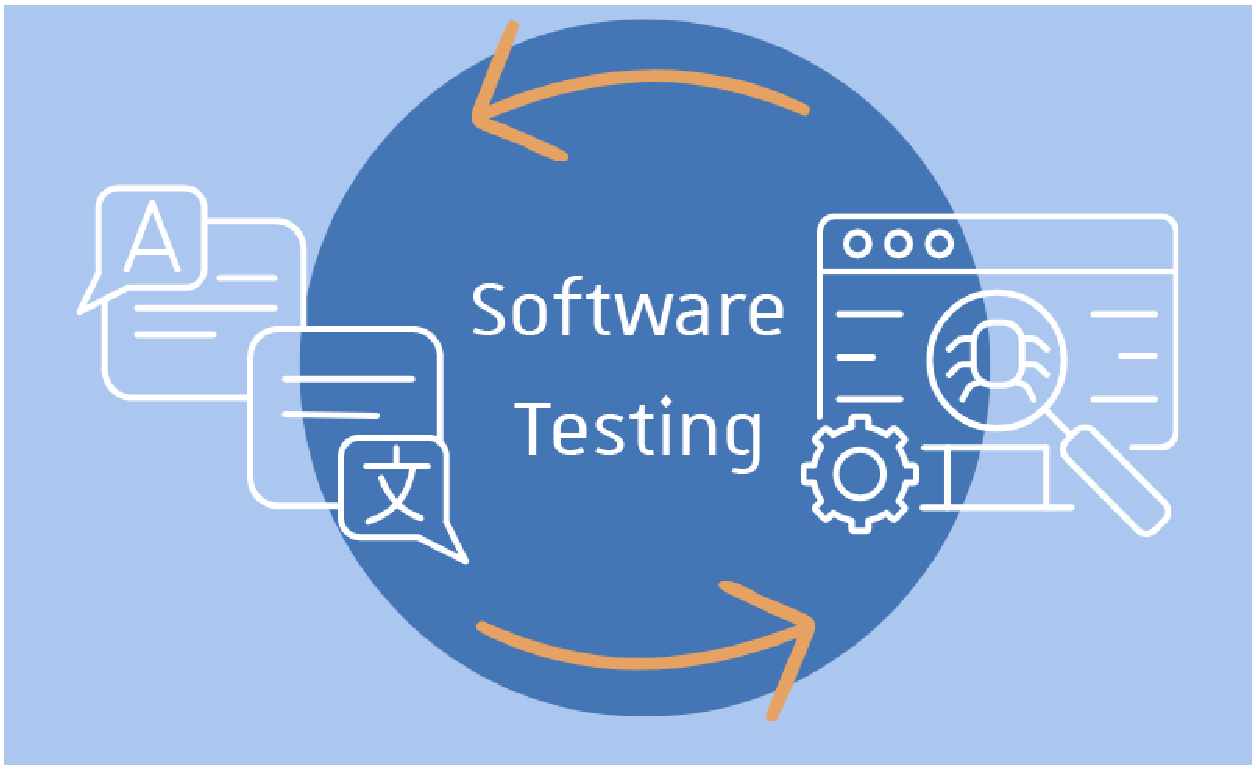
Introduction



Software testing has always been considered a single phase performed after coding.

But time has proved that our failures in software projects are mainly due to the fact that we have not realized the role of software testing as a process.

Thus, its role is not limited to only a single phase in the software development life cycle (SDLC), but it starts as soon as the requirements in a project have been gathered.

Software has pervaded our society, from modern households to spacecraft.

It has become an essential component of any electronic device or system. This is why software development has turned out to be an exciting career for computer engineers in the last 10–15 years.

However, software development faces many challenges. Software is becoming complex, but the demand for quality in software products has increased. This rise in customer awareness for quality increases the workload and responsibility of the software development team. That is why software testing has gained so much popularity in the last decade.

Organizations have separate testing groups with proper hierarchy. Software development is driven with testing outputs. If the testing team claims the presence of bugs in the software, then the development team cannot release the product.

