**Software testing**

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

. It must be remembered that testing does expose many defects existing in a software product.

. Thus testing provides a practical way of reducing defects in a system and increasing the users ‘confidence in a developed system.

**Validation**

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

**Verification**

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

Verification: "Are we building the product right?"   
 Validation: "Are we building the right product?

| **Verification** | **Validation** |
| --- | --- |
|  |  |
| * I t includes checking documents, design, codes and programs. | It includes testing and validating the actual product. |
| * Verification is the static testing. | Validation is the dynamic testing. |
| It does *not* include the execution of the code. | It includes the execution of the code. |
| * Methods used in verification are reviews, walkthroughs, inspections and desk-checking. | Methods used in validation are Black Box Testing, White Box Testing and non-functional testing. |
| * It checks whether the software conforms to specifications or not. | It checks whether the software meets the requirements and expectations of a customer or not. |
| * It can find the bugs in the early stage of the development. | It can only find the bugs that could not be found by the verification process. |
| * The goal of verification is application and software architecture and specification. | The goal of validation is an actual product. |
| * Quality assurance team does verification. | Validation is executed on software code with the help of testing team. |
| * It comes before validation. | It comes after verification. |
| * It consists of checking of documents/files and is performed by human. | It consists of execution of program and is performed by computer. |

Target of the test are -

* **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
* **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
* **Failure**- failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

**Software Testing Strategy**

Software Testing Strategic provide a road map that describes the steps to be conducted as part of testing, planned and undertaken, how much effort, time and resources will be required.

**Testing Strategic Approach**

* To perform effective testing, you should conduct effective technical reviews. By doing this, many errors will be eliminated before testing commences.
* Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
* Different testing techniques are appropriate for different software engineering approaches and at different points in time.
* Testing is conducted by the developer of the software and (for large projects) an independent test group.
* Testing and debugging are different activities, but   
  debugging must be accommodated in any testing   
  strategy.

**TESTING OBJECTIVES:**

1.**To evaluate the work products such as requirements, design, user stories, and code:**

The work products such as **Requirement document, Design**, and **User Stories** should be verified before the developer picks it up for development

2. **To build confidence in the quality level of the test object:** One of the critical objectives of software testing is to improve software quality. High-Quality software means a lesser number of defects

3. **To prevent defects in the software product:**

One of the objectives of software testing is to avoid the mistakes in the early stage of the development. Early detection of errors significantly reduces the cost and effort.

4. **To find defects in the software product:**

Another essential objective of software testing is to identify all defects in a product. The main motto of testing is to find maximum defects in a software product while validating whether the program is working as per the user requirements or not.

5. **To provide sufficient information to stakeholders to allow them to make informed decisions, especially regarding the level of quality of the test object:**

The purpose of testing is to provide complete information to the stakeholders about technical or other restrictions, risk factors, ambiguous requirements, etc.

**6. To reduce the level of risk of insufficient software quality:**

The possibility of loss is also known as risk. The objective of software testing is to reduce the occurrence of the risk

7. **To comply with contractual, legal, or regulatory requirements or standards, and to verify the test object’s compliance with such requirements or standards:**

This objective ensures that software developed for a specific region must follow the legal rules and regulations of that region. Moreover, the software product must be compatible with the national and international standards of testing. We have [**ISO/IEC/IEEE 29119**](https://www.toolsqa.com/software-testing/istqb/software-testing-standards-an-ultimate-guide-2019/) standards that deal with the software testing concept.   
What Testing Shows  
- errors and errors  
- requirements conformance   
- an indication of quality

***Ways of Testing***

[**Static Testing**](http://tryqa.com/what-is-static-testing/)**:**  It can test and find defects without executing code. Static Testing is done during verification process. This testing includes reviewing of the documents (including source code) and static analysis. This is useful and cost effective way of testing.  For example: reviewing, [walkthrough](http://tryqa.com/what-is-walkthrough-in-software-testing/), [inspection](http://tryqa.com/what-is-inspection-in-software-testing/), etc.

