in the market - MS Access, Oracle, MySQL, DB2 etc. It depends on the needs and requirements of an organisation as to which kind of database has to be integrated with their application. Every database has its own set of features and associated cost with it.

CRUD operations from the perspective of an end-user can thought of like this -

- Create - user 'Saving' any new transaction.
- Read/Retrieve - User 'Searching' or 'Viewing' any transaction.
- Update - when a user wants to 'Edit' or 'Modify' an existing data.
- Delete - when user wants to 'Remove' any data from the system.

Let's take up a very common scenario of any of the social networking application. When a user is new, he registers on the application and the data gets stored in the application server. If in case someone wants to update their information or add new information, they have the required option. Examples can be skype, facebook etc.



Why Database Testing :

- **To ensure Data Mapping -**The system must be able to track the records correctly. That means, respective values are updated in the tables. For instance person A wants to change his name and phone number that should reflect in appropriate record fields.

- **Maintain ACID Properties -** ACID properties lie at the core of a database structure. Atomicity, consistency, isolation, durability - these four components are essential and forms a complete package for a robust database. *Atomicity* means that changes must reflect all across uniformly or none should be affected. *Consistency* implies that a data must only be updated when it follows some predefined rules. *Isolation* determines how users are able to view the transactions, that is, isolation rules define how a change made to a certain thing is reflected all across. *Durability* states the transactions once committed will be saved permanently . For instance, if a booking has been done for a flight, the booking for a seat will remain even if there is some mishap in the system.

- **Ensure Data Integrity -**Data integrity means when a data is modified, the latest state of data must reflect in every system. A system therefore must show the updated and most recent values to all the users accessing it from anywhere.

- **Ensure Accuracy of implemented business Rules -**In the current scenario database follow the concept of RDBMs which is relational database management system. Referential integrity refers to the relation that exists among the tables. Primary key contraints, foreign keys, triggers, stored procedures etc form a set of business rules. Therefore developers integrate these business rules into their code when implementing database business logic. These business rules are essential in maintaining the ACID properties.

Grey Box Testing?

Grey Box testing is testing technique performed with limited information about the internal functionality of the system. Grey Box testers have access to the detailed design documents along with information about requirements.

Grey Box tests are generated based on the state-based models, UML Diagrams or architecture diagrams of the target system.

Gray-box testing Techniques:

- Regression testing
- Pattern Testing
- Orthogonal array testing
- Matrix testing

Benefits:

- Grey-box testing provides combined benefits of both white-box and black-box testing
- It is based on functional specification, UML Diagrams, Database Diagrams or architectural view
- Grey-box tester handles can design complex test scenario more intelligently

- The added advantage of grey-box testing is that it maintains the boundary between independent testers and developers

Drawbacks:

- In grey-box testing, complete white box testing cannot be done due to inaccessible source code/binaries.
- It is difficult to associate defects when we perform Grey-box testing for a distributed system.

Best Suited Applications:

Grey-box testing is a perfect fit for Web-based applications.

Grey-box testing is also a best approach for functional or domain testing.

JAD

JAD (Joint Application Development) is a software development approach which engages the client and/or the end users for designing and developing the system. This model was designed and put forward by Dr. Chuck Morris and Dr. Tony Crawford of IBM, who propose this model in the late 1970s. As compared to other primitive SDLC model, Joint Application Development model leads to faster progression of the system development which has better client approval.

This model furthermore, is vast when it comes to agile delivery wherein the software products need to be developed as well as shipped in short iterations depending on agreements among the industrial as well as industry stakeholders which are termed as Minimum Viable Product (MVP).

Phases of jad

Since you have become familiar with the JAD concept, it is time to know about its phases and how the model's design and development approach works:

1. Define Specific Objectives: The facilitator, in partnership with stakeholders, set all the objectives as well as a list of items which is then distributed to other developers and participants to understand and review. This objective contains elements like the scope of this projected system, its potential outcome, technical specification required, etc.

2. Session Preparation: The facilitator is solely responsible for this preparation where all relevant data is collected and sent to other members before time. For better insight, research carried out to know about the system requirement better and gather all the necessary information for development.