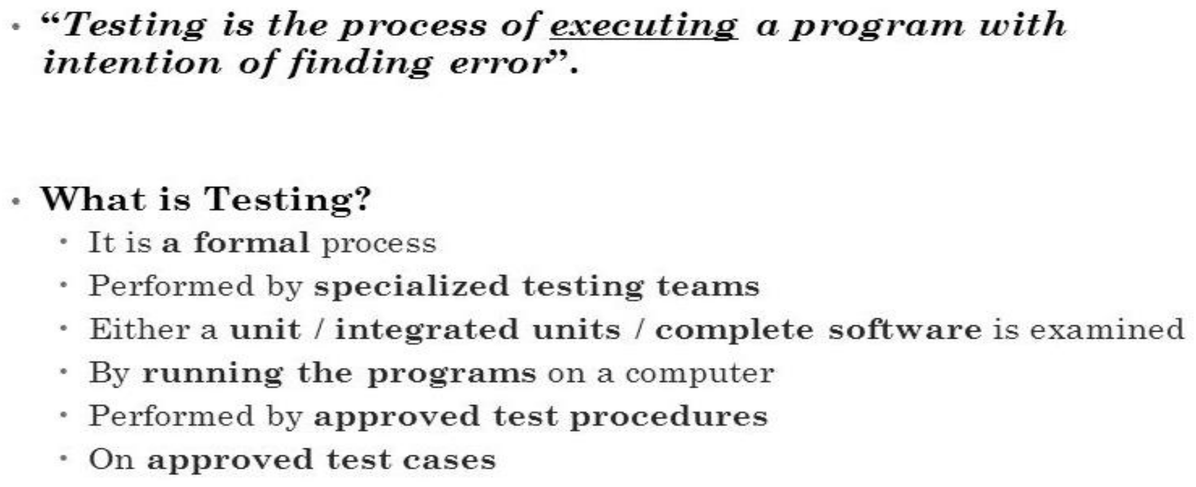
There still is a gap between academia and the demand of industries.

In the early days of software development, software testing was considered only a debugging process for removing errors after the development of software.

By 1970, the term ‘software engineering’ was in common use. But software testing was just a beginning at that time. In 1978, G. J. Myers realized the need to discuss the techniques of software testing in a separate subject. He wrote the book “The Art of Software Testing” which is a classic work on software testing.

Myers discussed the psychology of testing and emphasized that testing should be done with a mindset of finding errors and not to demonstrate that errors are not present.

What is testing?



- ***“Testing is the process of executing a program with intention of finding error”.***

- **What is Testing?**
  - It is **a formal** process
  - Performed by **specialized testing teams**
  - Either a **unit / integrated units / complete software** is examined
  - By **running the programs** on a computer
  - Performed by **approved test procedures**
  - On **approved test cases**

Definition of Software Testing:

Testing is the process of demonstrating that there are no errors.

Testing is the process of executing a program with the intent of finding errors. [Myers]

Testing can show the presence of bugs but never their absence. [W. Dijkstra]

Testing is a support function that helps developers look good by finding their mistakes before anyone else does. [James Bach]

Testing is concurrent lifecycle process of engineering, using and maintaining test-ware (testing artifacts) in order to measure and improve the quality of the software being tested. [Craig]

Software testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases.

Software testing can be divided into two steps:

1. Verification: it refers to the set of tasks that ensure that the software correctly implements a specific function. (Verification: “Are we building the product right?”)

2. Validation: it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. (Validation: “Are we building the right product?”)

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

Why Software Testing is Important?

Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

What is the need of Testing?

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.

- Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point, the store served coffee for free as they were unable to process the transaction.
- China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocents live.

What are the benefits of Software Testing?