[**Dynamic Testing**](http://tryqa.com/what-is-dynamic-testing-technique/)**:**  In dynamic testing the software code is executed to demonstrate the result of running tests. It’s done during validation process. Activities involved in this is Testing the software application (Desktop application, Web application, Mobile Apps)

Importance of Software Testing in Software Engineering

**Software testing is an important part** of software development. If software testing is not performed properly, applications can have errors which may lead to rework, costly failure or worse, loss of life.

There are many examples where software bugs have led to loss of life or millions of dollars in losses. Some of them are listed below.

* Knights Capital Group lost 440 million dollars in 30 minutes due to an error in their trading algorithm on 1 Aug 2012. The company’s share dropped 75% in two days after the software pushed faulty trades for over 150 different stocks.
* Mt. Gox Bitcoin Hack – Mt. Gox which was the largest bitcoin exchange in the world at the time, was hacked in June 2011 and lost approximately 850,000 bitcoins, valued at over half a billion dollars. Deficiency in network protocols which resulted in several k bitcoin being stole hackers use some stole credits card to transfer bitcoin, bugs in the bitcoin software.. Transaction malleability
* The $18 million Mariner 1 Spacecraft was destroyed once it was certain to crash after take off. The failure was traced back to a missing hyphen that let wrong guidance signals to be sent to the rocket. Determine the temp, magnetic field, and charge particles

**Testability**

•Software testability is simply how easily a computer program can be tested.

•Testing must exhibit set of characteristics that achieve the goal of finding errors with a minimum of effort.

**Characteristics of s/w Testability**

•**Operability:** The better it works, the more efficiently it can be tested.

•**Observability:** What you see is what you test.

•**Controllability:** The better we can control the software, the more the testing can be automated and optimized."

•**Decomposability**: By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting.

**Simplicity:** The less there is to test, the more quickly we can test it.

•**Stability:** The fewer the changes, the fewer the disruptions to testing.

•**Understandability:** The more information we have, the smarter we will test.

**TESTING ATTRIBUTES**

**1. A good test has a high probability of finding an error.**

–Tester must understand the software and attempt to develop a mental picture of how the software might fail.

**2. A good test is not redundant.**

–Testing time and resources are limited.

–There is no point in conducting a test that has the same purpose as another test.

–Every test should have a different purpose

**3. A good test should be “best of breed”**

–In a group of tests that have a similar intent, time and resource limitations may mitigate toward the execution of only a subset of these tests.

**4. A good test should be neither too simple nor too complex.**

–sometimes possible to combine a series of tests into one test case, the possible side effects associated with this approach may mask errors.

–Each test should be executed separately

## **Testing Principles**

•All tests should be traceable to customer requirements

•Tests should be planned long before testing begins

•Testing should begin “in the small” and progress toward testing “in the large.”

•Exhaustive testing is not possible

•To be most effective, testing should be conducted by an independent third party

**Fundamental test process in software testing?**

[Testing](http://tryqa.com/what-is-a-software-testing/) is a process rather than a single activity. This process starts from test planning then designing [test cases](http://tryqa.com/test-case/), preparing for execution and evaluating status till the test closure. So, we can divide the activities within the fundamental test process into the following basic steps

1)    Planning and Control  
2)    Analysis and Design  
3)    Implementation and Execution  
4)    Evaluating exit criteria and Reporting  
5)    Test Closure activities

**1)    Planning and Control:**

[**Testplanning**](http://tryqa.com/what-is-the-purpose-and-importance-of-test-plans/) has following major tasks:  
i.  To determine the scope and [risks](http://tryqa.com/what-is-risk-in-software-testing/) and identify the objectives of testing.  
ii. To determine the test approach.  
iii. To implement the test policy and/or the [**test strategy**](http://tryqa.com/what-are-the-test-approaches-or-strategies-in-software-testing/). (Test strategy is an outline that describes the testing portion of the [software development cycle](http://tryqa.com/what-are-the-software-development-life-cycle-sdlc-phases/). It is created to inform PM, testers and developers about some key issues of the testing process

iv. To determine the required test resources like people, test environments, PCs, etc.  
v. To schedule test analysis and design tasks, test implementation, execution and evaluation.  
vi. To determine the **Exit criteria** we need to set criteria such as **Coverage criteria.** (Coverage criteria are the percentage of statements in the software that must be executed during testing. This will help us track whether we are completing test activities correctly.

