# Manual Testing Notes

# Introduction to Software Testing & its Importance

* **Software testing is a process used to identify the correctness, completeness, and quality of developed computer software.** It includes a set of activities conducted with the intent of finding errors in software so that it could be corrected before the product is released to the end users.
* In simple words, **software testing is an activity to check whether the actual results match the expected results and to ensure that the software system is defect free**.
* Why is testing is important?
* This is China Airlines Airbus A300 crashing due to a software bug on April 26, 1994, killing 264 innocent lives
* Software bugs can potentially cause monetary and human loss, history is full of such examples
* In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others
* In April of 1999, a software bug caused the failure of a $1.2 billion military satellite launch, the costliest accident in history
* In may of 1996, a software bug caused the bank accounts of 823 customers of a major U.S. bank to be credited with 920 million US dollars
* As you see, **testing is important because software bugs could be expensive or even dangerous**
* As Paul Elrich puts it - "To err is human, but to really foul things up you need a computer."

# 2. Learn Fundamental Principles of Software Testing

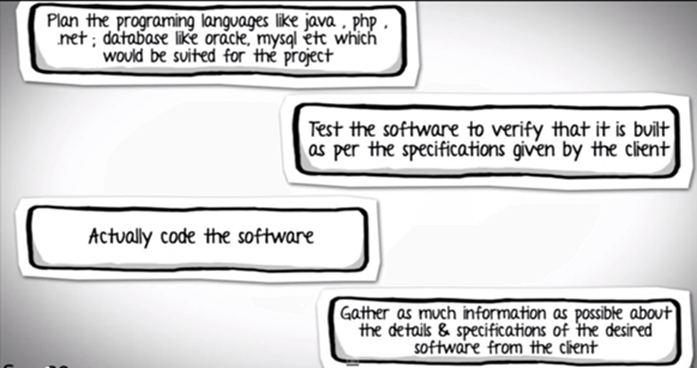
**.** Consider a scenario where you are moving a file from folder A to Folder B. Think of all the possible ways you can test this.

* Apart from the usual scenarios, you can also test the following conditions
* Trying to move the file when it is open
* You do not have the [security](http://www.guru99.com/ethical-hacking-tutorials.html) rights to paste the file in Folder B
* Folder B is on a shared drive and storage capacity is full.
* Folder B already has a file with the same name, infact the list is endless
* Or suppose you have 15 input fields to test, each having 5 possible values, the number of combinations to be tested would be 5^15
* **If you were to test the entire possible combinations project EXECUTION TIME & COSTS will rise exponentially.**
* Hence, one of the testing principles states that **EXHAUSTIVE testing is not possible**. **Instead, we need an optimal amount of testing based on the risk assessment of the application**.
* And the million dollar question is, how do you determine this risk?
* To answer this let's do an exercise
* In your opinion, Which operations are most likely to cause your Operating system to fail?
* I am sure most of you would have guessed, Opening 10 different application all at the same time.
* So if you were testing this Operating system you would realize that defects are likely to be found in  multi-tasking and needs to be tested thoroughly which brings us to our next principle **Defect Clustering which states that a small number of modules contain most of the defects detected.**
* By experience, you can identify such risky modules. But this approach has its own problems
* **If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs**
* This is the another principle of testing called **"Pesticide Paradox"**
* **To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.**
* But even after all this sweat & hard work in testing, you can never claim you product is bug-free. To drive home this point, let's see this video of public launch of Windows 98
* You think a company like MICROSOFT would not have tested their O.S thoroughly & would risk their reputation just to see their O.S crashing during its public launch!
* Hence, testing principle states that - **Testing shows presence of defects i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness**.
* But what if, you work extra hard, taking all precautions & make your software product 99% bug-free. And the software does not meet the needs & requirements of the clients.
* This leads us to our next principle, which states that-
* **Absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system build is unusable and does not fulfill the users needs & requirements**
* To fix this problem, the next principle of testing states that
* **Early Testing - Testing should start as early as possible in the Software Development Life Cycle**. So that any defects in the requirements or design phase are captured as well. More on this principle in a later training tutorial.
* And the last principle of testing states that the **Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application.**

|  |  |
| --- | --- |
| Principle 1 | Testing shows presence of defects |
| Principle 2 | Exhaustive testing is impossible |
| Principle 3 | Early Testing |
| Principle 4 | Defect Clustering |
| Principle 5 | Pesticide Paradox |
| Principle 6 | Testing is context dependent |
| Principle 7 | Absence of errors - fallacy |

# 3. Guide to SDLC , STLC & V-Model

Suppose, you are assigned a task, to develop a custom software for a client. Now, irrespective of your technical background, try and make an educated guess about the sequence of steps you will follow, to achieve the task.

[](http://cdn.guru99.com/images/6-2015/052715_0904_GuidetoSDLC1.png)

|  |  |
| --- | --- |
| **Different phases of Software Development Cycle** | **Activities performed in each stage** |
| **Requirement Gathering stage** | * Gather as much information as possible about the details & specifications of the desired software from the client. This is nothing but the Requirements gathering stage. |
| **Design Stage** | * Plan the programming language like [Java](http://www.guru99.com/java-tutorial.html), [PHP](http://www.guru99.com/php-tutorials.html), .net; database like Oracle, MySQL, etc. Which would be suited for the project, also some high-level functions & architecture. |
| **Built Stage** | * After design stage, it is built stage, that is nothing but actually code the software |
| **Test Stage** | * Next, you test the software to verify that it is built as per the specifications given by the client. |
| **Deployment stage** | * Deploy the application in the respective environment |
| **Maintenance stage** | * Once your system is ready to use, you may require to change the code later on as per customer request |

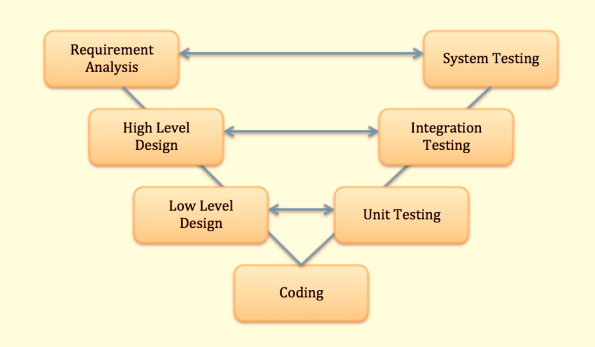
All these levels constitute the **waterfall method** of software development lifecycle. As you may observe, that **testing in the model starts only after implementation is done**.

But if you are working in the large project, where the systems are complex, it's easy to miss out the key details in the requirements phase itself. In such cases, an entirely wrong product will be delivered to the client and you might have to start afresh with the project OR if you manage to note the requirements correctly but make serious mistakes in design and architecture of your software you will have to redesign the entire software to correct the error.

Assessments of thousands of projects have shown that **defects introduced during requirements & design make up close to half of the total number of defects.**

Also, the **costs of fixing a defect increases across the development life cycle**. **The earlier in life cycle a defect is detected, the cheaper it is to fix it.**As the say, "A stitch in time saves a nine."

To address this concern, **the V model of testing** was developed where **for every phase, in the Development life cycle there is a corresponding Testing phase**

[](http://cdn.guru99.com/images/6-2015/052715_0904_GuidetoSDLC3.png)

* The left side of the model is Software Development Life Cycle - **SDLC**
* The right side of the model is Software Test Life Cycle - **STLC**
* The entire figure looks like a V, hence the name **V - model**

Apart from V model, there are iterative development models, where development is carried in phases, with each phase adding a functionality to the software. Each phase comprises of its independent set of development and testing activities.

Good examples of **Development lifecycles following iterative method** are **Rapid Application Development, Agile Development**

### Conclusion

There are numerous development life cycle models.**Development model selected for a project depends on the aims and goals of that project.**

* Testing is not a stand-alone activity, and it has to adapt the development model chosen for the project.
* In any model, testing should performed at all levels i.e. right from requirements until maintenance.

# 4.Software Testing Life Cycle STLC

Contrary to popular belief, Software Testing is not a just a single activity. It consists of series of activities carried out methodologically to help certify your software product. These activities (stages) constitute the Software Testing Life Cycle (STLC).

The different stages in Software Test Life Cycle –



Each of these stages have a definite [Entry and Exit criteria](http://www.guru99.com/faq.html#2)  , Activities & Deliverables associated with it.

In an Ideal world you will not enter the next stage until the exit criteria for the previous stage is met. But practically this is not always possible. So for this tutorial , we will focus of activities and deliverables for the different stages in STLC. Lets look into them in detail.

## Requirement Analysis

During this phase, [test team](http://www.guru99.com/how-to-organize-a-test-team.html) studies the requirements from a testing point of view to identify the testable requirements. The QA team may interact with various stakeholders (Client, Business Analyst, Technical Leads, System Architects etc) to understand the requirements in detail. Requirements could be either Functional (defining what the software must do) or Non Functional (defining system performance /[security](http://www.guru99.com/ethical-hacking-tutorials.html" \o "Security)availability ) .Automation feasibility for the given testing project is also done in this stage.

### ****Activities****

* Identify types of tests to be performed.
* Gather details about testing priorities and focus.
* Prepare [Requirement Traceability Matrix (RTM)](http://www.guru99.com/traceability-matrix.html).
* Identify test environment details where testing is supposed to be carried out.
* Automation feasibility analysis (if required).

### ****Deliverables****

* RTM
* Automation feasibility report. (if applicable)

## Test Planning

This phase is also called **Test Strategy** phase. Typically , in this stage, a Senior QA manager will determine effort and cost estimates for the project and would prepare and finalize the Test Plan.

### ****Activities****

* Preparation of test plan/strategy document for various types of testing
* Test tool selection
* Test effort estimation
* Resource planning and determining roles and responsibilities.
* Training requirement

### ****Deliverables****

* [Test plan](http://www.guru99.com/test-plan.html) /strategy document.
* [Effort estimation](http://www.guru99.com/testing-estimation.html) document.

## Test Case Development

This phase involves creation, verification and rework of test cases & test scripts. [Test data](http://www.guru99.com/software-testing-test-data.html) , is identified/created and is reviewed and then reworked as well.

### Activities

* Create test cases, automation scripts (if applicable)
* Review and baseline test cases and scripts
* Create test data (If Test Environment is available)

### ****Deliverables****

* Test cases/scripts
* Test data

## Test Environment Setup

Test environment decides the software and hardware conditions under which a work product is tested. Test environment set-up is one of the critical aspects of testing process and ***can be done in parallel with Test Case Development Stage***. [***Test team***](http://www.guru99.com/how-to-organize-a-test-team.html)***may not be involved in this activity*** if the customer/development team provides the test environment in which case the [test team](http://www.guru99.com/how-to-organize-a-test-team.html) is required to do a readiness check (smoke testing) of the given environment.

### Activities

* Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
* Setup test Environment and test data
* Perform smoke test on the build

### Deliverables

* Environment ready with test data set up
* Smoke Test Results.

## Test Execution

 During this phase [test team](http://www.guru99.com/how-to-organize-a-test-team.html) will carry out the testing based on the test plans and the test cases prepared. Bugs will be reported back to the development team for correction and retesting will be performed.

### Activities

* Execute tests as per plan
* Document test results, and log defects for failed cases
* Map defects to test cases in RTM
* Retest the defect fixes
* Track the defects to closure

### Deliverables

* Completed RTM with execution status
* Test cases updated with results
* Defect reports

## Test Cycle Closure

Testing team will meet , discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from the current test cycle. The idea is to remove the process bottlenecks for future test cycles and share best practices for any similar projects in future.

