divide and conquer
↳ linearize the control

{1 entry 1 exit}

UNIT 3

Structured programming: In structured programming, we sub-divide the whole program into small modules so that the program becomes easy to understand. The purpose of structured programming is to linearize control flow through a computer program so that the execution sequence follows the sequence in which the code is written. The dynamic structure of the program than resemble the static structure of the program. This enhances the readability, testability, and modifiability of the program. This linear flow of control can be managed by restricting the set of allowed applications construct to a single entry, single exit formats.

Why we use Structured Programming?

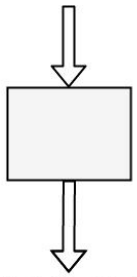
We use structured programming because it allows the programmer to understand the program easily. If a program consists of thousands of instructions and an error occurs then it is complicated to find that error in the whole program, but in structured programming, we can easily detect the error and then go to that location and correct it. This saves a lot of time.

These are the following rules in structured programming:

Structured Rule One: Code Block

If the entry conditions are correct, but the exit conditions are wrong, the error must be in the block. This is not true if the execution is allowed to jump into a block. The error might be anywhere in the program. Debugging under these circumstances is much harder.

Rule 1 of Structured Programming: A code block is structured, as shown in the figure. In flow-charting condition, a box with a single entry point and single exit point are structured. Structured programming is a method of making it evident that the program is correct.

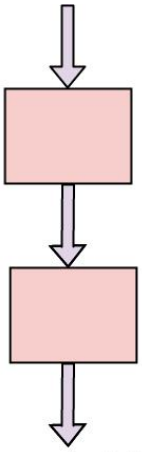


Rule1: Code block is structured

Structure Rule Two: Sequence

A sequence of blocks is correct if the exit conditions of each block match the entry conditions of the following block. Execution enters each block at the block's entry point and leaves through the block's exit point. The whole series can be regarded as a single block, with an entry point and an exit point.

Rule 2 of Structured Programming: Two or more code blocks in the sequence are structured, as shown in the figure.



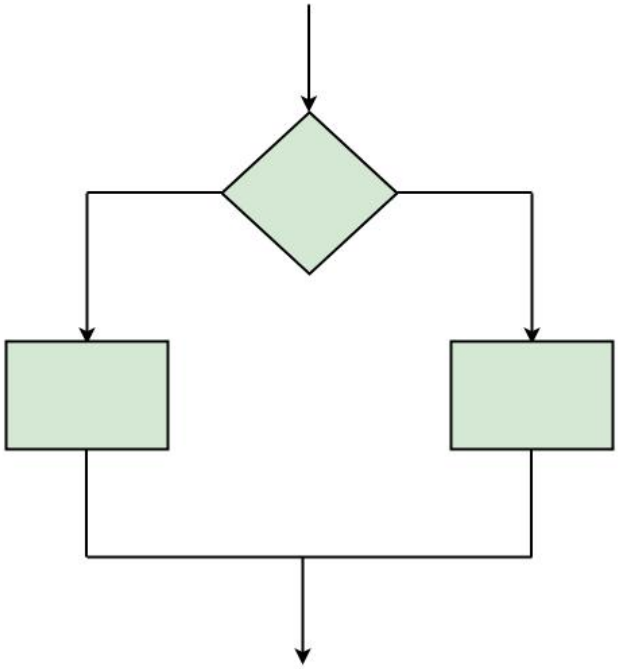
Rule2: A sequence of code blocks is structured

Structured Rule Three: Alternation

If-then-else is frequently called alternation (because there are alternative options). In structured programming, each choice is a code block. If alternation is organized as in the flowchart at right, then there is one entry point (at the top) and one exit point (at the bottom). The structure should be coded so that if the entry conditions are fulfilled, then the exit conditions are satisfied (just like a code block).

Rule 3 of Structured Programming: The alternation of two code blocks is structured, as shown in the figure.

An example of an entry condition for an alternation method is: register $8 includes a signed integer. The exit condition may be: register $8 includes the absolute value of the signed number. The branch structure is used to fulfill the exit condition.