**Test control** has the following major tasks:  
i.  To measure and analyze the results of reviews and testing.  
ii.  To monitor and document progress, [test coverage](http://tryqa.com/what-is-test-coverage-in-software-testing-its-advantages-and-disadvantages/) and exit criteria.  
iii.  To provide information on testing.  
iv.  To initiate corrective actions.  
v.   To make decisions.

**2)  Analysis and Design: D**ocumentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces.

[**Test analysis**](http://tryqa.com/what-is-test-analysis-or-how-to-identify-the-test-conditions/) **and** [**Test Design**](http://tryqa.com/what-is-test-design-technique/)has the following major tasks:  
i.   To review the **test basis.** (The test basis is the information we need in order to start the test analysis and   create our own test cases.

i.   To identify test conditions.  
iii.  To design the tests.  
iv.  To evaluate testability of the requirements and system.  
v.  To design the test environment set-up and identify and required infrastructure and tools

**Implementation and Execution:**  
During test implementation and execution, we take the test conditions into **test cases** and procedures and other **testware** such as scripts for automation, the test environment and any other test infrastructure

**Test execution** has the following major task:  
**i.** To execute test suites and individual test cases following the test procedures.  
**ii.** To re-execute the tests that previously failed in order to confirm a fix. This is known as **confirmation testing or** [**re-testing**](http://tryqa.com/what-is-retesting/)**.**  
**iii.** To log the outcome of the test execution and record the identities and versions of the software under tests. The **test log** is used for the audit trial

**v.** To Compare actual results with expected results.  
**vi.** Where there are differences between actual and expected results, it report discrepancies as Incidents

**Evaluating Exit criteria and Reporting:**  
Based on the risk assessment of the project we will set the criteria for each test level against which we will measure the “enough testing”. These criteria vary from project to project and are known as **exit criteria**.  
Exit criteria come into picture, when:  
— Maximum test cases are executed with certain pass percentage.  
— Bug rate falls below certain level.  
— When achieved the deadlines.

-To write a test summary report for stakeholders.

**5)  Test Closure activities:**  
Test closure activities are done when software is delivered. The testing can be closed for the other reasons also like:

* When all the information has been gathered which are needed for the testing.
* When a project is cancelled.
* When some target is achieved.
* When a maintenance release or update is done.

**Test closure activities** have the following major tasks:  
i.  To check which planned deliverables are actually delivered and to ensure that all incident reports have been resolved.  
ii. To finalize and archive testware such as scripts, test environments, etc. for later reuse.  
iii. To handover the testware to the maintenance organization. They will give support to the software.  
iv To evaluate how the testing went and learn lessons for future releases and projects.

**Type of Testing**

**Automation** testing is the process of testing the software using an automation tool to find the defects. In this process, testers execute the test scripts and generate the test results automatically by using automation tools. The execution of automation testing provides us various advantages, which are as Reusability, Consistency, Running tests anytime (24/7), Early Bug detection, Less Human Resources. Some of the famous automation testing tools for functional testing include Selenium and Katalon Studio.

**Automation Testing Methodologies**

Automation testing contains the following three different methodologies and approaches, which will help the test engineer to enhance the software product's quality.

* **GUI Testing**
* **Code-Driven**
* **Test Automation Framework**

**GUI Testing** can implement that software or an application, which contains GUIs. So, that the automation test engineers can record user actions and evaluate them many times. The **Test cases** can be written in several programming languages like [JAVA](https://www.javatpoint.com/java-tutorial), [C#](https://www.javatpoint.com/c-sharp-tutorial), [Python](https://www.javatpoint.com/python-tutorial), [Perl](https://www.javatpoint.com/perl-tutorial), etc.