### Activities

* Evaluate cycle completion criteria based on Time,Test coverage,Cost,Software,Critical Business Objectives , Quality
* Prepare test metrics based on the above parameters.
* Document the learning out of the project
* Prepare Test closure report
* Qualitative and quantitative reporting of quality of the work product to the customer.
* Test result analysis to find out the defect distribution by type and severity.

### Deliverables

* Test Closure report
* Test metrics

Finally, ***summary***of STLC along with Entry and Exit Criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STLC Stage** | **Entry Criteria** | **Activity** | **Exit Criteria** | **Deliverables** |
| Requirement Analysis | Requirements Document available (both functional and non functional)  Acceptance criteria defined.  Application architectural document available. | Analyse business functionality to know the business modules and module specific functionalities.  Identify all transactions in the modules. Identify all the user profiles.  Gather user interface/authentication, geographic spread requirements.  Identify types of tests to be performed.  Gather details about testing priorities and focus.  Prepare Requirement Traceability Matrix (RTM).  Identify test environment details where testing is supposed to be carried out.  Automation feasibility analysis (if required). | Signed off RTM  Test automation feasibility report signed off by the client | RTM  Automation feasibility report (if applicable) |
| Test Planning | Requirements Documents  Requirement Traceability matrix.  Test automation feasibility document. | Analyze various testing approaches available  Finalize on the best suited approach  Preparation of test plan/strategy document for various types of testing  Test tool selection  Test effort estimation  Resource planning and determining roles and responsibilities. | Approved test plan/strategy document.  Effort estimation document signed off. | Test plan/strategy document.  Effort estimation document. |
| Test case development | Requirements Documents  RTM and test plan  Automation analysis report | Create test cases, automation scripts (where applicable)  Review and baseline test cases and scripts  Create test data | Reviewed and signed test Cases/scripts  Reviewed and signed test data | Test cases/scripts  Test data |
| Test Environment setup | System Design and architecture documents are available  Environment set-up plan is available | Understand the required architecture, environment set-up  Prepare hardware and software requirement list  Finalize connectivity requirements  Prepare environment setup checklist  Setup test Environment and test data  Perform smoke test on the build  Accept/reject the build depending on smoke test result | Environment setup is working as per the plan and checklist  Test data setup is complete  Smoke test is successful | Environment ready with test data set up  Smoke Test Results. |
| Test Execution | Baselined RTM, Test Plan , Test case/scripts are available  Test environment is ready  Test data set up is done  Unit/Integration test report for the build to be tested is available | Execute tests as per plan  Document test results, and log defects for failed cases  Update test plans/test cases, if necessary  Map defects to test cases in RTM  Retest the defect fixes  Regression testing of application  Track the defects to closure | All tests planned are executed  Defects logged and tracked to closure | Completed RTM with execution status  Test cases updated with results  Defect reports |
| Test Cycle closure | Testing has been completed  Test results are available  Defect logs are available | Evaluate cycle completion criteria based on - Time, Test coverage , Cost , Software Quality , Critical Business Objectives  Prepare test metrics based on the above parameters.  Document the learning out of the project  Prepare Test closure report  Qualitative and quantitative reporting of quality of the work product to the customer.  Test result analysis to find out the defect distribution by type and severity | Test Closure report signed off by client | Test Closure report  Test metrics |

**Types of Testing**

# **1.** Unit Testing

Unit testing of software applications is done during the development (coding) of an application.

The objective of unit testing is to isolate a section of code and verify its correctness. In procedural programming a unit may be an individual function or procedure

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. Unit testing is usually performed by the developer.

### Why do Unit Testing? Why it is important?

Sometimes software developers attempt to save time by doing minimal unit testing. This is a myth because skimping on unit testing leads to higher defect fixing costs during system testing, integration testing and even beta testing after the application is completed. Proper unit testing done during the development stage saves both time and money in the end.

### Building unit Test Cases

Unit testing is commonly automated, but may still be performed manually. The IEEE does not favor one over the other. A manual approach to unit testing may employ a step-by-step instructional document.

Under the automated approach-

* A developer could write another section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is done.
* They could also isolate the function to test it more rigorously. This is a more thorough unit testing practice that involves copy and pasting the function to its own testing environment to other than its natural environment. Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces in the product. These dependencies can then be eliminated.

A coder may use a Unit Test Framework to develop automated test cases. Using an automation framework, the developer codes criteria into the test to verify the correctness of the unit. During execution of the test cases, the framework logs those that fail any criterion. Many frameworks will also automatically flag and report in a summary these failed test cases. Depending upon the severity of a failure, the framework may halt subsequent testing.

### Unit Testing Tools

There are several automated tools available to assist with unit testing. We will provide a few examples below:

* Rational Software - Rational Software by IBM has a unit test feature known as "Rational Test Realtime". The software contains a complete range of testing tools for much more than just unit testing. It is used for Ada, Java, C and C++. It creates unit tests by reverse engineering the software.  Operating systems it supports include Windows, Linux, Solaris, HP-UX and AIX. Go to <http://www-01.ibm.com/software/rational/> to learn more.
* JavaScript Assertion Unit- Also known as jsAsserUnit, this Freeware JavaScript unit testing tool can be used on any platform that supports JavaScript. It is available at<http://jsassertunit.sourceforge.net/docs/index.html>
* CUT - CUT is a Freeware unit test tool for C, C++ and Objective C. It is great for embedded software testing frameworks and desktop applications on Linux and Windows operating systems. Learn more at sourceforge.net by going to <http://sourceforge.net/projects/cut/>.
* Dotunit - Dotunit is a .net framework Freeware unit testing tool. Part of Junit on the Microsoft .net framework, Dotunit is used for automating unit testing on windows systems. This is another tool from sourceforge.net, so look for it at: <http://dotunit.sourceforge.net/>

 Those are just a few of the available unit testing tools. There are lots more, especially for C languages and Java, but you are sure to find a unit testing tool for your programming needs regardless of the language you use.

### Extreme Programming & Unit Testing

Unit testing in Extreme Programming involves the extensive use of testing frameworks. A unit test framework is used in order to create automated unit tests. Unit testing frameworks are not unique to extreme programming, but they are essential to it. Below we look at some of what extreme programming brings to the world of unit testing:

* Tests are written before the code
* Rely heavily on testing frameworks
* All classes in the applications are tested
* Quick and easy integration is made possible

### Unit Testing benefits

* Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.
* Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (i.e. Regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.
* Due to the modular nature of the unit testing, we can tests parts of project without waiting for others to be completed.

### Unit Testing Limitations

* Unit testing can't be expected to catch every error in a program. It is not possible to evaluate all execution paths even in the most trivial programs
* Unit testing by its very nature focuses on a unit of code. Hence it can't catch integration errors or broad system level errors.

### Unit testing best practices

* Unit Test cases should be independent. In case of any enhancements or change in requirements, unit test cases should not be affected.
* Test only one code at a time.
* Follow clear and consistent naming conventions for your unit tests
* In case of change in code in any module, ensure there is a corresponding unit test case for the module and the module passes the tests before changing the implementation
* Bugs identified during unit testing must be fixed before proceeding to the next phase in SDLC
* Adopt a "test as your code" approach. The more code you write without testing the more paths you have to check for errors.

**Summary**

* As you can see, there can be a lot involved in unit testing. It can be complex or rather simple depending on the application being tested and the testing strategies, tools and philosophies used. Unit testing is always necessary on some level. That is a certainty.

# Integration Testing:

#### Definition and Terminologies:

In Integration Testing, individual software modules are integrated logically and tested as a group.

A typical software project consists of multiple software modules, coded by different programmers.  Integration testing focuses on checking data communication amongst these modules.

Hence it is also termed as **'I & T'** (Integration and Testing), **'String Testing'** and sometimes 'Thread Testing'.

#### Need of Integration Testing:

Although each software module is unit tested, defects still exist for various reasons like

* A Module in general is designed by an individual software developer whose understanding and programming logic may differ from other programmers. Integration testing becomes necessary to verify the software modules work in unity
* At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence integration testing becomes necessary.
* Interfaces of the software modules with the database could be erroneous
* External Hardware interfaces, if any, could be erroneous
* Inadequate exception handling could cause issues.

## 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', 'Mail box' and 'Delete mails' and each of them are 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.

## 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', 'Mail box' and 'Delete mails' and each of them are 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.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 Mail box select the an email and click delete button | Selected email should appear in the Deleted/Trash folder |

## Approaches/Methodologies/Strategies of Integration Testing:

The Software Industry uses variety of strategies to execute Integration testing , viz.

* Big Bang Approach :
* Incremental Approach: which is further divided into 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 links to be tested could be missed easily.
* Since the integration testing can commence only after "all" the modules are designed, 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. Process continues until all of the modules are joined and tested successfully.

This process 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.

**Driver**: Calls the Module to be tested.

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

* **Bottom Up**
* **Top Down**

### 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.
* 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.

### 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 lower level are tested inadequately.

## 

## Integration Testing Procedure

The integration test procedure irrespective of the test strategies (discussed above):

1. Prepare the Integration Test Plan
2. Design the Test Scenarios, Cases, and Scripts.
3. Executing the test Cases followed by reporting the defects.
4. Tracking & re-testing the defects.
5. Steps 3 and 4 are repeated until the completion of Integration is successfully.

## Brief Description of Integration Test Plans:

It includes following attributes:

* Methods/Approaches to test (as discussed above).
* Scopes and Out of Scopes Items of Integration Testing.
* Roles and Responsibilities.
* Pre-requisites for Integration testing.
* Testing environment.
* Risk and Mitigation Plans.

## Entry and Exit Criteria.

Entry and Exit Criteria to Integration testing phase in any software development model

**Entry Criteria:**

* Unit Tested Components/Modules
* All High prioritized bugs fixed and closed
* All Modules to be code completed and integrated successfully.
* Integration test Plan, test case, scenarios to be signed off and documented.
* Required Test Environment to be set up for Integration testing

**Exit Criteria:**

* Successful Testing of Integrated Application.
* Executed Test Cases are documented
* All High prioritized bugs fixed and closed
* Technical documents to be submitted followed by release Notes.

## Best Practices/ Guidelines for Integration Testing

* First determine the Integration Test Strategy that could be adopted and later prepare the test cases and test data accordingly.
* Study the Architecture design of the Application and identify the Critical Modules. These need to be tested on priority.
* Obtain the interface designs from the Architectural team and create test cases to verify all of the interfaces in detail. Interface to database/external hardware/software application must be tested in detail.
* After the test cases, it's the test data which plays the critical role.
* Always have the mock data prepared, prior to executing. Do not select test data while executing the test cases.

# What is System Testing?

In the world of software testing, system testing is the testing of a complete and fully integrated software product.

Usually software is only one element of a larger computer based system. Ultimately, software is interfaced with other software/hardware systems. **System testing is actually a series of different tests whose sole purpose is to exercise the full computer based system**.

**System testing falls under the black box testing category of software testing**. White box testing is the testing of the internal workings or code of a software application. In contrast, black box or system testing is the opposite.System testing involves the external workings of the software from the user's perspective.

## What do you verify in System Testing ?

|  |
| --- |
| System testing involves testing the software code for following   * **Testing the fully integrated applications**  including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End scenario testing.. * Verify thorough **testing of every inpu**t in the application to check for desired outputs. * Testing of the **user's experience** with the application. . |

That is a very basic description of what is involved in system testing. You need to build  detailed test cases and test suites that test each aspect of the application as seen from the outside without looking at the actual source code.

## The Software Testing Hierarchy

As with almost any technical process, software testing has a prescribed order in which things should be done. The following is a list of software testing categories arranged in chronological order. These are the steps taken to fully test new software in preparation for marketing it:

* **Unit testing -** testing performed on each module or block of code during development. Unit testing is normally done by the programmer who writes the code.
* **Integration testing -** testing done before, during and after integration of a new module into the main software package. This involves testing of each individual code module. One piece of software can contain several modules which are often created by several different programmers. It is crucial to test each module's effect on the entire program model.
* **System testing -** testing done by a professional testing agent on the completed software product before it is introduced to the market.
* **Acceptance testing -** beta testing of the product done by the actual end users.