3. Session Conduct: Here the facilitator is accountable to identify those issues which have to be working out for making the system error-free. Here the facilitator will serve as a participant but will not have a say regarding any information.

4. Documentation: After the product is developed, the records and published documents are put forward into the meeting so that the stakeholders and consumers can approve it through the meeting.

Benefits of jad

- Improved Delivery Time: The time required for developing a product using JAD model is lesser and efficient than that of other traditional models.
- Cost Reduction: Efficiently analyzing the requirements and facts with business executives and stakeholders will make less effort to develop the system and hence less cost will be required for the entire development process.
- Better Understanding: Since the entire requirement is analyzed by business executives, followed by a cautious choice of developers and team member who can professionally interact with each other better usually helps in understanding the product development better.
- Improved Quality: Since all the key decision makers and stakeholders of the project are involved in the development of the project so there is the least chance of error and hence the product quality becomes better and more accurate.

Pareto Analysis?

- A statistical technique for making decisions which is used for selecting a limited number of tasks which produce significant overall effect. Pareto Analysis uses the 'Pareto Principle' – an

idea by which 80% of doing the entire job is generated by doing 20% of the work.

- When many possible courses of actions are completing the attention, the technique 'Pareto Analysis' is useful. In essence, the delivered benefit by each action is estimated by problem-solver, and selects the number of most effective actions which delivers the total benefit.

- Pareto Analysis is a mechanism to find changes that will give the most beneficial results. It states that only few factors are responsible for producing most problems. It is used so that the team can concentrate on those changes.

Prototype Testing?

Prototype Testing is conducted with the intent of finding defects before the website goes live. Online Prototype Testing allows seamlessly to collect quantitative, qualitative, and behavioural data while evaluating the user experience.

Characteristics of Prototype Testing:

- To evaluate new designs prior to the actual go live to ensure that the designs are clear, easy to use and meet users requirements.
- Is best when iterative testing is built into the development process, so that changes can be easily made often to ensure that major issues do not arise well before going live.
- Provides confirmation about the new design direction, branding and messaging are going in the right direction.

Random Testing?

Random Testing, also known as monkey testing, is a form of functional black box testing that is performed when there is not enough time to write and execute the tests.

Random Testing Characteristics:

- Random testing is performed where the defects are NOT identified in regular intervals.
- Random input is used to test the system's reliability and performance.

- Saves time and effort than actual test efforts.
- Other Testing methods Cannot be used to.

Random Testing Steps:

- Random Inputs are identified to be evaluated against the system.
- Test Inputs are selected independently from test domain.
- Tests are Executed using those random inputs.
- Record the results and compare against the expected outcomes.
- Reproduce/Replicate the issue and raise defects, fix and retest.

There are 2: Smart and Dump

Smart Monkeys – A smart monkey is identified by the below characteristics:-

- Have a brief idea about the application
- They know where the pages of application will redirect to.
- They know that the inputs they are providing are valid or invalid.
- They work or focus to break the application.
- In case they find an error, they are smart enough to file a bug.
- They are aware of the menus and the buttons.
- Good to do stress and load testing.

Dumb Monkey – A dumb monkey is identified by the below characteristics:

- They have no idea about the application.
- They don't know that the inputs they are providing are valid or invalid.
- They test the application randomly and are not aware of any starting point of the application or the end to end flow.
- Though they are not aware of application, but they too can identify bugs like environmental failure or hardware failure.
- They don't have much idea about the UI and functionality

Advantages of Monkey Testing:

- Can identify some out of the box errors.
- Easy to set up and execute

- Can be done by “not so skilled” resources.
- A good technique to test the reliability of the software
- Can identify bugs which may have higher impact.
- Not costly

Disadvantages of Monkey test:

- This can go on for days till a bug is not discovered.
- Number of bugs are less
- Reproducing the bugs (if occurs) becomes a challenge.
- Apart of some bugs, there can be some “Not Expected” output of a test scenario, analysis of which becomes a difficult and time consuming.