**Code-Driven**

The code-driven technique is the subsequent methodology used in automation testing. In this method, the test engineer will mainly concentrate on test case execution in order to identify whether the several parts of code are performing according to the given requirement or not. it is very a commonly used method in [**agile** software development](https://www.javatpoint.com/agile).

**Test automation framework** Another approach in automation testing is **test automation framework**. The test automation framework is a set of rules used to generate valuable results of the automated testing activity. Similarly, it brings together test data sources, function libraries, object details, and other reusable modules.

**Advantages of Automation Testing**

* Automation testing takes less time than manual testing.
* A tester can test the response of the software if the execution of the same operation is repeated several times.
* Automation Testing provides re-usability of test cases on testing of different versions of the same software.
* Automation testing is reliable as it eliminates hidden errors by executing test cases again in the same way.
* Automation Testing is comprehensive as test cases cover each and every feature of the application.
* It does not require many human resources, instead of writing test cases and testing them manually, they need an automation testing engineer to run them.
* The cost of automation testing is less than manual testing because it requires a few human resources.
* **Disadvantages of Automation Testing**
* Automation Testing requires high-level skilled testers.
* It requires high-quality testing tools.
* When it encounters an unsuccessful test case, the analysis of the whole event is complicated.
* Test maintenance is expensive because high fee license testing equipment is necessary.
* Debugging is mandatory if a less effective error has not been solved, it can lead to fatal results.

**Manual Testing**

Manual testing is a software testing process in which test cases are executed manually without using any automated tool. All test cases executed by the tester manually according to the end user's perspective. It ensures whether the application is working, as mentioned in the requirement document or not. Test cases are planned and implemented to complete almost 100 percent of the software application. Test case reports are also generated manually. The engineer will write the test cases and implement the software on the basis of written test cases.

**Advantages of Manual Testing**

* It does not require programming knowledge while using the Black box method.
* It is used to test dynamically changing GUI designs.
* Tester interacts with software as a real user so that they are able to discover usability and user interface issues.
* It ensures that the software is a hundred percent bug-free.
* It is cost-effective.
* Easy to learn for new testers.

## **Disadvantages of Manual Testing**

* It requires a large number of human resources.
* It is very time-consuming.
* Tester develops test cases based on their skills and experience. There is no evidence that they have covered all functions or not.
* Test cases cannot be used again. Need to develop separate test cases for each new software.
* It does not provide testing on all aspects of testing.
* Since two teams work together, sometimes it is difficult to understand each other's motives, it can mislead the process. manual testing tools, we have various tools such as [Jira](https://www.javatpoint.com/jira-tutorial), [Bugzilla](https://www.javatpoint.com/bugzilla), Mantis, Zap, NUnit, Tessy, LoadRunner, Citrus.
* **Summary**
* Automated testing is more reliable and quicker than manual testing.
* It increases the development team’s productivity (quicker feedback); however, the team can spend a lot of time debugging tests.
* Automated testing allows for different, complex types of testing.
* It improves project quality.
* It doesn’t find visual or UX bugs.

# **Software Testing Levels**

Testing levels are basically to identify missing areas and prevent overlap and repetition between the development life cycle phases. The various levels of testing are:

[Unit testing:](http://tryqa.com/what-is-unit-testing/) It is basically done by the developers to make sure that their code is working fine and meet the user specifications. They test their piece of code which they have written like classes, functions, interfaces and procedures. Unit test is inevitable because:

1. Unit tests help to fix bugs early in the development cycle and save costs.
2. It helps the developers to Better understand the testing code base and enables them to make changes quickly
3. Good unit tests serve as project documentation
4. Unit tests help with code re-use. Migrate both your code and your tests to your new project. Tweak the code until the tests run again.

Eg. calculator software” can have the function of implementing “addition,” while another line of code implements “multiplication,” etc. These addition and multiplication functions and other functions of the calculator software must be individually tested to ensure that the calculator operates correctly. Tools use can be [JUnit](http://junit.org/junit4/) ([Java framework](https://www.softwaretestinghelp.com/java-testing-tools/)), PHPUnit (PHP framework), NUnit (.Net framework) etc. are popular unit testing tools that are used for different languages.