## Types of System Testing

There are more than 50 types of System Testing. For an exhaustive list of software testing types click[here](http://www.guru99.com/types-of-software-testing.html). Below we have listed types of system testing a large software development company would typically use

1. **Usability Testing -** Usability testing mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives
2. **Load Testing -** Load testing is necessary to know that a software solution will perform under real life loads.
3. **Regression Testing** - Regression testing involves testing done to make sure none of the changes made over the course of the development process have caused new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.
4. **Recovery Testing -** Recovery testing is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.
5. **Migration Testing -** Migration testing is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.
6. **Functional Testing -** Also known as functional completeness testing, functional testing involves trying to think of any possible missing functions. Testers might make a list of additional functionalities that a product could have to improve it during functional testing.
7. **Hardware/Software Testing -** IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.

## What Types of System Testing Should Testers Use?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| <http://cdn.guru99.com/images/system%20testing%20types.png> | There are over 50 different types of system testing. The specific types used by a tester depend on several variables. Those variables include:   * **Who the tester works for** - This is a major factor in determining the types of system testing a tester will use. Methods used by large companies are different than that used by medium and small companies. * **Time available for testing** - Ultimately, all 50  testing types could be used. Time is often what limits us to using only the types that are most relevant for the software project. * **Resources available to the tester** - Of course some testers will not have necessary resources to conduct a testing type. For example if you are a tester working for a large software development firm, you are likely to have expensive automated testing software not available to others. * **Software Tester's Education** - There is a certain learning curve for each type of software testing available. To use some of the software involved, a tester has to learn how to use it. * **Testing Budget** - Money becomes a factor not just for smaller companies and individual software developers but large companies as well.   Smoke Vs Sanity Testing  Smoke and Sanity testing are the most misunderstood topics in Software Testing. There is enormous amount of literature on the subject, but most of them are confusing. The following article makes an attempt to address the confusion.  The key differences between Smoke and Sanity Testing can be learned with the help of following diagram - what is a Software Build?   If you are developing a simple computer program which consists of only one source code file, you merely need to compile and link this one file, to produce an executable file. This process is very simple. Usually this is not the case. A typical Software Project consists of hundreds or even thousands of source code files. Creating an executable program from these source files is a complicated and time-consuming task. You need to use "build" software to create an executable program and the process is called "**Software Build**"   what is Smoke Testing? Smoke Testing is performed after software build to **ascertain that the critical functionalities of the program is working fine**.It is executed **"before"** any detailed functional or regression tests are executed on the software build.The **purpose is to reject a badly broken application**, so that the QA team does not waste time installing and testing the software application.  In Smoke Testing, the **test cases chosen cover the most important functionality** or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system is working fine. For Example a typical smoke test would be - Verify that the application launches successfully, Check that the GUI is responsive ... etc. what is Sanity Testing? After receiving a **software build, with minor changes in code, or functionality, Sanity testing is performed to ascertain that the bugs have been fixed and no further issues are introduced due to these changes**.The goal is to determine that the proposed functionality works roughly as expected. **If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing**.  The **objective is "not" to verify thoroughly the new functionality**, but to determine that the developer has applied some rationality (sanity) while producing the software. For instance, if your scientific calculator gives the result of 2 + 2 =5! Then, there is no point testing the advanced functionalities like sin 30 + cos 50. Smoke Testing Vs Sanity Testing - Key Differences  |  |  | | --- | --- | | **Smoke Testing** | **Sanity Testing** | | Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine | Sanity Testing is done to check the new functionality / bugs have been fixed | | The objective of this testing is to verify the "stability" of the system in order to proceed with more rigorous testing | The objective of the testing is to verify the "rationality" of the system in order to proceed with more rigorous testing | | This testing is performed by the developers or testers | Sanity testing is usually performed by testers | | Smoke testing is usually documented or scripted | Sanity testing is usually not documented and is unscripted | | Smoke testing is a subset of Regression testing | Sanity testing is a subset of Acceptance testing | | Smoke testing exercises the entire system from end to end | Sanity testing exercises only the particular component of the entire system | | Smoke testing is like General Health Check Up | Sanity Testing is like specialized health check up |  Points to note.  * Both sanity tests and smoke tests are ways to avoid wasting time and effort by quickly determining whether an application is too flawed to merit any rigorous testing. * Sanity Testing is also called **tester acceptance testing**. * Smoke testing performed on a particular build is also known as a **build verification test**. * One of the best industry practice is to conduct a Daily build and smoke test in software projects. * Both smoke and sanity tests **can be executed manually or using an automation tool**.  When automated tools are used, the tests are often initiated by the same process that generates the build itself. * **As per the needs of testing, you may have to execute both Sanity and Smoke Tests on the software build. In such cases you will first execute Smoke tests and then go ahead with Sanity Testing. In industry, test cases for Sanity Testing are commonly combined with that for smoke tests, to speed up test execution. Hence it's a common that the terms are often confused and used interchangeably**  What is Regression Testing? The purpose of regression testing is to confirm that a  recent program or code change has not adversely affected existing features.  Regression testing is nothing but full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.  This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It  ensures that old code still works once  the new code changes are done. 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    Regression Testing Techniques 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 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 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 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 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 * Sample of Successful test cases * 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.  Extent of automation depends on the number of test cases that remain re-usable for successive regression cycles. |     Following are most important tools used for both functional and regression testing:  **Quick Test Professional (**[**QTP**](http://www.guru99.com/quick-test-professional-qtp-tutorial.html)**):**HP Quick Test Professional is automated software designed to automate functional and regression test cases. It uses [VbScript](http://www.guru99.com/vbscript-tutorials-for-beginners.html" \o "VBScript) language for automation. It is a Data driven , Keyword based tool.  **Rational Functional Tester (RFT):**IBM's rational functional tester is a [java](http://www.guru99.com/java-tutorial.html) 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.  **Selenium:**This is an open source tool used for automating web applications. Selenium can be used for browser based regression testing. Regression Testing and Configuration Management Configuration Management during Regression Testing becomes imperative in Agile Environments where 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. Challenges in Regression Testing: Following are the major testing problems for doing regression testing:   * With successive regression runs, test suites become fairly large.  Due to time and budget constraints, the entire regression test suite cannot be executed * Minimizing test suite while achieving maximum test coverage remains a challenge * Determination of frequency of Regression Tests , i.e., after every modification  or every build update or after a bunch of bug fixes, is a challenge.   **Conclusion:**  An effective regression strategy, save organizations both time and money. As per one of the case study in banking domain,  regression saves upto 60% time in bug fixes(which would have been caught by regression tests) and 40%  in money  **What is Non-functional Testing?**  Like functional testing, there are non-functional requirements like**performance, usability, load factor**that are also important.  <http://cdn.guru99.com/images/1-2015/012715_0453_NonFunction1.png>  Many times you have come across to long load time messages while accessing an application.  <http://cdn.guru99.com/images/1-2015/012715_0453_NonFunction2.png>  To address this issue, **performance testing is carried out to check & fine tune system response times.**The goal of performance testing is to reduce response time to an acceptable level. Likewise, you might have seen the message like **"Server Busy".** Hence, **load testing is carried out to check systems performance at different loads i.e. number of users accessing the system.**  Depending on the results and expected usage, more system resources may be added. In general there are three testing types   * **Functional** * **Non - Functional** * **Maintenance**   Under these types of testing, you have multiple TESTING Level's but usually people call them as Testing Types. You may find some difference in this classification in different resources as per the requirement, but the general theme remains the same. This is not the complete list as there are **more than 150 types of testing types** and still adding. No need to bother or worry, you will pick them up as you age in the testing industry. Also, note that not all testing types are applicable to all projects but depend on nature & scope of the project.  **Testing Techniques** **1.** Equivalence partitioning & Boundary value analysis Practically, due to time and budget considerations, it is not possible to perform exhausting testing for each set of test data, especially when there is a large pool test cases. What is Boundary Testing?  * Boundary testing is the process of testing between extreme ends or boundaries between partitions' of the input values. * So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing". * The basic idea in boundary value testing is to select input variable values at their:   1. Minimum  2. Just above the minimum  3. A nominal value  4. Just below the maximum  5. Maximum   * In Boundary Testing, Equivalence Class Partitioning has been a main role * Boundary Testing comes after the Equivalence Class Partitioning. Here we check the boundary values of ECP.  What is Equivalent Class Partitioning?  * It is a black box technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system, etc. In this technique, you divide the set of test condition into a partition that can be considered the same. * It divides the input data of software into different equivalence data classes. * We apply this technique, where there is a range in input field.  Example 1: Equivalence and Boundary Value  * Let's consider the behavior of tickets in the Flight reservation application, while booking a new flight. * Let's consider the behavior of tickets in the Flight reservation application, while booking a new flight.  1. Any Number greater than 10 entered in the reservation column (let say 11) is considered invalid. 2. Any Number less than 1 that is 0 or below, then it is considered invalid. 3. Numbers 1 to 10 are considered valid 4. Any 3 Digit Number say -100 is invalid.   We cannot test all the possible values because if done, the number of test cases will be more than 100.To address this problem, we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown below where the system behavior can be considered the same.  <http://cdn.guru99.com/images/3-2016/032316_0620_Equivalence4.png>  The divided sets are called Equivalence Partitions or Equivalence Classes. Then we pick only one value from each partition for testing. The hypothesis behind this technique is **that if one condition/value in a partition passes all others will also pass**. Likewise**, if one condition in a partition fails, all other conditions in that partition will fail**.  http://cdn.guru99.com/images/3-2016/032316_0620_Equivalence5.png  **Boundary Value Analysis**- in Boundary Value Analysis, you test boundaries between equivalence partitions  <http://cdn.guru99.com/images/3-2016/032316_0620_Equivalence6.png>  In our earlier example instead of checking, one value for each partition you will check the values at the partitions like 0, 1, 10, 11 and so on. As you may observe, you test values at**both valid and invalid boundaries**. Boundary Value Analysis is also called**range checking**.  Equivalence partitioning and boundary value analysis are closely related and can be used together at all levels of testing. Example 2: Equivalence and Boundary Value Suppose a password field accepts minimum 6 characters and maximum 10 characters  That means values in all partitions should be equivalent like " 0-5, 6-10, 11-14 ".   |  |  |  | | --- | --- | --- | | **Test Scenario #** | **Test Scenario Description** | **Expected Outcome** | | 1 | Enter 0 to 5 characters in password field | System should not accept | | 2 | Enter 6 to 10 characters in password field | System should accept | | 3 | Enter 11 to 14 character in password field | System should not accept |   **Examples Of Boundary Testing:**  **Input Box should accept the Number between 1 and 10**  **Test Scenario:**   |  |  |  | | --- | --- | --- | | **Test Scenario #** | **Test Scenario Description** | **Expected Outcome** | | 1 | Min value=1 & Max value=10 | System should accept | | 2 | Min value=0 & Max value=9 | System should NOT accept | | 3 | Min value=2 & Max value=11 | System should NOT accept | | 4 | Min value=0 & Max value=11 | System should NOT accept |  Why Equivalence & Boundary Analysis Testing  1. This testing is used to reduce infinite number of test cases 2. Very clear guidelines on determining test cases 3. Appropriate for calculation-intensive applications with variables that represent physical quantities   **Summary:**   * Boundary Analysis testing is used when practically it is impossible to test large pool of test cases individually * Two techniques - Equivalence Partitioning & Boundary Value Analysis testing techniques is used * In Equivalence Partitioning, first you divide a set of test condition into a partition that can be considered. * In Boundary Value Analysis you then test boundaries between equivalence partitions * Appropriate for calculation-intensive applications with variables that represent physical quantities   **What is Decision Table Testing?