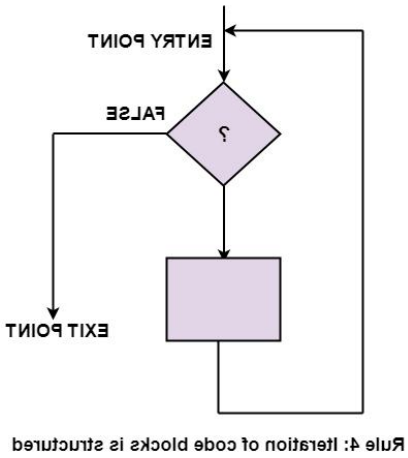


Rule 3: An alternation of code blocks is structured

Structured Rule 4: Iteration

Iteration (while-loop) is organized as at right. It also has one entry point and one exit point. The entry point has conditions that must be satisfied, and the exit point has requirements that will be fulfilled. There are no jumps into the form from external points of the code.

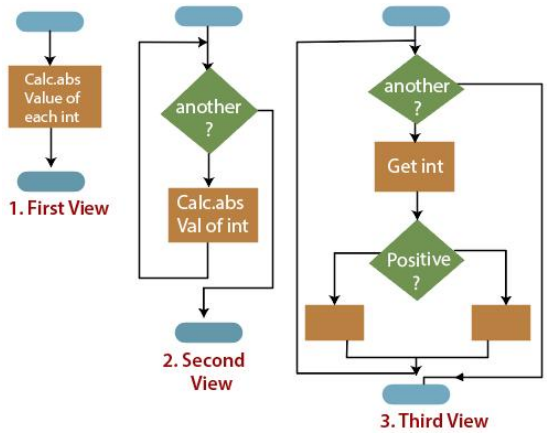
Rule 4 of Structured Programming: The iteration of a code block is structured, as shown in the figure.



Structured Rule 5: Nested Structures

In flowcharting conditions, any code block can be spread into any of the structures. If there is a portion of the flowchart that has a single entry point and a single exit point, it can be summarized as a single code block.

Rule 5 of Structured Programming: A structure (of any size) that has a single entry point and a single exit point is equivalent to a code block. For example, we are designing a program to go through a list of signed integers calculating the absolute value of each one. We may **(1)** first regard the program as one block, then **(2)** sketch in the iteration required, and finally **(3)** put in the details of the loop body, as shown in the figure.



The other control structures are the case, do-until, do-while, and for are not needed. However, they are sometimes convenient and are usually regarded as part of structured programming. In assembly language, they add little convenience.

Code documentation is a manual-cum-guide that helps in understanding and correctly utilizing the software code. The coding standards and naming conventions written in a commonly spoken language in code documentation provide enhanced clarity for the designer. Moreover, they act as a guide for the software maintenance team (this team focuses on maintaining software by improving and enhancing the software after it has been delivered to the end user) while the software maintenance process is carried out. In this way, code documentation facilitates code reusability.

While writing a software code, the developer needs proper documentation for reference purposes. Programming is an ongoing process and requires modifications from time to time. When a number of software developers are writing the code for the same software, complexity increases. With the help of documentation, software developers can reduce the complexity by referencing the code documentation. Some of the documenting techniques are comments, visual appearances of codes, and programming tools. **Comments** are used to make the reader understand the logic of a particular code segment. The **visual appearance of a code** is the way in which the program should be formatted to increase readability. The **programming tools** in code documentation are algorithms, flowcharts, and pseudo-codes.

Code documentation contains source code, which is useful for the software developers in writing the software code. The code documents can be created with the help of various coding tools that are used to auto-generate the code documents. In other words, these documents extract comments from the source code and create a reference manual in the form of text or HTML file. The auto-generated code helps the software developers to extract the source code from the comments. This documentation also contains application programming interfaces, data structures, and algorithms. There are two kinds of code documentation, namely, internal documentation and external documentation.

Documentation which focuses on the information that is used to determine the software code is known as internal documentation. It describes the data structures, algorithms, and control flow in the programs. There are various guidelines for making the documentation easily understandable to the reader. Some of the general conventions to be used at the time of internal documentation are header comment blocks, program comments, and formatting. Header comment blocks are useful in identifying the purpose of the code along with details such as how the c0ge functions and how each segment of code is used in the program.

Since software code is updated and revised several times, it is important to keep a record of the code information so that internal documentation reflects the changes made to the software code. Internal documentation should explain how each code section relates to user requirements in the software. Generally, internal documentation comprises the following information.

- Name, type, and purpose of each variable and data structure used in the code
- Brief description of algorithms, logic, and error-handling techniques
- Information about the required input and expected output of the program
- Assistance on how to test the software
- Information on the upgradations and enhancements in the program.

Documentation which focuses on general description of the software code and is not concerned with its detail is known as external documentation. It includes information such as function of code, name of the software developer who has written the code, algorithms used in the software code,

dependency of code on programs and libraries, and format of the output produced by the software code. Generally, external documentation includes structure charts for providing an outline of the program and describing the design of the program.