Here are the benefits of using software testing:

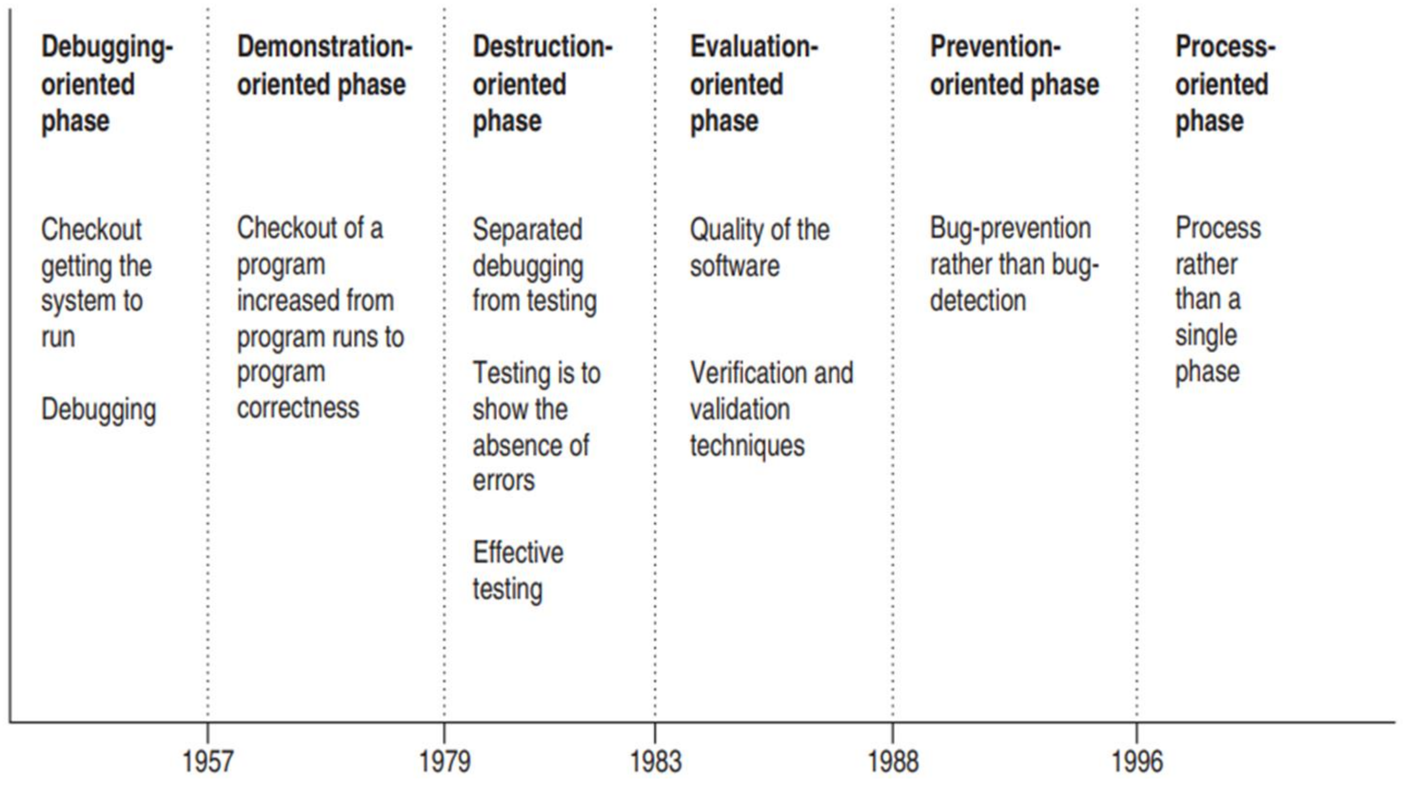
Cost-Effective: It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix.

Security: It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems earlier.

Product quality: It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.

Customer Satisfaction: The main aim of any product is to give satisfaction to their customers. UI/UX Testing ensures the best user experience.