**  Decision Table Testing is a good way to deal with a combination of inputs, which produce different results. It helps reduce test effort in verifying each and every combinations of test data, at the same time ensuring complete coverage  Example: To understand the importance of Decision Table Making we will see an example, let's consider the behavior of Flight Button for different combinations of Fly From & Fly To.  **Rule 1:**When destination for both Fly From & Fly To are not set the Flight Icon is disabled. In the decision table, we register values False for Fly From & Fly To and the outcome would be False, which is Flights Button will be disabled. Likewise, you can decide different outcome for different situation.  **Rule 2**: When Fly From destination is set but Fly to is not set, Flight button is disabled. Correspondingly, you register True for Fly from destination in the decision table, and the rest of the entries are false.  **Rule 3**: When Fly from destination is not set but Fly to destination is set, Flight button is disabled and you make entries in the decision table.  **Rule 4:** only when Fly to and Fly from destinations are set, Flights button is enabled and you make the corresponding entry in the decision table.  If you observe the outcomes for Rule 1, 2 & 3 it remains the same. So you can  select any one of them and rule 4 for your testing.  The **significance of this technique becomes immediately clear as the number of inputs increases**.**Number of possible Combinations is given by 2 ^ n , where n is  the number of  Inputs.** **For n = 10,** which is very common in the web based testing, having big input forms, the number of **combinations will be 1024.** Obviously, you cannot test all but **you will choose a rich sub-set of the possible combinations using decision based testing technique**  **What is Use Case Testing?**  **A use case is a description of a particular use of the system by an actor or user**. It is used widely in developing tests at system or acceptance level.  The Use-Case Testing Technique helps identifying test cases that cover the entire system, on a transaction by transaction basis from the start to finish.    In a use-case, **an actor is represented by "A" and system by "S"**. First **we list the Main Success Scenario.**   * Consider the first step of an end to end scenario for a login functionality for our Flight Reservation application where the Actor enters Agent Name and password to login into the Flight Reservation application. * In the next step, the system will validate the password * Next, if the password is correct, the access will be granted * There can be an **extension** **of this use case.**In case password is not valid system will display a message and ask for re-try four times * Or if Password, not valid four times system will close the application   Here we will test the success scenario and one case of each extension.  **What is Domain Testing?**  Domain testing is a software testing technique in which the output of a system has to be tested with a minimum number of inputs in such a case to ensure that the system does not accept invalid and out of range input values.  One of the most important white box testing method is a domain testing. The main goal of the Domain testing is to check whether the system accepts the input within the acceptable range and delivers the required output. Also, it verifies the system should not accept the inputs, conditions and indices outside the specified or valid range.  Domain testing is different from domain specific knowledge you need to test a software system. What is Domain Testing Domain testing is a type of functional testing which tests the application by giving inputs and evaluating its appropriate outputs.  In domain testing, we divide a domain into sub-domains (equivalence classes) and then test using values from each subdomain. For example, if a website (domain) has been given for testing, we will be dividing the website into small portions (subdomain) for the ease of testing.  Domain might involve testing of any one input variable or combination of input variables.  **Simpler practice of domain testing**  Practitioners often study the simplest cases of domain testing less than two other names, "boundary testing" and "equivalence class analysis."  **Boundary testing** - Boundary value analysis (BVA) is based on testing at the boundaries between partitions. We will be testing both the valid and invalid input values in the partition/classes.  **Equivalence Class testing** - The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning.'  That simplified form applies for Domain testing –   1. Only to tests of input variables 2. Only when tested at the system level 3. Only when tested one at a time 4. Only when tested in a very superficial way   It can be simplified as below:   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Variable** | **Valid Class Equivalence Class** | **Invalid Class Equivalence Class** | **Boundaries & Special cases** | **Notes** | | X | 0-100 |  | 0 |  | |  |  |  | 100 |  | |  |  | <0 | -1 |  | |  |  | >100 | 101 |  |   **Explanation:**   1. If a field accepts ranges from 0-100, the field should not accept -1 and 101 as they are invalid entries and beyond the boundaries. 2. The field should accept values such as 0,100 and any number between them.   **Building table like these (in practice)**   1. To build an equivalence class analysis over time put the information into a spreadsheet. Start by listing variables. Add information about them as you obtain it. 2. The table should eventually contain all variables. This means, all input variables, all output variables, and any intermediate variables that you can observe. 3. In practice, most tables that I have seen are incomplete. The best ones seen list all the variables and add detail for critical variables.  Domain testing strategy While domain testing, you need to consider following things,   1. What domain are we testing? 2. How to group the values into classes? 3. Which values of the classes to be tested? 4. How to determine the result?   **What domain are we testing?**  Any domain which we test has some input functionality and an output functionality. There will be some input variables to be entered, and the appropriate output has to be verified. Domain Testing Example  1. Consider a single input test scenario:   C = a+b, where a and b are input variables and C is the output variable.  Here in the above example, there is no need of classification or combination of the variables is required.   1. Consider the below multiple inputs and appropriate output scenario:   Consider a games exhibition for Children, 6 competitions are laid out, and tickets have to be given according to the age and gender input. The ticketing is one of the modules to be tested in for the whole functionality of Games exhibition.  According to the scenario, we got six scenarios based on the age and the competitions:   1. Age >5 and <10, Boy should participate in Storytelling. 2. Age >5 and <10 , girl should participate in Drawing Competition. 3. Age >10 and <15, Boy should participate in Quiz. 4. Age >10 and <15 , girl should participate in Essay writing. 5. Age<5, both boys and girls should participate in Rhymes Competition. 6. Age >15, both boys and girls should participate in Poetry competition.   Here the input will be Age and Gender and hence the ticket for the competition will be issued. This case partition of inputs or simply grouping of values come into the picture.  **How to group the values into classes?**  Partitioning some values means splitting it into non-overlapping subsets.  As we discussed earlier there are two types of partitioning:   1. **Equivalence partitioning** - Equivalence partitioning is a **software testing** technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. 2. **Boundary value analysis -** Boundary value analysis is a **software testing** technique in which tests are designed to include representatives of boundary values in a range. The idea comes from the **boundary**.   For the above example, we are partitioning the values into a subset or the subset. We are partitioning the age into the below classes :   1. **Class 1:** Children with age group 5 to 10 2. **Class 2 :** Children with age group less than 5 3. **Class 3:** Children with age group age 10 to 15 4. **Class 4:** Children with age group greater than 15.   **Which values of the classes to be tested?**  The values picked up for testing should be Boundary values:   1. Boundaries are representatives of the equivalence classes we sample them from. They're more likely to expose an error than other class members, so they're better representatives. 2. The best representative of an equivalence class is a value in between the range.   For the above example we have the following classes to be tested:  For example for the scenario #1:   1. **Class 1:** Children with age group 5 to 10 (Age >5 and <=10)   **Boundary values:**   1. Values should be Equal to or lesser than 10. Hence, age 10 should be included in this class. 2. Values should be greater than 5. Hence, age 5 should not be included in this class. 3. Values should be Equal to or lesser than 10. Hence, age 11 should not be included in this class. 4. Values should be greater than 5. Hence, age 6 should be included in this class.   **Equivalence partition Values:**  Equivalence partition is referred when one has to test only one condition from each partition. In this, we assume that if one condition in a partition works, then all the condition should work. In the same way, if one condition in that partition does not work then we assume that none of the other conditions will work. For example,  (Age >5 and <=10)  As the values from 6 to 10 are valid ones, one of the values among 6,7,8,9 and 10 have to be picked up. Hence selected age "8" is a valid input age for the age group between (Age >5 and <=10). This sort of partition is referred as equivalence partition.  **Scenario**  **Boundary values to be taken**  **Equivalence partitioning values**  Boy - Age >5 and <=10  Input age = 6  Input age = 5  Input age = 11  Input age = 10  Input age = 8  Girl - Age >5 and <=10  Input age = 6  Input age = 5  Input age = 11  Input age = 10  Input age = 8  Boy - Age >10 and <=15  Input age = 11  Input age = 10  Input age = 15  Input age = 16  Input age = 13  Girl - Age >10 and <=15  Input age = 11  Input age = 10  Input age = 15  Input age = 16  Input age = 13  Age<=5  Input age = 4  Input age = 5  Input age = 3  Age >15  Input age = 15  Input age = 16  Input age = 25  **How do we determine whether the program passed or failed the test?**  Passing the functionality not only depends upon the results of the above scenarios. The input given and the expected output will give us the results and this requires domain knowledge.  Determining the results of the example:  http://cdn.guru99.com/images/3-2016/032816_1046_WhatisDomai2.png Domain Testing Structure Usually, testers follow the below steps in a domain testing. These may be customized/ skipped according to our testing needs.   * Identify the potentially interesting variables. * Identify the variable(s) you can analyze now and order them (smallest to largest and vice versa). * Create and identify boundary values and equivalence class values as above. * Identify secondary dimensions and analyze each in a classical way. (In the above example, Gender is the secondary dimension). * Identify and test variables that hold results (output variables). * Evaluate how the program uses the value of this variable. * Identify additional potentially-related variables for combination testing. * Imagine risks that don't necessarily map to an obvious dimension. * Identify and list unanalyzed variables. Gather information for later analysis. * Summarize your analysis with a risk/equivalence table.   **Summary:**  Domain testing, as it is described above, requires knowledge of providing right input to achieve the desired output. Thus, it is only possible to use it for small chunks of code.  **Types of Testing - The Mega List!**  User acceptance is a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This testing happens in the final phase of testing before moving the software application to Market or Production environment.  The main purpose of this testing is to validate the end to end business flow. It does NOT focus on the cosmetic errors, Spelling mistakes or System testing. This testing is carried out in separate testing environment with production like data setup. It is a kind of black box testing where two or more end users will be involved. Who Performs UAT?  * Client * End users  Need of User Acceptance Testing: Once a software has undergone Unit, Integration and System testing the need of Acceptance Testing may seem redundant. **But Acceptance Testing is required because**  http://cdn.guru99.com/images/5-2015/050115_0701_WhatisUserA3.png   * Developers code software based on requirements document which is their "own" understanding of the requirements and **may not actually be what the client needs from the software**. * Requirements changes during the course of the project may not be communicated effectively to the developers.  Acceptance Testing and V-Model In VModel, User acceptance testing corresponds to the requirement phase of the Software Development life cycle(SDLC).  http://cdn.guru99.com/images/5-2015/050115_0701_WhatisUserA4.png How is UAT PerformedPrerequisites of User Acceptance Testing: Following are the entry criteria for User Acceptance Testing:   * Business Requirements must be available. * Application Code should be fully developed * Unit Testing, Integration Testing & System Testing should be completed * No Showstoppers, High, Medium defects in System Integration Test Phase - * Only Cosmetic error are acceptable before UAT * Regression Testing should be completed with no major defects * All the reported defects should be fixed and tested before UAT * Traceability matrix for all testing should be completed * UAT Environment must be ready * Sign off mail or communication from System Testing Team that the system is ready for UAT execution  User Acceptance Testing Process: UAT is done by the intended users of the system or software. This testing usually happens at the client location which is known as Beta Testing. Once Entry criteria for UAT are satisfied, following are the tasks need to be performed by the testers:  <http://cdn.guru99.com/images/5-2015/050115_0701_WhatisUserA6.png>   * Analysis of Business Requirements * Creation of UAT test plan * Identify Test Scenarios * Create UAT Test Cases * Preparation of Test Data(Production like Data) * Run the Test cases * Record the Results * Confirm business objectives  Analysis of Business Requirements One of the most important activities in the UAT is to identify and develop test scenarios. These test scenarios are derived from the following documents:   * Project Charter * Business Use Cases * Process Flow Diagrams * Business Requirements Document(BRD) * System Requirements Specification(SRS)  Creation of UAT Plan: The UAT test plan outlines the strategy that will be used to verify and ensure an application meets its business requirements. It documents entry and **exit criteria for UAT, Test scenarios and test cases approach and timelines of testing**. Identify Test Scenarios and Test Cases: Identify the test scenarios with respect to high level business process and create test cases with clear test steps. Test Cases should sufficiently cover most of the UAT scenarios. Business Use cases are input for creating the test cases. Preparation of Test Data: It is best advisable to use live data for UAT. Data should be scrambled for privacy and [security](http://www.guru99.com/ethical-hacking-tutorials.html) reasons. Tester should be familiar with the data base flow. Run and record the results: Execute test cases and report bugs if any. Re-test bugs once fixed. [Test Management](http://www.guru99.com/test-management.html) tools can used for execution. Confirm Business Objectives met: Business Analysts or UAT Testers needs to send a sign off mail after the UAT testing. After sign-off the product is good to go for production. Deliverables for UAT testing are Test Plan, UAT Scenarios and Test Cases, Test Results and Defect Log Exit criteria for UAT: Before moving into production, following needs to be considered:   * No critical defects open * Business process works satisfactorily * UAT Sign off meeting with all stakeholders   **Qualities of UAT Testers:**  UAT Tester should possess good knowledge of the business. He should be independent and think as an**unknown user to the system**. Tester should be Analytical and Lateral thinker and combine all sort of data to make the UAT successful.  