[Component testing:](http://tryqa.com/what-is-component-testing/) It is also called as module testing. The basic difference between the unit testing and component testing is in unit testing the developers test their piece of code but in component testing the whole component is tested. Both the modules are developed separately and when they are tested one by one then we call this as a component or module testing.

Integration testing is done when two modules are integrated, in order to test the behavior and functionality of both the modules after integration. Integration testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated. Integration Testing focuses on checking data communication amongst these modules.

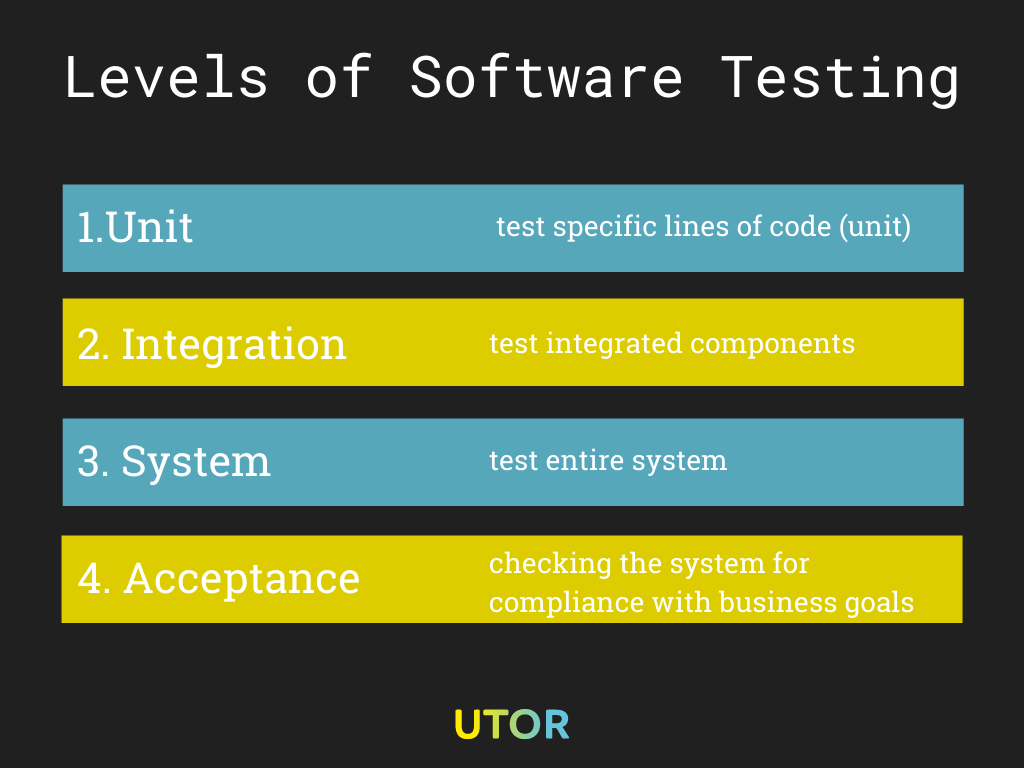
* Unit testing can prove that everything works as expected, while[integration testing](https://u-tor.com/topic/integration-testing) can prove otherwise. This is why we ensure that all recommended levels of software tests are carried out rigorously, to avoid potential pitfalls. Tools use can be Rational Integration Tester, Protractor, Steam

[System testing:](http://tryqa.com/what-is-system-testing/) In system testing the testers basically test the compatibility of the application with the system. System integration testing may be performed after system testing or in parallel with system testing.  In system testing the required operations of the software and its compatibility with operating systems are tested. Eg, checking if a login feature responds when the user enters a password. But in non-functional testing, we check how long it takes the user to log in after password entry. System testing should be carried out by a QA team of software testers.

* Performance testing: it is carried out to check whether the system meet the nonfunctional requirement as specified in the SRS.
* Regression testing: Regression Testing is a type of testing that is done to verify that a code change in the software does not impact the existing functionality of the product. This is to ensure that the product works fine with new functionality, bug fixes or any changes to the existing feature. Previously executed test cases are re-executed in order to verify the impact of the change.