External documentation is useful for software developers as it consists of information such as description of the problem along with the program written to solve it. In addition, it describes the approach used to solve the problem, operational requirements of the program, and user interface components. For the purpose of readability and proper understanding, the detailed description is accompanied by figures and illustrations that show how one component is related to another. External documentation explains why a particular solution is chosen and implemented in the software. It also includes formulas, conditions, and references from where the algorithms or documentation are derived. External documentation makes the user aware of the errors that occur while running the software code. For example, if an array of five numbers is used, it should be mentioned in the external documentation that the limit of the array is five.

Code Documentation Tools

While writing software code documentation, it is important to consider the code documentation tools required for writing the software code. The software documentation tools conform to standards by generating the required elements automatically with configurable format and style. These tools combine the selected comment sections with the software code to generate a usable documentation with the essential level of details in it. Some of the code documentation tools are listed in Table.

Table Code Documentation Tools

Documentation Tools	Language Supported	Description
Cocoon	C++	Used to process C++ library files and generates web pages that are useful to document the libraries, classes, and global functions.
CcDoc	C++	Used for implementing the document standards in Java and C++.
CxRef	C	Used to generate documents in HTML, RTF, and so on. It also includes cross-references from source code of C programs.
DOC++	C, C++, Java	Used for providing output for the documentations produced in C, C++, and Java.
JavaDoc	Java	Used as a standard for

		documentation in Java.
Perceps	C++	Used to break C and C++ header files into separate header files. It generates documentation in various formats according to class definitions, declarations, and comments included in those files.
RoboDoc	Assembler, C, Perl, LISP, Fortran, Shell scripts, COBOL	Used to convert formatted documentation into cross-referenced set of HTML pages, which describe the software code.
DocJet	Java, C, C++, Visual Basic	Used to generate documentation from comments in the source code.
ObjectManual	C++	Used to generate documentation in the form of HTML, XML, and RTF pages.
Together	Java, C++	Used to generate documentation from UML and its source code.
Doc-o-matic	C++, C#, ASP.NET, VB.NET, Java, JavaScript, JSP	Used to create documentations such as source code documentation, online help, and user manuals. It is integrated with easy to use interface for managing the documentation projects.

Code documentation tools should be simple to use because easy-to-use documentation tools provide rapid feedback. However, the basic features of software code documentation tools are listed below.

- **Target media:** Views software code easily in a web browser. The target media is useful in displaying the structure and layout of the page with sufficient precision. It is required for code documentation, so that the software code can be easily used in the web browser. An example of target media is **HTML.**

- **Documentation structure:** Includes the index to pages and chapters. The chapters in the documentation should include information such as title, introduction, table of contents, and sections.

- **Comment extraction capabilities:** Extracts the software code comments regardless of the style used in the software code.

- **Languages supported:** Makes the code documentation consistent while writing the software code. The code documentation tools support multiple programming languages and are preferred for concentrating on a particular language.

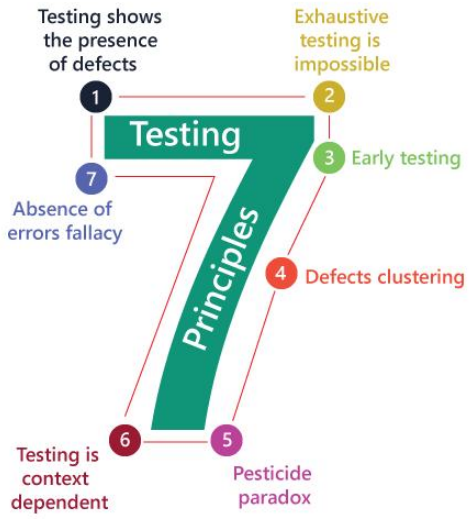
- **Formatting and style elements:** Make the format of the software code proper. Special elements such as tags or mark-ups are required to determine the layout, structure, and style of the code documentation.

- **Code readability:** Makes the software code consistent and easily readable. This is because code documentation is itself not sufficient and requires the comments in the software code to make the code documentation readable and understandable.

Software testing is a procedure of implementing software or the application to identify the defects or bugs. For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time. Here, in this section, we are going to learn about the seven essential principles of software testing.