RISK BASED TESTING

Risk Based Testing (RBT) is a testing process with unique features. It is basically for those project and application that is based on risk. Using risk, Risk based testing prioritize and emphasize the suitable tests at the time of test execution. In other word, Risk is the chance of event of an unwanted outcome. This unwanted outcome is also related with an impact. Some time it is difficult to test all functionality of the application or it is might not possible. Use Risk based testing in that case; it tests the functionality which has the highest impact and probability of failure.

It’s better to start risk based testing with product risk analysis. There are numerous methods used for this are,

- Clear understanding of software requirements specification, design documents and other documents.
- Brainstorming with the project stakeholders.

  Risk-based testing is the process to understand testing efforts in a way that reduces the remaining level of product risk when the system is developed,
- Risk-based testing applied to the project at very initial level, identifies risks of the project that expose the quality of the project, this knowledge guides to testing planning, specification, preparation and execution.

- Risk-based testing includes both mitigation (testing to give chances to decrease the likelihood of faults, specially high-impact faults) and contingency (testing to know work-around to create the defects that do get past us less painful).
- Risk-based testing also includes measurement process that recognizes how well we are working at finding and removing faults in key areas.
- Risk-based testing also uses risk analysis to recognize proactive chances to take out or avoid defects through non-testing activities and to help us select which test activities to perform.

Major processes to execute the Risk-based testing are described below:

- Process 1 – Describe all requirements in terms of Risk involved in the project
- Process 2 – In terms of risk assessment, prioritize the requirements
- Process 3 – Plan and define tests according to requirement prioritization
- Process 4 – Execute test according to prioritization and acceptance criteria.

Process1– Projects associates various risks that are identified by the stakeholders of the project. The stake holders are basically a mixture of business and technical team. Stake holders involve various people from various departments for example, the client, customers, business experts, technical experts, project manager, project Leader, users, developers and infrastructure representative.

Process 2 – Once all the possible risks and their impacts are analyzed, the project manager has to get the requirements prioritized. Priority of the requirements should be agreed upon and should be updated in functional requirement document; the same should also be conveyed to the development and the test team.

Process 3 – After getting the Requirement with the priority tagged, we can start the test activities with keeping the priority of the requirements in mind.

Process 4 – If any of the identified risk realizes by the time of "Test execution Schedule", then there is quite good chances of schedule slippage from development side. In this case, the final deadline given to the customer can't be changed. In this situation, again, Test manager has to apply the Pareto principle and finalizes the scope of testing in the reduced timeline that will ensure least risk and highest quality.

Advantages

- Improved quality – All of the critical functions of the application are tested. Real time clear understanding of project risk.
- Give more focus on risks of the business project instead of the functionality of the information system.
- Provides a negotiating instrument to client and test manager similar when existing means are limited.
- Associate the product risk to the requirement identifies gaps.
- During testing, test reporting always takes place in a language (risks) that all stake-holder understands.
- Testing always concentrate on the most important matters first with optimal test delivery, in case of – limited time, money and qualified resources. With the time and resources we have, we just can able to complete 100% testing, so we need to determine a better way to

accelerate our testing effort with still managing the risk of the application under test. Efforts are not wasted on non-critical or low risk functions.

- Improve customer satisfaction – Due to customer involvement and good reporting and progress tracking.

Some Helpful Test Techniques to proceed Risk-based testing

- Path Flow testing
- Some amount of exploratory / Experience based testing
- Boundary Value analysis
- Equivalence partitioning
- Decision tables

REGRESSION TESTING?