[Acceptance testing:](http://tryqa.com/what-is-acceptance-testing/) Acceptance testing are basically done to determine whether to access or reject the delivery of the sytem. it ensure that the requirements and the specification are met. This stage determines if the software is finally ready to be launched to the general public.

* [Alpha testing:](http://tryqa.com/what-is-alpha-testing/) Alpha testing is done at the developer’s site within the organization. It is done at the end of the development process. Testing is performed to identify all possible issues and bugs before releasing the final product to the end users. Alpha testing is carried out by the testers who are internal employees of the organization. The main goal is to identify the tasks that a typical user might perform and test them.
* [Beta testing:](http://tryqa.com/what-is-beta-testing/) Beta testing is done at the customers site. It is done just before the launch of the product. 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 test products in customer’s environment. Beta version of the software is released to a limited number of end-users of the product to obtain feedback on the product quality.

[](https://u-tor.com/wp-content/uploads/2021/06/Levels-of-Software-Testing.png)

The illustration shows the 4 levels of software testing,

**Example: 1**

Let us understand these three types of testing with an oversimplified example.

For a functional **mobile phone**, the main parts required are **“battery” and “sim card”.**

[**Unit testing Example**](https://www.softwaretestinghelp.com/unit-testing/) – The battery is checked for its **life, capacity 2000mAh** and other parameters. Sim card is checked for its activation.

[**Integration Testing Example**](https://www.softwaretestinghelp.com/what-is-integration-testing/) – Battery and sim card are integrated i.e**. assembled** in order to start the mobile phone.

[**Functional Testing Example**](https://www.softwaretestinghelp.com/guide-to-functional-testing/) – The functionality of a mobile phone is checked in terms of its features and battery usage as well as sim card facilities.

**Example:**2. **Web application requires its users** to Log in

Login” page which has these elements:

* Account/Username
* Password
* Login/Sign in Button

**For Unit testing: the test cases are**

* Field length – username and password fields.
* Input field values should be valid.
* The login button is enabled only after valid values (Format and lengthwise) are entered in both the fields

**For Integration Testing : the test cases are:**

* The user sees the welcome message after entering valid values and pushing the login button.

**Functional Testing Vs. Structural Testing**

In the black-box testing approach, test cases are designed using only the functional specification of the software, i.e. without any knowledge of the internal structure of the software. For this reason, black-box testing is known as functional testing. On the other hand, in the white-box testing approach, designing test cases requires thorough knowledge about the internal structure of software, and therefore the white-box testing is called structural testing.

**Testing Approaches:**

There are three types of software testing approaches.

1. White Box Testing
2. Black Box Testing
3. Grey Box Testing

**WHITE-BOX TESTING**

It is also called Glass Box, Clear Box, and Structural Testing. White Box Testing is based on the application’s internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing is usually done at the unit level. In white box the test case are design to:

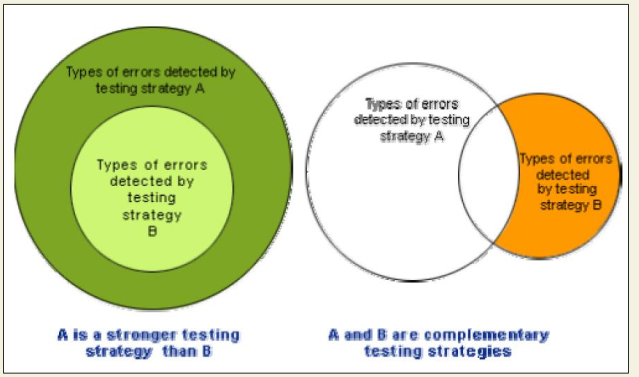
1- Guarantee that all independent paths within a module have been executed once.

2- Exercise all logical decision on their true or false sides.

3- Execute all lops at the boundaries and within their operational bunds and

4- Internal structures to ensure their validity.