EVOLUTION OF SOFTWARE TESTING


Debugging-oriented phase
Checkout getting the system to run
Debugging
Demonstration-oriented phase
Checkout of a program increased from program runs to program correctness
Destruction-oriented phase
Separated debugging from testing
Testing is to show the absence of errors
Effective testing
Evaluation-oriented phase
Quality of the software
Verification and validation techniques
Prevention-oriented phase
Bug-prevention rather than bug-detection
Process-oriented phase
Process rather than a single phase
1957
1979
1983
1988
1996

Debugging-oriented Phase (Before 1957)

This phase is the early period of testing. At that time, testing basics were unknown. Programs were written and then tested by the programmers until they were sure that all the bugs were removed. The term used for testing was checkout, focused on getting the system to run. Debugging was a more general term at that time and it was not distinguishable from software testing. Till 1956, there was no clear distinction between software development, testing, and debugging.

Demonstration-oriented Phase (1957–78)

The term 'debugging' continued in this phase. However, in 1957, Charles Baker pointed out that the purpose of checkout is not only to run the software but also to demonstrate the correctness according to the mentioned requirements. Thus, the scope of checkout of a program increased from program runs to program correctness. Moreover, the purpose of checkout was to show the absence of errors. There was no stress on the test case design. In this phase, there was a misconception that the software could be tested exhaustively.

Destruction-oriented Phase (1979–82)

This phase can be described as the revolutionary turning point in the his-tory of software testing. Myers changed the view of testing from 'testing is to show the absence of errors' to 'testing is to find more and more errors.' He separated debugging from testing and stressed on the valuable test cases if they explore more bugs. This phase has given importance to effective testing in comparison to exhaustive testing. The importance of early testing was also realized in this phase.

Evaluation-oriented Phase (1983–87)

With the concept of early testing, it was realized that if the bugs were identified at an early stage of development, it was cheaper to debug them as compared to the bugs found in implementation or post-implementation phases. This phase stresses on the quality of software products such that it can be evaluated at every stage of development. In fact, the early testing concept was established in the form of verification and validation activities which help in producing better quality software. In 1983, guidelines by the National Bureau of Standards were released to choose a set of verification and validation techniques and evaluate the software at each step of software development.

Prevention-oriented Phase (1988–95)

The evaluation model stressed on the concept of bug-prevention as compared to the earlier concept of bug-detection. With the idea of early detection of bugs in earlier phases, we can prevent the bugs in implementation or fur-ther phases. Beyond this, bugs can also be prevented in other projects with the experience gained in similar software projects. The prevention model includes test planning, test analysis, and test design activities playing a major role, while the evaluation model mainly relies on analysis and reviewing techniques other than testing.

Process-oriented Phase (1996 onwards)

In this phase, testing was established as a complete process rather than a single phase (performed after coding) in the software development life cycle (SDLC). The testing process starts as soon as the requirements for a project are specified and it runs parallel to SDLC. Moreover, the model for measuring the performance of a testing process has also been developed like CMM. The model for measuring the testing process is known as Testing Maturity Model (TMM). Thus, the emphasis in this phase is also on quantification of various parameters which decide the performance of a testing process.

The evolution of software testing was also discussed by Hung Q. Nguyen and Rob Pirozzi in a white paper, in three phases, namely Software Testing 1.0, Software Testing 2.0, and Software Testing 3.0.

Software Testing 1.0

In this phase, software testing was just considered a single phase to be per-formed after coding of the software in SDLC. No test organization was there. A few testing tools were present but their use was limited due to high cost. Management was not concerned with testing, as there was no quality goal.

Software Testing 2.0

In this phase, software testing gained importance in SDLC and the concept of early testing also started. Testing was evolving in the direction of planning the test resources. Many testing tools were also available in this phase.