Tester or Business Analyst or Subject Matter Experts who understand the business requirements or flows can prepare test and data which are realistic to the business. Best Practices: Following points needs to be considered to make UAT Success:   * Prepare UAT plan early in the project life cycle * Prepare Checklist before the UAT starts * Conduct Pre-UAT session during System Testing phase itself * Set the expectation and define the scope of UAT clearly * Test End to End business flow and avoid system tests * Test the system or application with real world scenarios and data * Think as an Unknown user to the system * Perform Usability Testing * Conduct Feedback session and meeting before moving to production  UAT Tools There are several tools in the market used for User acceptance testing and some are listed for reference:  [Fitnesse tool](http://www.fitnesse.org/) : It is [java](http://www.guru99.com/java-tutorial.html) tool used as a testing engine. It is easy to create tests and record results in a table. Users of the tool enter the formatted input and tests are created automatically. The tests are then executed and output is returned back to the user.  [Watir](http://watir.com/) : It is tool kit used to automate browser based tests during User acceptance testing. Ruby is the programming language used for inter process communication between ruby and Internet explorer. Some Important points of UAT  * Most of the times in a regular software developing scenarios, UAT is carried out in the QA environment. If there is no staging or UAT environment * UAT is classified into Beta and Alpha testing but it is not so important when software is developed for a service based industry * UAT makes more sense when the customer is involved to a greater extent  Conclusion: UAT is one of the many flavors of testing that has emerged over last twenty five years. With UAT, the client can be sure "What to expect" from the product rather than assuming. The benefit of UAT is that there will be no surprises when the product is released to the market.  **Alpha Vs Beta testing**  Alpha testing is a type of acceptance testing; performed to identify all possible issues/bugs before releasing the product to everyday users or public.  The focus of this testing is to simulate real users by using blackbox and whitebox techniques. The aim is to carry out the tasks that a typical user might perform. Alpha testing is carried out in a lab environment and usually the testers are internal employees of the organization. To put it as simple as possible, this kind of testing is called alpha only because it is done early on, near the end of the development of the software, and before beta testing.  http://cdn.guru99.com/images/AlphaBetaTestingPhase.png What is Beta Testing? Beta Testing of a product is performed by "real users" of the software application in a "real environment" and can be considered as a form of external user acceptance testing.   Beta version of the software is released to a limited number of end-users of the product to obtain feedback on the product quality. Beta testing reduces product failure risks and provides increased quality of the product through customer validation.  It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of Beta Testing. This testing helps to tests the product in real time environment. Alpha Testing versus Beta testing: Following are the differences of Alpha and Beta Testing:   |  |  | | --- | --- | | **Alpha Testing** | **Beta Testing** | | Alpha testing performed by Testers  who are usually internal employees of the organization | Beta testing is performed by Clients or End Users who are not employees of the organization | | Alpha Testing performed at developer's site | Beta testing is performed at client location or end user of the product | | Reliability and security testing are not performed  in-depth Alpha Testing | Reliability, Security, Robustness are checked during Beta Testing | | Alpha testing involves both the white box and black box techniques | Beta Testing typically uses black box testing | | Alpha testing requires lab environment or testing environment | Beta testing doesn't require any lab environment or testing environment. Software is made available to the public and  is said to be real time environment | | Long execution cycle may be required for Alpha testing | Only few weeks of execution are required for Beta testing | | Critical issues or fixes can be addressed by developers immediately in Alpha testing | Most of the issues or feedback is collected from Beta testing will be implemented in future versions of the product | | Alpha testing is to ensure the quality of the product before moving to Beta testing | Beta testing also concentrates on quality of the product, but gathers users input on the product and ensures that the product is ready for real time users. |  Types of Beta Testing There are different types of Beta tests, and they are as follows:  **Traditional Beta testing:**Product is distributed to the target market, and related data is gathered in all aspects. This data can be used for Product improvement.  **Public Beta Testing:**Product is publicly released to the outside world via online channels and data can be gathered from anyone. Based on feedback, product improvements can be done. For example, Microsoft conducted the largest of all Beta Tests for its OS -- Windows 8 before officially releasing it.  **Technical Beta Testing:**Product is released to the internal group of an organization and gather feedback/data from the employees of the organization.  **Focused Beta:**Product is released to the market for gathering feedback on specific features of the program. For example, important functionality of the software.  **Post release Beta:**Product is released to the market and data is gathered to make improvements for the future release of the product. Entry and Exit criteria Entry Criteria for **Alpha testing:**   * Software requirements document or Business requirements specification * Test Cases for all the requirements * Testing Team with good knowledge about the software application * Test Lab environment setup * QA Build ready for execution * Test Management tool for uploading test cases and logging defects * Traceability Matrix to ensure that each design requirement has alteast one test case that verifies it   Exit Criteria for **Alpha testing**   * All the test cases have been executed and passed. * All severity issues need to be fixed and closed * Delivery of Test summary report * Make sure that no more additional features can be included * Sign off on Alpha testing   Entrance criteria for**Beta Testing:**   * Sign off document on Alpha testing * Beta version of the software should be ready * Environment ready to release the software application to the public * Tool to capture real time faults   Exit Criteria **for Beta Testing:**   * All major and minor issues are closed * Feedback report should be prepared from public * Delivery of Beta test summary report  Phases of Testing Alpha and Beta tests are typically carried for "off-the shelf" software or product oriented companies. The Phases of Testing for a product company typically varies from a service oriented organization. Following is the testing phase adopted by product firms  http://cdn.guru99.com/images/TestingPhase.png  **Pre-Alpha**:- Software is a prototype. UI is complete. But not all features are completed. At this stage software is not published.  **Alpha**: Software is near its development and is internally tested for bugs/issues  **Beta**: Software is stable and is released to limited user base. The goal is to get customer feedback on the product and make changes in software accordingly  **Release Candidate (RC):** Based on the feedback of Beta Test, you make changes to the software and want to test out the bug fixes. At this stage,  you do not want to make radical changes in functionality but just check for bugs. RC is also put out to the public  **Release:** All works, software is released to public.  **Note**: Above is a standard definition of the Testing stages but in order to garner marketing buzz, companies combine stages like "pre-alpha beta", "pre-beta" etc. Advantages & Disadvantages **Advantages Beta Testing**   * Reduces product failure risk via customer validation. * Beta Testing allows a company to test post-launch infrastructure. * Improves product quality via customer feedback * Cost effective compared to similar data gathering methods * Creates goodwill with customers and increases customer satisfaction   **Disadvantages Beta Testing**   * Test Management is an issue. As compared to other testing types which are usually executed inside a company in a controlled environment, beta testing is executed out in the real world where you seldom have control. * Finding the right beta users and maintaining their participation could be a challenge   **Advantages of Alpha Testing:**   * Provides better view about the reliability of the software at an early stage * Helps simulate real time user behavior and environment. * Detect many showstopper or serious errors * Ability to provide early detection of errors with respect to design and functionality   **Disadvantages of Alpha Testing:**   * In depth functionality cannot be tested as software is still under development stage Sometimes developers and testers are dissatisfied with the results of alpha testing  Myths of Alpha and Beta Testing: Myth: Alpha and Beta testing are carried out too late in the testing life cycle to merit a benefit  Truth:  Alpha and Beta testing help highlight crucial issues with the software application and provide user feedback.  Myth: Same type of testing is done in Alpha and Beta Testing and they are not different scenarios  Truth: Alpha Testing is done in a lab environment, whereas Beta testing is done by real users reporting real problems based on firsthand experience. Both scenarios are remarkably different.  Myth: Beta type of testing is difficult and time consuming  Truth: Beta testing takes time/experience to obtain the most value. But the data/feedback it provides is invaluable.  Myth: Beta testing generates little or no useful data  Truth: A successful Beta Test can generate tons of valuable information which is otherwise difficult to procure in a lab environment. Conclusion No matter how many tests you perform, how many bugs you kill, your software is useless if your users do not like it. Beta testing (second Greek alphabet)helps provide authentic feedback of your software from real users.  Alpha testing (first Greek alphabet) helps simulate realtime user environment before the software is sent for Beta Testing and helps shape a stable software candidate eligible for Beta Tests.  Alpha and Beta Testing are indispensable in your testing life cycle.  **Usability Testing Tutorial: Need, Process, BestPractice**  In Usability Testing, a small-set of target end-users, of a  software system, "**use**" it to expose usability defects. This testing mainly focuses on the user's-ease to use the application, flexibility in handling controls and ability of the system to meet its objectives.  This testing is recommended during the initial design phase of  SDLC, which gives more visibility on the expectations of the users. Need for Usability Testing Aesthetics and design are important. How well a product looks usually determines how well it works.  There are many software applications / websites, which miserably fail, once launched, due to following reasons -   * Where do I click next? * Which page needs to be navigated? * Which Icon or Jargon represents what? * Error messages are not consistent or effectively displayed * Session time not sufficient.   Usability Testing identifies usability errors in the   system early in development cycle and can save a product from failure. Goals of Usability Testing   Goal of this testing is to satisfy users and it mainly concentrates on the following parameters of a system:  **Effectiveness of the system**   * Is the system is easy to learn? * Is the system useful and adds value to the target audience? * Is Content, Color, Icons, Images used are aesthetically pleasing ?   **Efficiency**   * Navigation required to reach desired screen/webpage should be very less. Scroll bars shouldn't be used frequently. * Uniformity in the **format** of screen/pages in your application/website. * Provision to search within your software application or website   **Accuracy**   * No outdated or incorrect data like contact information/address should be present. * No broken links should be present.   **User Friendliness**  . Controls used should be self-explanatory and must not require training to operate   * Help should be provided for the users to understand the application / website * Alignment with above goals helps in effective usability testing  Usability Testing Process Usability testing process consists of the following phases  **Planning**:-  During this phase the goals of usability test are determined. Having volunteers sit in front of your application and recording their actions is not a goal. You need to determine critical functionalities and objectives of system. You need to assign tasks to your testers, which exercise these critical functionalities. During this phase , usability testing method, number & demographics of usability testers , test report formats are also determined  **Recruiting**: During this phase, you recruit the desired number of testers as per your usability test plan. Finding testers who match your demographic (age , sex etc ) and professional ( education , job etc .) profile can take time.  **Usability** **Testing**: During this phase, usability tests are actually executed.  **Data Analysis**: Data from usability tests is thoroughly analyzed to derive meaningful inferences and give actionable recommendations to improve overall usability of your product.  **Reporting**:  Findings of the usability test is shared with all concerned stakeholders which can include designer, developer, client, and CEO Methods of Usability Testing There are two methods available to do usability testing -   1. Laboratory Usability Testing 2. Remote Usability Testing   **Laboratory Usability Testing:**. This testing is conducted in a separate lab room in presence of the observers. The testers are assigned tasks to execute. The role of the observer is to monitor behavior of the testers and report outcome of  testing. The observer remains silent during the course of testing.  In this testing both observers and testers are present in same physical location.  **Remote Usability Testing**  : Under this testing observers and testers are remotely located. Testers access the System Under Test, remotely and perform assigned tasks. Tester's voice , screen activity , testers facial expressions are recorded by an automated software. Observers analyze this data and report findings of the test. Bestpractices- Usability testing   The primary goal of this testing is to find crucial usability problems before the product is launched . Following things have to be considered to make testing success:   * Start the testing during the early stage of design and development * It's a good practice to conduct usability testing on your competitor's product before you begin development. This will help you determine usability standards for your target audience * Select the appropriate users to test the system(Can be experts/non-experts users/50-50 of Experts and Non-Experts users) * Use a bandwidth shaper . For instance , your target audience has poor network connectivity , limit network bandwidth to say 56 Kbps for your usability testers. * Testers need to concentrate on  critical  & frequently used functionalities of the system. * Assign a single observer to each tester. This helps observer to accurately note tester's behavior. If an observer is assigned to multiple testers, results may be compromised * Educate Designers and Developers that this testing outcomes is not a sign of failure but it's a sign of Improvement  Pros and Cons of Usability testing: As with anything in life, usability testing has its merits and de-merits. Lets look at them  **Pros:**   * It helps uncover usability issues before the product is marketed. * It helps improve end user satisfaction * It makes your system highly effective and efficient * It helps gather true feedback from your target audience who actually use your system during usability test. You do not need to rely on "opinions" from random people.   **Cons:-**   * Cost is a major consideration in usability testing. It takes lots of resources to set up a Usability Test Lab. Recruiting and management of usability testers can also be expensive   However, these costs pay themselves up in form of higher customer satisfaction, retention and repeat business. Usability testing is therefore highly recommended.  **Functional Testing**  Functional testing verifies that each **function** of the software application operates in conformance with the requirement specification. This testing mainly involves black box testing and it is not concerned about the source code of the application.  Each and every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results.  This testing involves checking of User Interface, APIs, Database, security, client/ server applications and functionality of the Application Under Test. The testing can be done either manually or using automation What do you test in Functional Testing? The prime objective of Functional testing is   checking the functionalities of the software system. It mainly concentrates on -   * **Mainline functions**:  Testing the main functions of an application * **Basic Usability**: It involves basic usability testing of the system. It checks whether an user can freely navigate through the screens without any difficulties. * **Accessibility**:  Checks the accessibility of the system for the user * **Error Conditions**: Usage of testing techniques to check for error conditions.  It checks whether suitable error messages are displayed.  Functional Testing Process: In order to functionally test an application, following steps must be observed.  Understand the Requirements  Identify test input (test data)  Compute the expected outcomes with the selected test input values  Execute test cases  Comparison of actual and computed expected result Functional Vs Non-Functional Testing:  |  |  | | --- | --- | | **Functional Testing** | **Non-Functional Testing** | | Functional testing is performed using the functional specification provided by the client and verifies the system against the functional requirements. | Non-Functional testing  checks the  Performance, reliability, scalability and other non-functional aspects of the software system. | | Functional testing is executed first | Non functional testing should be performed after functional testing | | Manual testing or automation tools can be used for functional testing | Using tools will be effective for this testing | | Business requirements are the inputs to functional testing | Performance parameters like speed , scalability are inputs to non-functional testing. | | Functional testing describes what the product does | Nonfunctional testing describes how good the product works | | Easy to do manual testing | Tough to do manual testing | | Types of Functional testing are   * Unit Testing * Smoke Testing * Sanity Testing * Integration Testing * White box testing * Black Box testing * User Acceptance testing * Regression Testing | Types of Non functional testing are   * Performance Testing * Load Testing * Volume Testing * Stress Testing * Security Testing * Installation Testing * Penetration Testing * Compatibility Testing * Migration Testing |  ****Functional testing tools:**** here are several tools available in the marker to perform functional testing. They are explained as follows:   * JUnit - Used mainly for [Java](http://www.guru99.com/java-tutorial.html) applications and this can be used in Unit and system testing * soapUI - This is an open source functional testing tool, mainly used for Web service testing. It supports multiple protocols such HTTP, SOAP and JDBC. * Watir - This is functional testing tool for web applications. It supports tests executed at the web browser and uses ruby scripting language   **Conclusion:**  Functional testing is process of testing functionalities of the system and ensures that the system is working as per the functionalities specified in the business document. The goal of this testing is to check whether the system is functionally perfect!!!  **"End to End Testing"**  Unlike System Testing, End-to-End Testing not only validates the software system under test but also  checks it's integration with external interfaces. Hence, the name "**End-to-End**". The purpose of End-to-End Testing is to exercise a complete production-like scenario. Along with the software system, it also validates batch/data processing from other upstream/downstream systems.  End to End Testing is usually executed after functional and system testing. It uses actual production like data and test environment to simulate real-time settings. End-to-End testing is also called Chain Testing Why End to End Testing ?   Modern software systems are complex and are interconnected with multiple sub-systems  A sub-system may be different from the current system or may be owned by another organization.  **If any one of the sub-system fails, the whole software system could collapse**. This is major risk and can be avoided by End-to-End testing. End-to-End testing verifies the complete system flow. It increase test coverage of various sub-systems. It helps detect issues with sub-systems and increases confidence in the overall software product.   End to End testing Process: The following diagram gives an overview of the End to End testing process.  alt  The chief activities involved in End to End Testing are -   * Study of end to end testing requirements * Test Environment setup and hardware/software requirements * Describe all the systems and its subsystems processes. * Description of roles and responsibilities for all the systems * Testing methodology and standards * End to end requirements tracking and designing of test cases * Input and  output data for each system    How to create End-to-End Test Cases? End to End Testing Design framework consists of three parts   1. Build user functions 2. Build Conditions 3. Build Test Cases   Let's look at them in detail: - Build User Functions Following activities should be done as a part of build user functions:   * List down the features of the system and their interconnected components * List the input data, action and the output data for each feature or function * Identify the relationships between the functions * Determine whether the function can be reusable or independent     For example -Consider a scenario where you login into your bank account and transfer some money to another account from some other bank (3rdparty sub-system)   1. Login into the banking system 2. Check for the balance amount in the account 3. Transfer some  amount from your account to some other bank account (3rdparty sub-system) 4. Check the your latest account balance 5. Logout of the application    Build Conditions based on User Function Following activities are performed as a part of build conditions:   * Building a set of conditions for each user function defined * Conditions include sequence, timing and data conditions   For example -Checking of more conditions like  **Login Page**   * Invalid User Name and Password * Checking with valid user name and password * Password strength checking * Checking of error messages   **Balance Amount**   * Check the current balance after 24 hours.(If the transfer is sent to different bank) * Check for the error message if the transfer amount is greater than the current balance amount    Build Test Scenario Building the test scenario for the user function defined  In this case,   * Login into the system * Check of bank balance amount * Transfer the bank balance amount  Build Multiple Test cases Build one or more test cases for each scenario defined .Test cases may include each condition as single test case.   Metrics for end to end testing:   Following are few of many metrics used for End to End Testing.   * **Test Case preparation status:** It givesTest Case preparation progress against planned * **Weekly Test Progress-**Provides week-wise details of percentage test completion- Failed, not executed & executed against planned for execution tests. * **Defects Status & Details-**It gives Percentage of open & closed defects by week. Also, week-wise defects distribution based on severity and priority * **Environment Availability -**Total number of hours "up" / Total number of hours scheduled per day for testing    Difference between End to End Testing and System Testing:  |  |  | | --- | --- | | **End to End Testing** | **System Testing** | | Validates the software system as well as interconnected sub-systems | Validates just the software system as per the requirements specifications. | | It checks the complete end-to-end process flow. | It checks system functionalities and features. | | All interfaces, backend systems will be considered for testing | Functional and Non-Functional Testing will be considered for testing | | It's executed once system testing is completed. | It's executed after integration testing. | | End to End testing involves checking external interfaces which can be complex to automate. Hence Manual Testing is preferred. | Both Manual and Automation can be performed for system testing |   **Conclusion**  End to end testing is the process verifying a software system along with its sub-systems. The biggest challenge in this testing is to have enough knowledge of the whole system as well as interconnected sub-system.  **Positive Vs Negative testing**  **Software testing** is process of verifying and validating the software or application and checks whether it is working as expected. The intent is to find defects and improve the product quality. There are two ways to test the software , viz,  Positive Testing and Negative Testing.  **Positive testing** can be performed on the system by providing the **valid data** **as input.** It checks whether an application behaves as expected with the positive input. . This is to test to check the application that does what it is  supposed to do so  For example -  alt  There is a text box in an application which can accept only numbers. Entering values up to 99999 will be acceptable by the system and any other values apart from this should not be acceptable. To do positive testing, set the valid input values from 0 to 99999 and check whether the system is accepting the values.  **Negative Testing** can be performed on the system by providing **invalid data as input**. It checks whether an application behaves as expected with the negative input. This is to test the application that does not do anything that it is not supposed to do so. For example –  alt  For the example above  Negative testing can be performed by testing by entering  alphabets characters from A to Z or from a to z. Either system text box should not accept the values or else it should throw an error message for these invalid data inputs.  In both the testing, following needs to be considered:   * Input data * Action which needs to be performed * Output Result   **Testing Technique used for Positive and Negative Testing:**  Following techniques are used for Positive and negative validation of testing is:   * Boundary Value Analysis * Equivalence Partitioning     **Boundary Value Analysis:**  This is one of the software testing technique in which the test cases are designed to include values at the  boundary. If the input data is used within the boundary value limits, then it is said to be Positive Testing. If the input data is picked outside the boundary value limits, then it is said to be Negative Testing.  For example -    A system can accept the numbers from 0 to 10 numeric values. All other numbers are invalid values. Under this technique , boundary values 0 , 10 and -10 will be tested.    alt  **Equivalence Partitioning:**  This is a software testing technique which divides the input date into many partitions .Values from each partition must be tested at least once.  Partitions with valid values are used for Positive Testing. While ,partitions with invalid values are used for negative testing.  alt  For example-  Numeric values Zero to ten can be divided to two( or three )partition. In our case we have two partitions -10 to -1  and  0 to 10. Sample values (5 and -5) can be taken from each part to test the scenarios.  **Conclusion:**  Testing helps deliver quality software application and ensures the software is bug  free before the software is launched. For effective testing, use both - Positive and Negative testing which give enough confidence in the quality of the software. Real time users are can input any values and those needs to be tested before release.  **Static Vs Dynamic Testing**  Under**Static Testing** code is not executed. Rather it manually checks the code, requirement documents, and design documents to find errors. Hence, the name "static".    Main objective of this testing is to improve the quality of software products by finding errors in early stages of the development cycle. This testing is also called as Non-execution technique or verification testing.  Static testing involves manual or automated reviews of the documents. This review, is  done during initial phase of testing to catch defect early in STLC. It examines work documents and provides review comments  Work document can be of following:   * Requirement specifications * Design document * Source Code * Test Plans * Test Cases * Test Scripts * Help or User document * Web Page content   Under**Dynamic Testing** code is executed. It checks for functional behavior of software system , memory/cpu usage and overall performance of the system. Hence the name "Dynamic"  Main objective of this testing is to confirm that the software product works in conformance with  the business requirements. This testing is also called as Execution technique or validation testing.  