One white-box testing strategy is said to be ***stronger***than another strategy, if all types of errors detected by the first testing strategy is also detected by the second testing strategy, and the second testing strategy additionally detects some more types of errors. When two testing strategies detect errors that are different at least with respect to some types of errors, then they are called ***complementary***



**Differential between White box and Black box testing**

|  |  |  |
| --- | --- | --- |
| S/N | White box testing | Black box testing |
| 1. | Performed after knowing the internal structure of a software | Performed without any idea about the internal structure. |
| 2. | Include functional, data driving and close box testing | Code based, structural and clear box testing. |
| 3 | Can be executed by developed or testers | Most executed by testers |
| 4 | Detailed design doc is required for testing | RSD is required |
| 5. | Easy to automate | Taught to automate |
| 6. | Extra lines of coding can be removed along with bugs tracking | Can be used for segments with lot coding |

### **Grey Box Testing:**

* Grey box is the combination of both White Box and Black Box Testing. The tester who works on this type of testing needs to have access to design documents. It is also called [**semi-transparent technique testing**](https://www.softwaretestinghelp.com/grey-box-testing-tutorial/)**which means, the testers are only partially aware** of the internal structure, functions, and designs along with the requirements.  This helps to create better test cases in this process. **Grey Box Testing also covers** Matrix Testing and Pattern Testing.

*~~Buggy software~~*

**Integration Testing Overview**

Integration testing is done to test the modules/components when integrated to verify that they work as expected i.e. to test the modules which are working fine individually does not have issues when integrated.

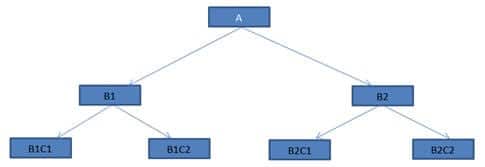
The main function or goal of this testing is to test the interfaces between the units/modules.

We normally do Integration testing after “Unit testing”. Once all the individual units are created and tested, we start combining those “Unit Tested” modules and start doing the integrated testing. the individual modules are first tested in isolation. Once the modules are unit tested, they are integrated one by one, till all the modules are integrated, to check the combinational behavior, and validate whether the requirements are implemented correctly or not.

**Integration Testing Approaches**

**Bottom-up approach:**

### Starts from the lowest or the innermost unit of the application, and gradually moves up. The Integration testing starts from the lowest module and gradually progresses towards the upper modules of the application. This integration continues till all the modules are integrated and the entire application is tested as a single unit.



*Step 1*: Modules B1C1, B1C2 & B2C1, B2C2 are the lowest module which is unit tested.

*Step 2:* Module B1 & B2 (if completed) otherwise Drivers(program or a “stimulator) may replaces the modules. The functionality of Module B1 and B2 is that it calls the modules B1C1, B1C2 & B2C1,

*Step 3:* B1 & B2 will be integrated with module A ultimately, the drivers are then removed and combined moving upward in the program structure.

The advantage of this approach is that, if a major fault exists at the **lowest unit of the program, it is easier to detect it, a**nd corrective measures can be taken.The major **drawback** of this approach is that the higher-level issues can only be identified at the end when all the units have been integrated. that is the main program actually does not exist until the last module is integrated and tested. As a result, the higher level design flaws will be detected only at the end.

**Drivers** serve the same purpose as stubs, but drivers are used in Bottom-up integration testing and are also more complex than stubs. Drivers are also used when some modules are missing and unavailable at time of testing of a specific module because of some unavoidable reasons,

This reduces useless delay in testing and makes the testing process faster.

#### **Top-down approach**

This technique starts from the topmost module and gradually progress towards the lower modules. Only the top module is unit tested in isolation. After this, the lower modules are integrated one by one. The process is repeated until all the modules are integrated and tested

 Step 1: Module A, and lower modules B1 and B2 are integrated one by one.

Step 2: Now here the lower modules B1 and B2 are not actually available for integration. So in order to test the topmost modules A, we develop “**STUBS**”.