Let us see the seven different testing principles, one by one:

- Testing shows the presence of defects
- Exhaustive Testing is not possible
- Early Testing
- Defect Clustering
- Pesticide Paradox
- Testing is context-dependent
- Absence of errors fallacy



Testing shows the presence of defects

The test engineer will test the application to make sure that the application is bug or defects free. While doing testing, we can only identify that the application or software has any errors. The primary purpose of doing testing is to identify the numbers of unknown bugs with the help of various methods and testing techniques because the entire test should be traceable to the customer requirement, which means that to find any defects that might cause the product failure to meet the client's needs.

By doing testing on any application, we can decrease the number of bugs, which does not mean that the application is defect-free because sometimes the software seems to be bug-free while performing multiple types of testing on it. But at the time of deployment in the production server, if the end-user encounters those bugs which are not found in the testing process.

Exhaustive Testing is not possible

Sometimes it seems to be very hard to test all the modules and their features with effective and non-effective combinations of the inputs data throughout the actual testing process.

Hence, instead of performing the exhaustive testing as it takes boundless determinations and most of the hard work is unsuccessful. So we can complete this type of variations according to the importance of the modules because the product timelines will not permit us to perform such type of testing scenarios.

Early Testing

Here early testing means that all the testing activities should start in the early stages of the software development life cycle's **requirement analysis stage** to identify the defects because if we find the bugs at an early stage, it will be fixed in the initial stage itself, which may cost us very less as compared to those which are identified in the future phase of the testing process.

To perform testing, we will require the requirement specification documents; therefore, if the requirements are defined incorrectly, then it can be fixed directly rather than fixing them in another stage, which could be the development phase.

Defect clustering

The defect clustering defined that throughout the testing process, we can detect the numbers of bugs which are correlated to a small number of modules. We have various reasons for this, such as the modules could be complicated; the coding part may be complex, and so on.

These types of software or the application will follow the **Pareto Principle**, which states that we can identify that approx. Eighty percent of the complication is present in 20 percent of the modules. With the help of this, we can find the uncertain modules, but this method has its difficulties if the same tests are performing regularly, hence the same test will not able to identify the new defects.

Pesticide paradox

This principle defined that if we are executing the same set of test cases again and again over a particular time, then these kinds of the test will not be able to find the new bugs in the software or the application. To get over these pesticide paradoxes, it is very significant to review all the test cases frequently. And the new and different tests are necessary to be written for the implementation of multiple parts of the application or the software, which helps us to find more bugs.

Testing is context-dependent

Testing is a context-dependent principle states that we have multiple fields such as e-commerce websites, commercial websites, and so on are available in the market. There is a definite way to test the commercial site as well as the e-commerce websites because every application has its own needs, features, and functionality. To check this type of application, we will take the help of various kinds of testing, different technique, approaches, and multiple methods. Therefore, the testing depends on the context of the application.

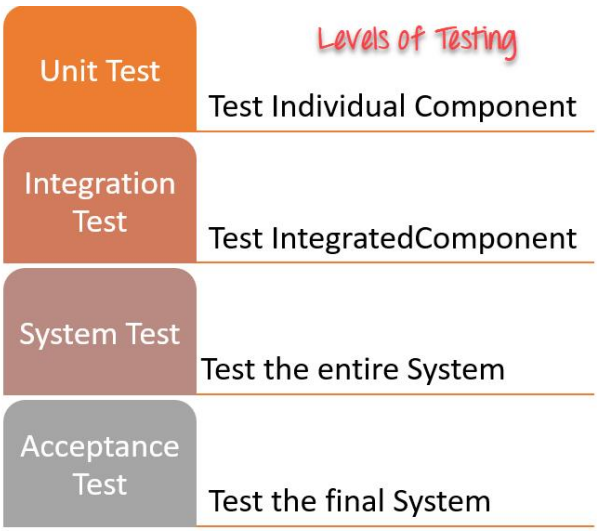
Absence of errors fallacy

Once the application is completely tested and there are no bugs identified before the release, so we can say that the application is 99 percent bug-free. But there is the chance when the application is tested beside the incorrect requirements, identified the flaws, and fixed them on a given period would not help as testing is done on the wrong specification, which does not apply to the client's requirements. The absence of error fallacy means identifying and fixing the bugs would not help if the application is impractical and not able to accomplish the client's requirements and needs.

4 Levels of Testing

There are mainly four **Levels of Testing** in software testing :

1. **Unit Testing** : checks if software components are fulfilling functionalities or not.
2. **Integration Testing** : checks the data flow from one module to other modules.
3. **System Testing** : evaluates both functional and non-functional needs for the testing.
4. **Acceptance Testing** : checks the requirements of a specification or contract are met as per its delivery.