Dynamic testing executes the software and validates the output with the expected outcome. Dynamic testing is performed at all levels of testing and it can be either black or white box testing. Testing Techniques used for Static Testing:  * **Informal Reviews:**This is one of the type of review which doesn't follow any process to find errors in the document. Under this technique , you just review the document and give informal comments on it. * **Technical Reviews:**A team consisting of your  peers,   review the technical specification of the software product and checks whether it is suitable for the project. They try to  find any discrepancies in the specifications and standards followed. This review concentrates mainly on the technical document related to the software such as Test Strategy, Test Plan and requirement specification documents. * **Walkthrough:**The author of the work product explains the product to his team. Participants can ask questions if any.  Meeting is led by the author. Scribe makes note of review comments * **Inspection:**The main purpose is to find defects and meeting is led by trained moderator. This review is a formal type of review where it follows strict process to find the defects. Reviewers have checklist to review the work products .They record the defect and inform the participants to rectify those errors. * **Static code Review:** This is systematic review of the software source code without executing the code. It checks the syntax of the code, coding standards, code optimization, etc. This is also termed as white box testing .This review can be done at any point during development.  Testing Techniques used for Dynamic Testing:  * **Unit Testing:**Under unit testing ,  individual units or modules is  tested by the developers. It  involves testing of source code by developers. * **Integration Testing:**Individual modules are grouped together and tested by the developers. The purpose is to determine that modules are working as expected once they are integrated. * **System Testing:**System testing is performed on the whole system by checking whether the system or application meets the requirement specification document.  ****Difference between Static and Dynamic Testing:****  |  |  | | --- | --- | | **Static Testing** | **Dynamic Testing** | | Testing done without executing the program | Testing done by executing the program | | This testing does verification process | Dynamic testing does validation process | | Static testing is about prevention of defects | Dynamic testing is about finding and fixing the defects | | Static testing gives assessment of code and documentation | Dynamic testing gives bugs/bottlenecks in the software system. | | Static testing involves checklist and process to be followed | Dynamic testing involves test cases for execution | | This testing can be performed before compilation | Dynamic testing is performed after compilation | | Static testing covers the structural and statement coverage testing | Dynamic testing covers the executable file of the code | | Cost of finding defects and fixing is less | Cost of finding and fixing defects is high | | Return on investment will be high as this process involved at early stage | Return on investment will be low as this process involves after the development phase | | More reviews  comments are highly recommended for good quality | More defects are highly recommended for good quality. | | Requires loads of meetings | Comparatively requires lesser meetings |  ****Conclusion:**** Verification and Validation are two measures used to check that the software product meets the requirements specifications. Static testing involves verification whereas dynamic testing involves validation. Together they help improve software quality.  **Learn Stress Testing with no stress!!!**  Stress testing is used to test the stability & reliability of the system. This test mainly determines the system on its robustness and error handling under extremely heavy load conditions**. It even tests beyond the normal operating point and evaluates how the system works under those extreme conditions. Stress Testing is done to make sure that the system would not crash under crunch situations**  Stress testing is also known as endurance testing. Under Stress Testing, AUT is be stressed for a short period of time to know its withstanding capacity. Most prominent use **of stress testing is to determine the limit, at which the system or software or hardware breaks**. It also checks whether system demonstrates effective error management under extreme conditions.  The application under testing will be stressed when 5GB data is copied from the website and pasted in notepad. Notepad is under stress and gives 'Not Responded' error message. ****Need for Stress Testing**** Consider the following scenarios -   * During festival time, an online shopping site may witness a spike in traffic,or when it announces a sale. * When a blog is mentioned in a leading newspaper, it experiences a sudden surge in traffic.   It is imperative to perform Stress Testing to accommodate such abnormal traffic spikes.Failure to accommodate this sudden traffic may result in loss of revenue and repute.  Stress testing is also extremely valuable for the following reasons:   * To check whether the system works under abnormal conditions. * Displaying appropriate error message when the system is under stress. * System failure under extreme conditions could result in enormous revenue loss * It is better to be prepared for extreme conditions by executing Stress Testing.  Goals of stress testing: The goal of stress testing is to analyze the behavior of the system after failure. For stress testing to be successful, system should display appropriate error message while it is under extreme conditions.  To conduct Stress Testing, sometimes, massive data sets may be used which may get lost during Stress Testing. Testers should not lose this [security](http://www.guru99.com/ethical-hacking-tutorials.html) related data while doing stress testing.  The main purpose of stress testing is to make sure that the system recovers after failure which is called as**recoverability**.  Load Testing is to test the system behavior under normal workload conditions, and it is just testing or simulating with the actual workload  But Stress testing is to test the system behavior under extreme conditions and is carried out till the system failure.  Load testing does not break the system, but stress testing tries to break the system by testing with overwhelming data or resources. Types of Stress Testing: Following are the types of stress testing and are explained as follows: ****Distributed Stress Testing:**** In distributed client-server systems, testing is done across all clients from the server. The role of stress server is to distribute a set of stress tests to all stress clients and track on the status of the client. After the client contacts the server, server adds the name of the client and starts sending data for testing.  Meanwhile, client machines send signal or heart beat that it is connected with the server. If the server does not receive any signals from the client machine, it needs to be investigated further for debugging. From figure, server can connect with the 2 clients (Client1 and Client2), but it cannot send or receive signal from Client 3 & 4.  Night run is the best option to run these stress testing scenarios. Large server farms, need more efficient method for determining which computers have had stress failures that need to be investigated . ****Application Stress Testing:**** This testing concentrate on finding defects related to data locking and blocking, network issues and performance bottlenecks in an application. ****Transactional Stress Testing:**** It does stress testing on one or more transactions between two or more applications. It is used for fine-tuning & optimizing the system. ****Systemic Stress Testing:**** This is integrated stress testing which can be tested across multiple systems running on the same server. It is used to find defects where one application data blocks another application. ****Exploratory Stress Testing:**** This is one of the types of stress testing which is used to test the system with unusual parameters or conditions that are unlikely to occur in a real scenario. It is used to find defects around unexpected scenarios like   1. Large number of users logged at the same time 2. If a virus scanner started in all machines simultaneously 3. If Database gone offline when it being accessed from a web site, 4. When a large volume of data is inserted to the database simultaneously  Tools recommended for Stress Testing:****Stress Tester**** This tool provides extensive analysis of the web application performance, provides results in graphical format, and it is extremely easy to use. No high level scripting is required and gives good return on investment. ****Neo load**** This is a popular tool available in the market to test the web and [mobile](http://www.guru99.com/mobile-testing.html) applications. This tool can simulate thousands of users in order to evaluate the application performance under load and analyze the response times. It also supports Cloud integrated - performance, load and stress testing. It is easy to use, Cost effective, and provides good scalability. ****App Perfect**** AppPerfect is a tool used for integrated load, stress and performance testing such as response time, hit count ratio, resources utilization, scalability and reliability. It is extremely easy to use with minimal programming knowledge and is cost effective. It gives, user-friendly report with defined output like CSV, XLS, HTML and PDF. Metrics for stress testing Metrics help in evaluating a System's performance and generally studied at the end of Stress Test. Commonly used metrics are -  Measuring Scalability & Performance   * Pages per Second :Measures how many pages have been requested / Second * Throughput :Basic Metric - Response data size/Second * Rounds :Number of times test scenarios has been planned Versus Number of times client has executed   Application Response   * Hit time:Average time to retrieve an image or a page * Time to the first byte:Time taken to return the first byte of data or information * Page Time :Time taken to retrieve all the information in a page   Failures   * Failed Connections :Number of failed connections refused by the client(Weak Signal) * Failed Rounds :Number of rounds it gets failed * Failed Hits :Number of failed attempts done by the system(Broken links or unseen images)  Conclusion Stress testing's objective is to check the system under extreme conditions. It monitors system resources such as Memory, processor, network etc, and checks the ability of the system to recover back to normal status. It checks whether system displays appropriate error messages while under stress.  **Performance testing**  What is performance testing? Software performance testing is a means of quality assurance (QA). It involves testing software applications to ensure they 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, 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  Why do performance testing? Performance testing is done to provide stakeholders with information about their application regarding speed, stability and scalability. More importantly, performance testing uncovers what needs to be improved before the product goes to market. Without performance testing, software is likely to suffer from issues such as: running slow while several users use it simultaneously, inconsistencies across different operating systems and poor usability. Performance testing will determine whether or not their software meets speed, scalability and stability requirements under expected workloads. Applications sent to market with poor performance metrics due to non existent or poor performance testing are likely to gain a bad reputation and fail to meet expected sales goals.Also, mission critical applications like space launch programs or life saving medical equipments should be performance tested to ensure that they run for a long period of time without deviations. 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 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 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 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 slow down and try to fix it there. Bottle necking 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 performance of two software systems. It can also help identify parts of your software system which degrade its performance.  Below is a generic performance testing process  performance_testing_process   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 market. The tool you choose for testing will depend on many factors such as types of protocol supported , license cost , hardware requirements , platform support etc. Below is a list of popularly used testing tools.   * [HP Loadrunner](http://www.guru99.com/loadrunner-v12-tutorials.html)**-** 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](http://www.guru99.com/loadrunner-v12-tutorials.html" \o "Loadrunner) features a virtual user generator which simulates the actions of live human users. * [HTTP Load](http://www.acme.com/software/http_load/)**-** a throughput testing tool aimed at testing web servers by running several http or https fetches simultaneously to determine how a server handles the workload. * [Proxy Sniffer](http://www.proxy-sniffer.com/) **-** one of the leading tools used for load testing of web and application servers. It is a cloud based tool that's capable of simulating thousands of users. Summary   **Summary**  Performance testing is necessary before marketing any software product. It ensures customer satisfaction & protects investor's investment against product failure. Costs of performance testing are usually more than made up for with improved customer satisfaction, loyalty and retention.  **How to Load Test your Application**  Load testing is a kind of performance testing which determines a system's performance under real-life load conditions. This testing helps determine how the application behaves when multiple users access it simultaneously.  This testing usually identifies -   * The maximum operating capacity of an application * Determine whether current infrastructure is sufficient to run the application * Sustainability of application with respect to peak user load * Number of concurrent users that an application can support, and scalability to allow more users to access it.   It is a type of non-functional testing. Load testing is commonly used for the Client/Server, Web based applications - both Intranet and Internet. Need of Load Testing:   Some extremely popular sites have suffered serious downtimes when they get massive traffic volumes. E-commerce websites invest heavily in advertising campaigns, but not in Load Testing to ensure optimal system performance, when that marketing brings in traffic.  Consider the following examples   * Popular toy store Toysrus.com, could not handle the increased traffic generated by their advertising campaign resulting in loss of both marketing dollars, and potential toy sales. * An Airline website was not able to handle 10000+ users during a festival offer. * Encyclopedia Britannica declared free access to their online database as a promotional offer. They were not able to keep up with the onslaught of traffic for weeks.  |  | | --- | | Many sites suffer delayed load times when they encounter heavy traffic. Few Facts -   * Most users  click away after 8 seconds delay in loading a page * $ 4.4 Billion Lost annually  due to poor performance |  Why Load Testing ?  * Load testing gives confidence in the system & its reliability and performance. * Load Testing helps identify the bottlenecks in the system under heavy user stress scenarios before they happen in a production environment. * Load testing gives excellent protection against poor performance and accommodates complementary strategies for performance management and monitoring of a production environment.  