Step 3: Modules B1C1, B1C2 & B2C1, B2C2 are test and conducted as each component is integrated

Top-down approach can be Depth-first or Breadth-first integration

Depth-first integration integrate all components on a major control path of the program structure. Breadth-first integration incorporate all component directly subordinate at each level, moving across the structure horizontally.

The top-down approach is a very organic way of integrating as it is consistent with how things happen in the real environment. The drawback is that the major functionality is tested at the end.

***Stubs*** are developed by software developers to use them in place of modules, if the respective modules aren’t developed, missing in developing stage, or are unavailable currently while Top-down testing of modules. A Stub simulates module which has all the capabilities of the unavailable module. Stubs are used when the lower-level modules are needed but are unavailable currently.

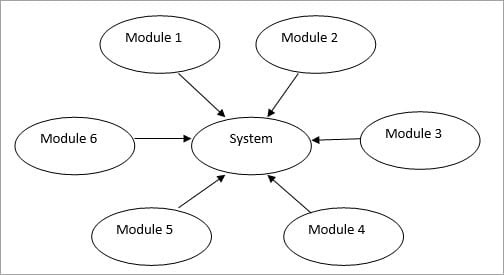
**Note:** The behavior of **stubs** is not that simple most of the time involves complex business logic like connecting to a database. As a result, creating Stubs becomes as complex and time taking as the real module. In some cases, Stub module may turn out to be bigger than the stimulated module.

#### **Big Bang Approach:**

Big bang approach integrates all the modules in one go i.e. it does not go for integrating the modules one by one. It verifies if the system works as expected or not once integrated. If any issue is detected in the completely integrated module, then it becomes difficult to find out which module has caused the issue and therefore it is a good approach for small systems.

Big bang is not system testing, only the integration of modules or units is tested and not the whole system as it is done in system testing. The major **advantage** is that everything integrated is tested at one time.

One major **disadvantage** is that it becomes difficult to identify the failures.



**Sandwich testing**

**Sandwich testing**” which combines the features of both Top-down and bottom-up approach. When we test huge programs like Operating systems, we have to have some more techniques which are efficient and boosts more confidence. Sandwich testing plays a very important role here, where both, the Top down and bottom up testing are started simultaneously. In case of our figure, our testing will start from B1 and B2, where one arm will test the upper module A and another arm will test the lower modules B1C1, B1C2 & B2C1, B2C2.

Since both the approach starts simultaneously, this technique is a bit complex and requires more people along with specific skill sets and thus adds to the cost.

**Comments on Integration Testing**

Selection of an integration strategy depends upon software characteristics and, sometimes, project schedule. In general, a combined approach (sometimes called *sandwich testing*) that uses top-down tests for upper levels of the program structure, coupled with bottom-up tests for subordinate levels may be the best compromise. As integration testing is conducted, the tester should identify *critical modules.*

A critical module has one or more of the following characteristics:

* 1. addresses several software requirements, (2)
  2. has a high level of control (resides relatively high in the program structure),
  3. is complex or error prone ( cyclomatic complexity may be used as an indicator),
  4. has deﬁnite performance requirements. Critical modules should be tested as early as is possible. In addition, regression tests should focus on critical module function

**TEST CASE DESIGN**

The design of tests for software and other engineered products can be as challenging as the initial design of the product itself. Yet for reasons that we have already discussed, software engineers often treat testing as an afterthought, developing test cases that may "feel right" but have little assurance of being complete. Recalling the Objectives of testing, we must design tests that have the highest likelihood of finding the most errors with a minimum amount of time and effort.

•Test case design methods provide a mechanism that can help to ensure the completeness of tests and provide the highest likelihood for uncovering errors in software.

Example:

This code segment has a simple programming error.

* ***if (x>y)***
* ***max = x;***
* ***else***
* ***max = x;***

For the above code segment, the test suite, **{(x=3,y=2);(x=2,y=3)}** can detect the error,

Where as a larger test suite **{(x=3,y=2);(x=4,y=3);(x=5,y=1)}** does not detect the error.

Test suite should be carefully designed than picked randomly. Therefore, systematic approaches should be followed to design an optimal test suite. In an optimal test suite, each test case is designed to detect different errors.