Each Testing Level Details

Unit testing:

A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately.

The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. This kind of testing is performed by developers.

Integration testing:

Integration means combining. For Example, In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing.

Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

System Testing

System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

Acceptance testing:

Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

What is Functional Testing?

FUNCTIONAL TESTING is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.
Functional testing mainly involves black box testing and it is not concerned about the source code of the application. This testing checks User Interface, APIs, Database, Security, Client/Server communication and other 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 a 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.

How to do Functional Testing

Following is a step by step process on **How to do Functional Testing** :

- Understand the Functional Requirements
- Identify test input or test data based on requirements
- Compute the expected outcomes with selected test input values
- Execute test cases
- Compare actual and computed expected results

Structural testing is a type of software testing which uses the internal design of the software for testing or in other words the software testing which is performed by the team which knows the development phase of the software, is known as structural testing.

Structural testing is basically related to the internal design and implementation of the software i.e. it involves the development team members in the testing team. It basically tests different aspects of the software according to its types. Structural testing is just the opposite of behavioral testing.

Types of Structural Testing:

There are 4 types of Structural Testing:

Control Flow Testing:

Control flow testing is a type of structural testing that uses the programs's control flow as a model. The entire code, design and structure of the software have to be known for this type of testing. Often this type of testing is used by the developers to test their own code and implementation. This method is used to test the logic of the code so that required result can be obtained.

Data Flow Testing:

It uses the control flow graph to explore the unreasonable things that can happen to data.

The detection of data flow anomalies are based on the associations between values and variables. Without being initialized usage of variables. Initialized variables are not used once.

Slice Based Testing:

It was originally proposed by Weiser and Gallagher for the software maintenance. It is useful for software debugging, software maintenance, program understanding and quantification of functional cohesion. It divides the program into different slices and tests that slice which can majorly affect the entire software.

Mutation Testing:

Mutation Testing is a type of Software Testing that is performed to design new software tests and also evaluate the quality of already existing software tests. Mutation testing is related to modification a program in small ways. It focuses to help the tester develop effective tests or locate weaknesses in the test data used for the program

Test Plan

A **Test Plan** is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process, which is minutely monitored and controlled by the test manager.

What is Test Case?

A test case is a defined format for software testing required to check if a particular application/software is working or not. A test case consists of a certain set of conditions that need to be checked to test an application or software i.e. in more simple terms when conditions are checked it checks if the resultant output meets with the expected output or not. A test case consists of various parameters such as Id, condition, steps, input, expected result, result, status, and remarks.

Parameters of a Test Case:

- **Module Name:** Subject or title that defines the functionality of the test.
- **Test Case Id:** A unique identifier assigned to every single condition in a test case.
- **Tester Name:** The name of the person who would be carrying out the test.
- **Test scenario: The test** scenario provides a brief description to the tester, as in providing a small overview to know about what needs to be performed and the small features, and components of the test.
- **Test Case Description:** The condition required to be checked for a given software. for eg. Check if only numbers validation is working or not for an age input box.
- **Test Steps:** Steps to be performed for the checking of the condition.
- **Prerequisite:** The conditions required to be fulfilled before the start of the test process.
- **Test Priority:** As the name suggests gives the priority to the test cases as in which had to be performed first, or are more important and which could be performed later.
- **Test Data:** The inputs to be taken while checking for the conditions.
- **Test Expected Result:** The output which should be expected at the end of the test.
- **Test parameters:** Parameters assigned to a particular test case.
- **Actual Result:** The output that is displayed at the end.
- **Environment Information:** The environment in which the test is being performed, such as operating system, security information, the software name, software version, etc.
- **Status:** The status of tests such as pass, fail, NA, etc.
- **Comments:** Remarks on the test regarding the test for the betterment of the software.

Reliability Testing is a testing technique that relates to test the ability of a software to function and given environmental conditions that helps in uncovering issues in the software design and functionality. It is defined as a type of software testing that determines whether the software can

perform a failure free operation for a specific period of time in a specific environment. It ensures that the product is fault free and is reliable for its intended purpose.

Objective of Reliability Testing:

The objective of reliability testing is:

- To find the perpetual structure of repeating failures.
- To find the number of failures occurring is the specific period of time.
- To discover the main cause of failure.
- To conduct performance testing of various modules of software product after fixing defects.