Software Testing 3.0

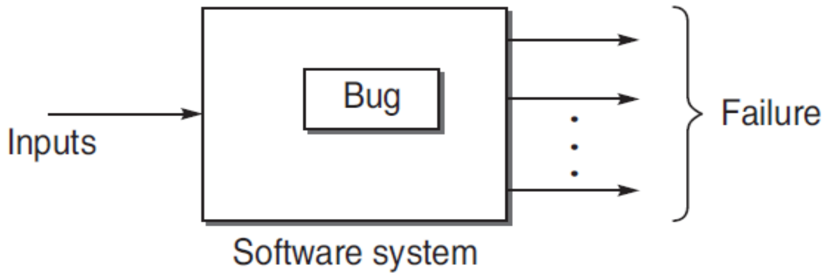
In this phase, software testing is being evolved in the form of a process which is based on strategic effort. It means that there should be a process which gives us a roadmap of the overall testing process. Moreover, it should be driven by quality goals so that all controlling and monitoring activities can be performed by the managers. Thus, the management is actively involved in this phase.

Failure, Fault, Defect and Bug:

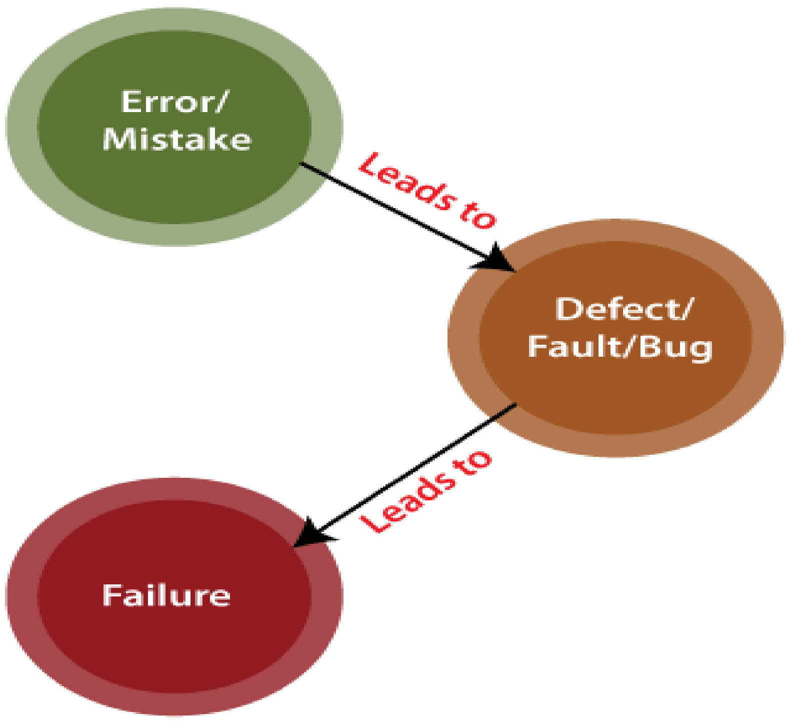
Failure: When the software is tested, failure is the first term being used. It means the inability of a system or component to perform a required function according to its specification. In other words, when results or behavior of the system under test are different as compared to specified expectations, then failure exists.

Fault/Defect/Bug: Failure is the term which is used to describe the problems in a system on the output side, as shown in Fig