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

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

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

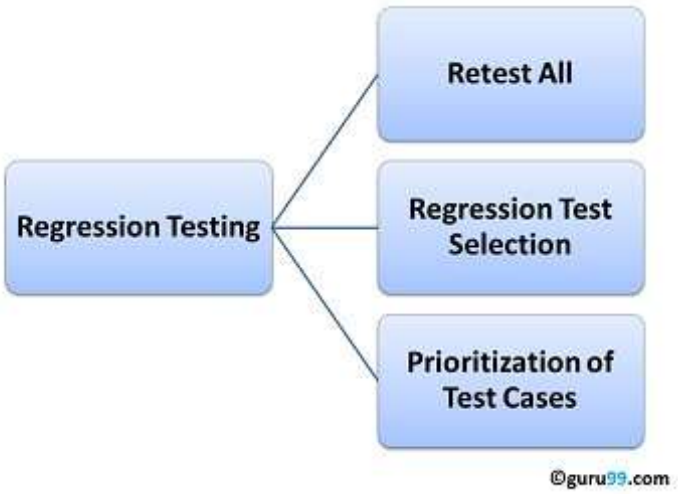
Need of Regression Testing

Regression Testing is required when there is a

- Change in requirements and code is modified according to the requirement
- New feature is added to the software
- Defect fixing
- Performance issue fix

How to do Regression Testing

Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of existing features. These modifications may cause the system to work incorrectly. Therefore, Regression Testing becomes necessary. Regression Testing can be carried out using the following techniques:



Retest All

- This is one of the methods for Regression Testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.

Regression Test Selection

- Instead of re-executing the entire test suite, it is better to select part of the test suite to be run
- Test cases selected can be categorized as 1) Reusable Test Cases 2) Obsolete Test Cases.
- Re-usable Test cases can be used in succeeding regression cycles.
- Obsolete Test Cases can't be used in succeeding cycles.

Prioritization of Test Cases

- Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite.

Selecting test cases for regression testing

It was found from industry data that a good number of the defects reported by customers were due to last minute bug fixes creating side effects and hence selecting the Test Case for regression testing is an art and not that easy. Effective Regression Tests can be done by selecting the following test cases -

- Test cases which have frequent defects
- Functionalities which are more visible to the users
- Test cases which verify core features of the product
- Test cases of Functionalities which has undergone more and recent changes
- All Integration Test Cases
- All Complex Test Cases
- Boundary value test cases
- A sample of Successful test cases
- A sample of Failure test cases

Regression Testing Tools

If your software undergoes frequent changes, regression testing costs will escalate.

In such cases, Manual execution of test cases increases test execution time as well as costs.

Automation of regression test cases is the smart choice in such cases.

The extent of automation depends on the number of test cases that remain re-usable for successive regression cycles.

Following are the most important tools used for both functional and regression testing in software engineering.

Ranorex Studio — all-in-one regression test automation for desktop, web, and mobile apps with built-in Selenium WebDriver. Includes a full IDE plus tools for codeless automation.

Testim - is an AI based test automation platform for web and mobile apps. It uses dynamic locators that learn with every execution and adapts to code changes, minimizing maintenance time spent fixing flaky tests

Selenium: This is an open source tool used for automating web applications. Selenium can be used for browser-based regression testing.

Quick Test Professional (QTP): HP Quick Test Professional is automated software designed to automate functional and regression test cases. It uses VBScript language for automation. It is a Data-driven, Keyword based tool.

Rational Functional Tester (RFT): IBM's rational functional tester is a Java tool used to automate the test cases of software applications. This is primarily used for automating regression test cases and it also integrates with Rational Test Manager.

Regression Testing and Configuration Management

Configuration Management during Regression Testing becomes imperative in Agile Environments where a code is being continuously modified. To ensure effective regression tests, observe the following :

- Code being regression tested should be under a configuration management tool
- No changes must be allowed to code, during the regression test phase. Regression test code must be kept immune to developer changes.
- The database used for regression testing must be isolated. No database changes must be allowed

Difference between Re-Testing and Regression Testing:

Retesting means testing the functionality or bug again to ensure the code is fixed. If it is not fixed,Defect needs to be re-opened. If fixed, Defect is closed.

Regression testing means testing your software application when it undergoes a code change to ensure that the new code has not affected other parts of the software.

STRUCTURED WALKTHROUGH?

A structured walkthrough, a static testing technique performed in an organized manner between a group of peers to review and discuss the technical aspects of software development process. The main objective in a structured walkthrough is to find defects inorder to improve the quality of the product.

Structured walkthroughs are usually NOT used for technical discussions or to discuss the solutions for the issues found. As explained, the aim is to detect error and not to correct errors. When the walkthrough is finished, the author of the output is responsible for fixing the issues.