Types of Reliability Testing:

There are three types of reliability testing:-

1. **Feature Testing:**
   Following three steps are involved in this testing:
   - Each function in the software should be executed at least once.
   - Interaction between two or more functions should be reduced.
   - Each function should be properly executed.
2. **Regression Testing:**
   Regression testing is basically performed whenever any new functionality is added, old functionalities are removed or the bugs are fixed in an application to make sure with introduction of new functionality or with the fixing of previous bugs, no new bugs are introduced in the application.
3. **Load Testing:**
   Load testing is carried out to determine whether the application is supporting the required load without getting breakdown. It is performed to check the performance of the software under maximum work load.

The study of reliability testing can be divided into three categories:-

1. Modelling
2. Measurement
3. Improvement

Measurement of Reliability Testing:

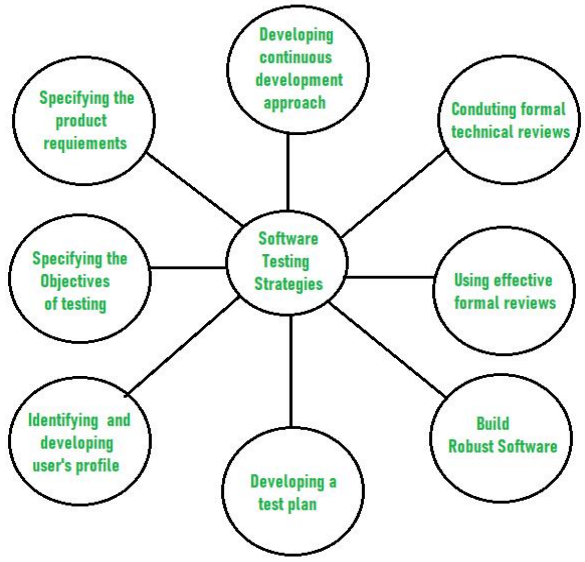
- **Mean Time Between Failures (MTBF):**
  Measurement of reliability testing is done in terms of mean time between failures (MTBF).
- **Mean Time To Failure (MTTF):**
  The time between two consecutive failures is called as mean time to failure (MTTF).
- **Mean Time To Repair (MTTR):**
  The time taken to fix the failures is known as mean time to repair (MTTR).
  MTBF = MTTF + MTTR

Software Testing is a type of investigation to find out if there is any default or error present in the software so that the errors can be reduced or removed to increase the quality of the software and to check whether it fulfills the specifies requirements or not.

According to Glen Myers, software testing has the following objectives:

- The process of investigating and checking a program to find whether there is an error or not and does it fulfill the requirements or not is called testing.
- When the number of errors found during the testing is high, it indicates that the testing was good and is a sign of good test case.

- Finding an unknown error that wasn't discovered yet is a sign of a successful and a good test case.



1. **Before testing starts, it's necessary to identify and specify the requirements of the product in a quantifiable manner.** Different characteristics quality of the software is there such as maintainability that means the ability to update and modify, the probability that means to find and estimate any risk, and usability that means how it can easily be used by the customers or end-users. All these characteristic qualities should be specified in a particular order to obtain clear test results without any error.
2. **Specifying the objectives of testing in a clear and detailed manner.** Several objectives of testing are there such as effectiveness that means how effectively the software can achieve the target, any failure that means inability to fulfill the requirements and perform functions, and the cost of defects or errors that mean the cost required to fix the error. All these objectives should be clearly mentioned in the test plan.
3. **For the software, identifying the user's category and developing a profile for each user.** Use cases describe the interactions and communication among different classes of users and the system to achieve the target. So as to identify the actual requirement of the users and then testing the actual use of the product.
4. **Developing a test plan to give value and focus on rapid-cycle testing.** Rapid Cycle Testing is a type of test that improves quality by identifying and measuring the any changes that need to be required for improving the process of software. Therefore, a test plan is an important and effective document that helps the tester to perform rapid cycle testing.
5. **Robust software is developed that is designed to test itself.** The software should be capable of detecting or identifying different classes of errors. Moreover, software design should allow automated and regression testing which tests the software to find out if there is any adverse or side effect on the features of software due to any change in code or program.
6. **Before testing, using effective formal reviews as a filter.** Formal technical reviews is technique to identify the errors that are not discovered yet. The effective technical reviews conducted before testing reduces a significant amount of testing efforts and time duration required for testing software so that the overall development time of software is reduced.

7. **Conduct formal technical reviews to evaluate the nature, quality or ability of the test strategy and test cases.** The formal technical review helps in detecting any unfilled gap in the testing approach. Hence, it is necessary to evaluate the ability and quality of the test strategy and test cases by technical reviewers to improve the quality of software.
8. **For the testing process, developing a approach for the continuous development.** As a part of a statistical process control approach, a test strategy that is already measured should be used for software testing to measure and control the quality during the development of software.

Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing.
Verification means **Are we building the product right?**
Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product. Validation is the dynamic testing.
Validation means **Are we building the right product?**

Alpha and Beta Testing phases mainly focus on discovering the bugs from an already tested product and they give a clear picture of how the product is used by the real-time users. They also help in gaining experience with the product before its launch and valuable feedback is effectively implemented to increase the usability of the product.

Goals and methods of Alpha & Beta Testing do switch between themselves based on the process followed in the project and can be tweaked to be in-line with the processes.

Both these testing techniques have saved thousands of dollars to large-scale software releases for companies like Apple, Google, Microsoft, etc.

What is Alpha Testing?

This is a form of internal acceptance testing performed mainly by the in-house software QA and testing teams. Alpha testing is the last testing done by the test teams at the development site after the acceptance testing and before releasing the software for beta test.

Alpha testing can also be done by potential users or customers of the application. Still, this is a form of in-house acceptance testing

What is Beta Testing?

This is a testing stage followed by the internal full alpha test cycle. This is the final testing phase where companies release the software to a few external user groups outside the company's test teams or employees. This initial software version is known as the beta version. Most companies gather user feedback in this release.

In short, beta testing can be defined as the testing carried out by real users in a real environment.
Though companies do rigorous in-house quality assurance from dedicated test teams, it's practically impossible to test an application for each and every combination of the test environment. Beta releases make it easier to test the application on thousands of test machines and fix the issues before releasing the application to the public.

The selection of beta test groups can be done based on the company's needs. The company can either invite a few users to test the preview version of the application or they can release it openly to give it a try by any user.

Fixing the issues in the beta release can significantly reduce the development cost as most of the minor glitches get fixed before the final release. So far, many big companies have successfully used beta versions of their most anticipated applications.

For example, recently Microsoft corporation released Windows 10 beta and based on the feedback from thousands of users they managed to release a stable OS version. In the past, Apple also released OS X beta in public and fixed many minor issues and improved the OS based on user feedback.

Alpha Vs Beta Testing

How Alpha and Beta testing differ from each other in various terms:

Alpha Testing	Beta Testing
Basic Understanding	
First phase of testing in Customer Validation	Second phase of testing in Customer Validation
Performed at developer's site - testing environment. Hence, the activities can be controlled	Performed in real environment, and hence activities cannot be controlled
Only functionality, usability are tested. Reliability and Security testing are not usually performed in-depth	Functionality, Usability, Reliability, Security testing are all given equal importance to be performed
White box and / or Black box testing techniques are involved	Only Black box testing techniques are involved
Build released for Alpha Testing is called Alpha Release	Build released for Beta Testing is called Beta Release
System Testing is performed before Alpha Testing	Alpha Testing is performed before Beta Testing
Issues / Bugs are logged into the identified tool directly and are fixed by developer at high priority	Issues / Bugs are collected from real users in the form of suggestions / feedbacks and are considered as improvements for future releases.
Helps to identify the different views of product usage as different business streams are involved	Helps to understand the possible success rate of the product based on real user’s feedback / suggestions.
Test Goals	
To evaluate the quality of the product	To evaluate customer satisfaction
To ensure Beta readiness	To ensure Release readiness (for Production launch)
Focus on finding bugs	Focus on collecting suggestions / feedback and evaluate them effectively
Does the product work?	Do customers like the product?
When	
Usually after System testing phase or when the	Usually after Alpha Testing and product is

Alpha Testing	Beta Testing
product is 70% - 90% complete	90% - 95% complete
Features are almost freezed and no scope for major enhancements	Features are freezed and no enhancements accepted
Build should be stable for technical user	Build should be stable for real users
Test Duration	
Many test cycles conducted	Only 1 or 2 test cycles conducted
Each test cycle lasts for 1 - 2 weeks	Each test cycle lasts for 4 - 6 weeks
Duration also depends on the number of issues found and number of new features added	Test cycles may increase based on real user's feedback / suggestion
Stake Holders	
Engineers (in-house developers), Quality Assurance Team, and Product Management Team	Product Management, Quality Management, and User Experience teams
Participants	
Technical Experts, Specialized Testers with good domain knowledge (new or who were already part of System Testing phase), Subject Matter Expertise	End users to whom the product is designed
Customers and / or End Users can participate in Alpha Testing in some cases	Customers also usually participate in Beta Testing
Expectations	
Acceptable number of bugs that were missed in earlier testing activities	Major completed product with very less amount of bugs and crashes
Incomplete features and documentation	Almost completed features and documentation
Entry Criteria	