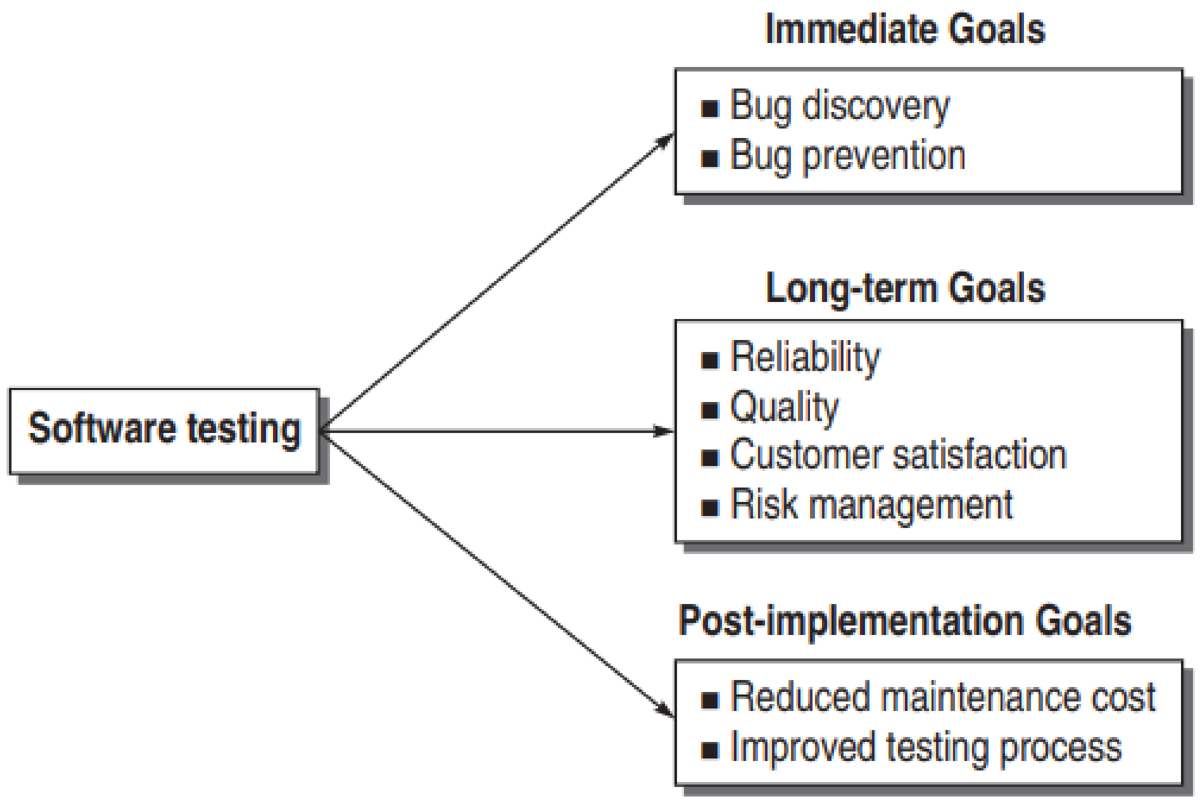
Fault is a condition that in actual causes a system to produce failure. Fault is synonymous with the words *defect or bug. Therefore, fault is the reason embedded in any phase of SDLC* and results in failures. It can be said that failures are manifestation of bugs. One failure may be due to one or more bugs and one bug may cause one or more failures. Thus, when a bug is executed, then failures are generated. But this is not always true. Some bugs are hidden in the sense that these are not executed, as they do not get the required conditions in the system. So, hidden bugs may not always produce failures. They may execute only in certain rare conditions.



- **Error:** Whenever a development team member makes a mistake in any phase of SDLC, errors are produced. It might be a typographical error, a misleading of a specification, a misunderstanding of what a subroutine does, and so on. Error is a very general term used for human mistakes. Thus, an error causes a bug and the bug in turn causes failures, as shown in Fig. 2.2.


Error/ Mistake
Leads to
Defect/ Fault/Bug
Leads to
Failure

Goals of Software Testing:



Short-term or immediate goals

These goals are the immediate results after performing testing.

These goals may be set in the individual phases of SDLC. Some of them are discussed below.

Bug discovery: The immediate goal of testing is to find errors at any stage of software development. More the bugs discovered at an early stage, better will be the success rate of software testing.

Bug prevention: It is the consequent action of bug discovery. From the behavior and interpretation of bugs discovered, everyone in the software development team gets to learn how to code safely such that the bugs discovered should not be repeated in later stages or future projects. Though errors cannot be prevented to zero, they can be minimized. In this sense, bug prevention is a superior goal of testing.

Long-term goals

These goals affect the product quality in the long run, when one cycle of the SDLC is over. Some of them are discussed here.