Benefits:

- Saves time and money as defects are found and rectified very early in the lifecycle.
- This provides value-added comments from reviewers with different technical backgrounds and experience.
- It notifies the project management team about the progress of the development process.
- It creates awareness about different development or maintenance methodologies which can provide a professional growth to participants.

Structured Walkthrough Participants:

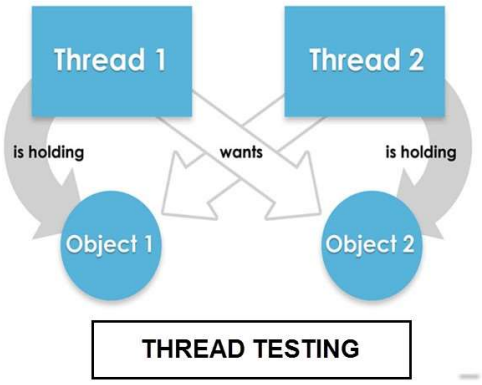
- **Author -** The Author of the document under review.
- **Presenter -** The presenter usually develops the agenda for the walkthrough and presents the output being reviewed.
- **Moderator -** The moderator facilitates the walkthrough session, ensures the walkthrough agenda is followed, and encourages all the reviewers to participate.
- **Reviewers -** The reviewers evaluate the document under test to determine if it is technically accurate.

- **Scribe -** The scribe is the recorder of the structured walkthrough outcomes who records the issues identified and any other technical comments, suggestions, and unresolved questions.

THREAD TESTING?

Thread testing is defined as a software testing type, which verify the key functional capabilities of a specific task(thread). It is usually conducted at the early stage of Integration Testing phase.

Thread based testing is one of the incremental strategies adopted during System Integration Testing. That's why, thread test should probably more properly be called a "**thread interaction test."**



Types of Thread Testing

Thread based testing are classified into two categories

- **Single thread testing**: A single thread testing involves one application transaction at a time
- **Multi-thread testing**: A multi-thread testing involves several concurrently active transaction at a time

How to do Thread Testing

The thread process focuses on the integration activities rather than the full development lifecycle. For Example,

- Thread-based testing is a generalized form of session-based testing, in that sessions are a form of thread, but a thread is not necessarily a session.
- For thread testing, the thread or program (small functionality) are integrated and tested incrementally as a subsystem, and then executed for a whole system.
- At the lowest level, it provided integrators with better knowledge of the scope of what to test
- Rather than testing software components directly, it required integrators to concentrate on testing logical execution paths in the context of the entire system.

Tips for Multithread Testing

- Test your multithreaded program by executing it repeatedly with a different mix of applications running
- Test your multithreaded program by having multiple instances of the program active at the same time
- Execute your multithreaded program on different hardware models with varying stress levels and workloads
- Code inspection
- Only collect errors and failures that occurred in threads other than the main one

Disadvantages of Thread Testing

- For multithreading testing, the biggest challenge is that you should be able to program reproducible test for unit test
- Writing unit tests for multithreaded code is a challenging task
- Testing criteria for multi-thread testing are different than single thread testing. For multithread testing various factors like memory size, storage capacity, timing problems, etc. varies when called on different hardware.

What is Performance Testing?

- Performance Testing is defined as a type of software testing to ensure software applications will perform well under their expected workload.

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

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

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

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

Types of Performance Testing

- **Load testing -** checks the application's ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.
- **Stress testing -** involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application.
- **Endurance testing -** is done to make sure the software can handle the expected load over a long period of time.
- **Spike testing -** tests the software's reaction to sudden large spikes in the load generated by users.

- **Volume testing -** Under Volume Testing large no. of. Data is populated in a database and the overall software system's behavior is monitored. The objective is to check software application's performance under varying database volumes.
- **Scalability testing -** The objective of scalability testing is to determine the software application's effectiveness in "scaling up" to support an increase in user load. It helps plan capacity addition to your software system.