Goals of Load Testing: Loading testing identifies the following problems before moving the application to market or Production:   * Response time for each transaction * Performance of System components under various loads * Performance of  Database components under different loads * Network delay between the client and the server * Software design issues * Server configuration issues like [Web server](http://www.guru99.com/apache.html), application server, database server etc. * Hardware limitation issues like CPU maximization, memory limitations, network bottleneck, etc.   Load testing will determine whether system needs to be fine-tuned or modification of hardware and software is required to improve performance. Environment needs to be setup before starting the load testing:  |  |  | | --- | --- | | Hardware Platform | Software Configuration | | * Server Machines * Processors * Memory * Disk Storage * Load Machines configuration * Network configuration | * Operating System * Server Software |  Prerequisites of load testing: The chief metric for load testing is response time. Before you begin load testing , you must determine -   * Whether the response time is already measured and compared - Quantitative * Whether the response time is applicable to the business process - Relevant * Whether the response time is justifiable - Realistic * Whether the response time is achievable - Achievable * Whether the response time is measurable using a tool or stopwatch - Measurable  Strategies of load Testing: There are many numbers of ways to perform load testing. Following are a few load testing strategies-   * **Manual Load Testing**: This is one of strategies to execute load testing, but it does not produce repeatable results, cannot provide measurable levels of stress on an application and is an impossible process to coordinate. * **In house developed load testing tools**: An organization, which realizes the importance of load testing, may build their own tools to execute load tests. * **Open source load testing tools**: There are several load testing tools available as open source that are free of charge. They may not be as sophisticated as their paid counterparts, but if you are on a budget, they are the best choice. * **Enterprise class load testing tools**: They usually come with capture/playback facility. They support a large number of protocols. They can simulate an exceptionally large number of users.  ****Load Testing Process:**** The load testing process can be briefly described as below -   1. Create a dedicated test environment for load testing 2. Determine the following 3. Load Test Scenarios 4. Determine load testing transactions for an application    * Prepare Data for each transaction    * Number of Users accessing the system need to be predicted    * Determine connection speeds. Some users may be connected via leased lines while others may use dial-up    * Determine different browsers and operating systems used by the users    * Configuration of all the servers like web, application and DB Servers 5. Test Scenario execution and monitoring . Collecting various metrics 6. Analyze the results. Make recommendations 7. Fin-tune the System 8. Re-test  Guidelines for load testing:  1. Load testing should be planned once the application becomes functionally stable. 2. Large number of unique data should be ready in the data pool 3. Number of users should be decided for each scenario or scripts 4. Avoid creation of detailed logs to conserve the disk IO space 5. Try to avoid downloading of images in the site 6. Consistency of response time over elapsed period should be logged & the same should be compared with various test runs  Difference between Load and Stress testing:  |  |  | | --- | --- | | **Load Testing** | **Stress Testing** | | Load testing identifies the bottlenecks  in the system under various workloads and checks how the system reacts when the load is gradually increased | Stress testing determines the breaking point of the system to reveal the maximum point after which it breaks. |  Difference between Functional and Load Testing:  |  |  | | --- | --- | | **Functional Testing** | **Load Testing** | | Results of functional tests are easily predictable as we have proper steps and preconditions defined | Results of load tests are unpredictable | | Results of functional tests vary slightly | Load test results vary drastically | | Frequency of executing functional testing will be high | Frequency of executing load testing will be low | | Results of functional tests are dependent on the test data | Load testing depends on the number of users. |  Load Testing Tools: Tools recommended for load testing are:   * Loadrunner * Web Load * Astra Load Test * Radview's Web Load * Studio, Rational Site Load * Silk Performer   Loadrunner and Web Load are the popular tools. Their features are listed below - ****Load Runner:**** Load runner is HP tool  used to test the applications under normal and peak load conditions. Load runner generates load by creating virtual users that emulate network traffic. It  simulates real time usage like a production environment and gives graphical results. ****Web load:**** Web load supports HTTP1.0 and 1.1, including cookies, proxies, SSL, TSL, client certificates, authentications, persistent connections and chunked transfer coding. During execution, Webload gathers results at a per-client, per-transaction and per-instance level from the computers that are generating the load. Advantages and disadvantages of Load testing: Following are the advantages of Load testing:   * Performance bottlenecks identification before production * Improves the scalability of the system * Minimize  risk related to system down time * Reduced costs of failure * Increase customer satisfaction   Disadvantages of Load testing:   * Need programming knowledge to use load testing tools. * Tools can be expensive as pricing depends on the number of virtual users supported.  Conclusion: Load testing typically improves performance bottlenecks, scalability and stability of the application before it is available for production. This testing helps to identify the maximum operating capacity of applications as well as system bottlenecks  **What is 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**, implementation details and knowledge of internal paths of the software.This type of testing is based entirely on the software requirements and specifications. Black box testing - Steps Here are the generic steps followed to carry out any type of Black Box Testing.   * Initially requirements and specifications of the system are examined. * Tester chooses valid inputs (positive test scenario) to check whether SUT processes them correctly . Also some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them. * Tester determines expected outputs for all those inputs. * Software tester constructs test cases with the selected inputs. * The test cases are executed. * Software tester compares the actual outputs with the expected outputs. * Defects if any are fixed and re-tested.  Types of Black Box Testing There are many types of Black Box Testing but following are the prominent ones -   * **Functional testing** - This black box testing type is related to functional requirements of a system; it is done by software testers. * **Non-functional testing**- This type of black box testing is not related to testing of a specific functionality , but non-functional requirements  such as performance, scalability, usability. * **Regression testing**- Regression testing is done  after code fixes , upgrades or any other system maintenance to check the new code has not affected the existing code.    Tools used for Black Box Testing: Tools used for Black box testing largely depends on the type of black box testing your are doing. Black box testing strategy: Following are the prominent test strategy amongst the many used in Black box Testing   * **Equivalence Class Testing:** It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage. * **Boundary Value Testing:** Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not.It is very useful in reducing the number of test cases. It is mostly suitable for the systems where input is within certain ranges. * **Decision Table Testing**: A decision table puts causes and their effects in a matrix. There is unique combination in each column.  Comparison of Black Box and White Box Testing: While **White Box Testing (Unit Testing) validates internal structure and working of your software code,**the main **focus of black box testing is on the validation of your functional requirements.**  To conduct White Box Testing , knowledge of underlying programming language is essential. Current day**software systems use a variety of programming languages and technologies** and its not possible to know all of them. **Black box testing gives abstraction from code and focuses testing effort on the software system behaviour.**  Also **software**systems are not developed in a single chunk but **development is  broken down  in different modules**. **Black box testing facilitates testing communication amongst modules**(Integration Testing) .  In case you push code fixes in your live software system , a complete system check (black box regression tests) becomes essential.  Though **White box testing** has its own merits and help **detect many internal errors which may degrade system performance**  Black Box Testing and Software Development Life Cycle (SDLC)    **Black box testing has its own life cycle called Software Test Life Cycle** ([STLC](http://www.guru99.com/software-testing-life-cycle.html)) and it is relative to every stage of Software Development Life Cycle.   * **Requirement** - This is the initial stage of SDLC and in this stage requirement is gathered. Software testers also take part in this stage. * **Test Planning  & Analysis** - [Testing Types](http://www.guru99.com/types-of-software-testing.html) applicable to the project are determined. A Test Plan is created which determines possible project risks and their mitigation. * **Design** - In this stage Test cases/scripts  are created on the basis of software requirement documents * **Test Execution**- In this stage Test Cases prepared are executed. Bugs if any are fixed and re-tested.   **White Box Testing**  White Box Testing is the testing of a software solution's internal coding and infrastructure.It focuses primarily on strengthening security, the flow of inputs and outputs through the application, and improving design and usability. White box testing is also known as **clear box testing, open box testing, logic driven testing or path driven testing or structural testing and glass box testing**.  It is one of two parts of the **"box testing" approach** of software testing. Its counter-part, blackbox testing, involves testing from an external or end-user type perspective. On the other hand, Whitebox testing is based on the inner workings of an application and revolves around internal testing. The term "whitebox" was used because of the see-through box concept. The clear box or whitebox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "black box testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested What do you verify in White Box Testing ?   White box testing involves the testing of the software code for the following:   * Internal security holes * Broken or poorly structured paths in the coding processes * The flow of specific inputs through the code * Expected output * The functionality of conditional loops * Testing of each statement, object and function on an individual basis     The testing can be done at system, integration and unit levels of software development. One of the basic goals of whitebox testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that when a specific input does not result in the expected output, you have encountered a bug. How do you perform White Box Testing?   To give you a simplified explanation of white box testing, we have divided it into **two basic steps**. This is what testers do when testing an application using the white box testing technique:    **STEP 1) UNDERSTAND THE SOURCE CODE**  The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.    **Step 2) CREATE TEST CASES AND EXECUTE**  The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This  method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include manual testing, trial and error testing and the use of testing tools as we will explain further on in this article. White Box Testing Techniques   A major White box testing technique is **Code Coverage analysis**. Code Coverage analysis, eliminates gaps in a test case suite.It identifies areas of a program that are not exercised by a set of test cases.Once gaps are identified, you create test cases to verify untested parts of code, thereby increase the quality of the software product  There are automated tools available to perform Code coverage analysis. Below are a few coverage analysis techniques  **Statement Coverage**  This technique requires **every possible statement in the code to be tested at least once** during the testing process.  **Branch Coverage -**This technique **checks every possible path** (if-else and other conditional loops) of a software application. Tools: An example of a tool that handles branch coverage testing for C, C++ and [Java](http://www.guru99.com/java-tutorial.html) applications is [TCAT-PATH](http://www.soft.com/Products/Coverage/tcatpath.html)    Apart from above, there are **numerous coverage types such as Condition Coverage, Multiple Condition Coverage, Path Coverage, Function Coverag**e etc.Each technique has its own merits and attempts to test (cover) all parts of software code.  Using **Statement and Branch coverage you generally attain 80-90% code coverage** which is sufficient. Types of White Box Testing   White box testingencompasses several testing types used to evaluate the usability of an application, block of code or specific software package. There are listed below --    **Unit Testing :**It is often the first type of testing done on an application. Unit testing is performed on each unit or block of code as it is developed.   Unit testing is essentially done by the programmer. As a software developer, you develop a few lines of code, a single function or an object and test it to make sure it works before continuing  Unit testing helps identify majority of bugs, early in the software development lifecycle. Bugs identified in this stage are cheaper and easy to fix.    **Testing for Memory Leaks** : Memory leaks are leading causes of slower running applications. A QA specialist who is experienced at detecting memory leaks is essential in cases where you have a slow running software application.  There are many tools available to assist developers/testers with memory leak testing , example , [Rational Purify](http://www-01.ibm.com/software/awdtools/purify/win/) for windows application  Apart from above a few testing types are part of both black box and white box testing. They are listed as below -    **White Box Penetration Testing:** In this testing, the tester/developer has full information of the application's source code, detailed network information, IP addresses involved and all server information the application runs on.  The aim is to attack the code from several angles to expose security threats    **White Box Mutation Testing**: Mutation testing is often used to discover the best coding techniques to use for expanding a software solution .    **Ending Notes:**   * White box testing can be quite complex. The complexity involved has a lot to do with the application being tested. A small application that performs a single simple operation could be white box tested in few minutes, while larger programming applications take days, weeks and even longer to fully test. * White box testing should be done on a software application as it is being developed, after it is written and again after each modification |