Alpha Testing	Beta Testing
• Alpha Tests designed and reviewed for Business requirements • Traceability matrix should be achieved for all the between alpha tests and requirements • Testing team with knowledge about the domain and product • Environment setup and build for execution • Tool set up should be ready for bug logging and test management System testing should be signed-off (ideally)	• Beta Tests like what to test and procedures documented for Product usage • No need of Traceability matrix • Identified end users and customer team up • End user environment setup • Tool set up should be ready to capture the feedback / suggestions • Alpha Testing should be signed off
Exit Criteria	
• All the alpha tests should be executed and all the cycles should be completed • Critical / Major issues should be fixed and retested • Effective review of feedback provided by participants should be completed • Alpha Test Summary report • Alpha testing should be signed off	• All the cycles should be completed • Critical / Major issues should be fixed and retested • Effective review of feedback provided by participants should be completed • Beta Test summary report • Beta Testing should be signed off
Rewards	
No specific rewards or prizes for participants	Participants are rewarded
Pros	
• Helps to uncover bugs that were not found during previous testing activities • Better view of product usage and reliability • Analyze possible risks during and after launch of the product • Helps to be prepared for future customer support • Helps to build customer faith on the product • Maintenance Cost reduction as the bugs are identified and fixed before Beta / Production launch • Easy Test Management	• Product testing is not controllable and user may test any available feature in any way - corner areas are well tested in this case • Helps to uncover bugs that were not found during previous testing activities (including alpha) • Better view of product usage, reliability, and security • Analyze the real user's perspective and opinion on the product • Feedback / suggestions from real users helps in improvising the product in future • Helps to increase customer satisfaction on the product

Alpha Testing	Beta Testing
Cons	
• Not all the functionality of the product is expected to be tested • Only Business requirements are scoped	• Scope defined may or may not be followed by participants • Documentation is more and time consuming - required for using bug logging tool (if required), using tool to collect feedback / suggestion, test procedure (installation / uninstallation, user guides) • Not all the participants assure to give quality testing • Not all the feedback are effective - time taken to review feedback is high • Test Management is too difficult

What is Testing?

It is basically a process using which we verify and validate that an application or software is free of bugs, meets all the technical requirements, abides by all the requirements of development and designing, and meets all the user requirements. Testing ensures that the intended software/ application meets these requirements efficiently and effectively and handles all of the boundary cases and exceptional cases.

What is Debugging?

It is basically a process using which we fix any bug present in a software or application. In this, we first identify, then analyze, and then remove the errors. Debugging begins after the intended software fails in proper execution. Here, we conclude the problem by solving it and testing the software successfully. This process is considered to be extremely tedious and complex because we need to identify and resolve errors present in every stage of debugging.

Differences Between Testing and Debugging

Let us talk about the differences present between Testing and Debugging.

Parameters	Testing	Debugging

Basics	It is the process using which we find errors and bugs.	It is the process using which we correct the bugs that we found during the testing process.
Code Failure	We can identify the failure of any implemented code using this process.	We use this process to provide the code failure with an absolution.
Errors	Errors get displayed in this process.	Errors get deducted and dissolved in this process.
Performer	A tester performs testing on any given software or application.	Either a developer or a programmer performs this step of debugging in an application or software.
Design Knowledge	One does not need any design knowledge to perform testing.	Design knowledge is a prerequisite to debugging.
Insiders and Outsiders	Both- insiders and outsiders can perform testing.	Only an insider can perform debugging. No outsider can perform it on the intended software.
Mode of Operation	The process of testing can be both- automated as well as manual.	The process of debugging must always be manual since we are fixing the available errors and bugs. We cannot debug a system/ software/ application on auto-pilot.
Basis of Operation	The process of testing is based on various levels of testing- system testing, integration testing, unit testing, etc.	The process of debugging is based on various types of bugs present in a system.
SDLC	It is a stage of SDLC (Software Development Life Cycle).	It’s not at all an aspect of the SDLC. It rather occurs as a consequence of the process of testing.
Moment of Initiation	This process can begin after we have completed writing the code.	This process can begin after the execution of the test case and the identification of errors.
Steps	This process consists of both- validation as well as verification of the errors.	This process seeks to match the available symptoms with the cause. Only then it leads to the correction of the errors.