Common Performance Problems

Most performance problems revolve around speed, response time, load time and poor scalability. Speed is often one of the most important attributes of an application. A slow running application will lose potential users. Performance testing is done to make sure an app runs fast enough to keep a user's attention and interest. Take a look at the following list of common performance problems and notice how speed is a common factor in many of them:

- **Long Load time -** Load time is normally the initial time it takes an application to start. This should generally be kept to a minimum. While some applications are impossible to make load in under a minute, Load time should be kept under a few seconds if possible.
- **Poor response time -** Response time is the time it takes from when a user inputs data into the application until the application outputs a response to that input. Generally, this should be very quick. Again if a user has to wait too long, they lose interest.
- **Poor scalability -** A software product suffers from poor scalability when it cannot handle the expected number of users or when it does not accommodate a wide enough range of users. Load Testing should be done to be certain the application can handle the anticipated number of users.
- **Bottlenecking -** Bottlenecks are obstructions in a system which degrade overall system performance. Bottlenecking is when either coding errors or hardware issues cause a decrease of throughput under certain loads. Bottlenecking is often caused by one faulty section of code. The key to fixing a bottlenecking issue is to find the section of code that is causing the slowdown and try to fix it there. Bottlenecking is generally fixed by either fixing poor running processes or adding additional Hardware. Some **common performance bottlenecks** are

- CPU utilization
- Memory utilization
- Network utilization
- Operating System limitations
- Disk usage

Performance Testing Process

The methodology adopted for performance testing can vary widely but the objective for performance tests remain the same. It can help demonstrate that your software system meets certain pre-defined performance criteria. Or it can help compare the performance of two software systems. It can also help identify parts of your software system which degrade its performance.

Below is a generic process on how to perform performance testing

1. **Identify your testing environment -** Know your physical test environment, production environment and what testing tools are available. Understand details of the hardware, software and network configurations used during testing before you begin the testing process. It will help testers create more efficient tests. It will also help identify possible challenges that testers may encounter during the performance testing procedures.
2. **Identify the performance acceptance criteria -** This includes goals and constraints for throughput, response times and resource allocation. It is also necessary to identify project success criteria outside of these goals and constraints. Testers should be empowered to set performance criteria and goals because often the project specifications will not include a wide enough variety of performance benchmarks. Sometimes there may be none at all. When possible finding a similar application to compare to is a good way to set performance goals.

3. **Plan & design performance tests -** Determine how usage is likely to vary amongst end users and identify key scenarios to test for all possible use cases. It is necessary to simulate a variety of end users, plan performance test data and outline what metrics will be gathered.
4. **Configuring the test environment -** Prepare the testing environment before execution. Also, arrange tools and other resources.
5. **Implement test design -** Create the performance tests according to your test design.
6. **Run the tests -** Execute and monitor the tests.
7. **Analyze, tune and retest** - Consolidate, analyze and share test results. Then fine tune and test again to see if there is an improvement or decrease in performance. Since improvements generally grow smaller with each retest, stop when bottlenecking is caused by the CPU. Then you may have the consider option of increasing CPU power.

Performance Test Tools

There are a wide variety of performance testing tools available in the market. The tool you choose for testing will depend on many factors such as types of the protocol supported, license cost, hardware requirements, platform support etc. Below is a list of popularly used testing tools.

- LoadNinja – is revolutionizing the way we load test. This cloud-based load testing tool empowers teams to record & instantly playback comprehensive load tests, without complex dynamic correlation & run these load tests in real browsers at scale. Teams are able to increase test coverage. & cut load testing time by over 60%.
- NeoLoad **-** is the performance testing platform designed for DevOps that seamlessly integrates into your existing Continuous Delivery pipeline. With NeoLoad, teams test 10x faster than with traditional tools to meet the new level of requirements across the full Agile software development lifecycle - from component to full system-wide load tests.
- HP LoadRunner **-** is the most popular performance testing tools on the market today. This tool is capable of simulating hundreds of thousands of users, putting applications under real-life loads to determine their behavior under expected loads. Loadrunner features a virtual user generator which simulates the actions of live human users.
- Jmeter **-** one of the leading tools used for load testing of web and application servers.