Quality Since software is also a product, its quality is primary from the users' point of view. Thorough testing ensures superior quality. Therefore, the first goal of understanding and performing the testing process is to enhance the quality of the software product. Though quality depends on various factors, such as correctness, integrity, efficiency, etc., **reliability** is the major factor to achieve quality. The software should be passed through a rigorous reliability analysis to attain high quality standards.

Reliability is a matter of confidence that the software will not fail, and this level of confidence increases with rigorous testing. The confidence in reliability, in turn, increases the quality, as shown in Fig. 2

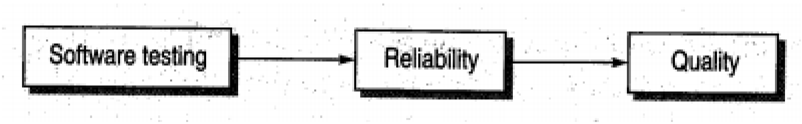


Figure 2: Testing Produces Reliability and Quality

Testing produces reliability and quality.

Customer satisfaction

From the users' perspective, the prime concern of testing is customer satisfaction only. If we want the customer to be satisfied with the software product, then testing should be complete and thorough. Testing should be complete in the sense that it must satisfy the user for all the specified requirements mentioned in the user manual, as well as for the unspecified requirements which are otherwise understood. A complete testing process achieves reliability, reliability enhances the quality, and quality in turn, in-creases the customer satisfaction, as shown in Fig.

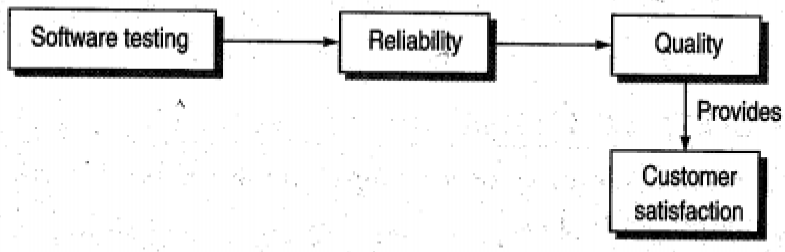


Figure 3: Quality Leads to Customer Satisfaction

Risk management

Risk is the probability that undesirable events will occur in a system. These undesirable events will prevent the organization from successfully implementing its business initiatives. Thus, risk is basically concerned with the business perspective of an organization.

Risks must be controlled to manage them with ease. Software testing may act as a control, which can help in eliminating or minimizing risks (see Fig. 1.5).

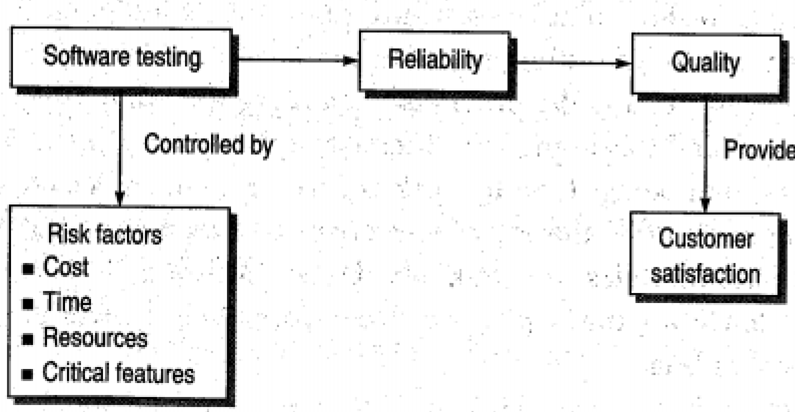


Figure 4: Testing Controlled by Risk Factors

Thus, managers depend on software testing to assist them in controlling their business goals. The purpose of software testing as a control is to provide information to management so that they can better react to risk situations.

For example, testing may indicate that the software being developed cannot be delivered on time, or there is a probability that high priority bugs will not be resolved by the specified time.

Testing Principles:

- **Effective testing, not exhaustive testing.**

  All possible combinations of tests become so large that it is impractical to test them all. So considering the domain of testing as infinite, exhaustive testing is not possible. Therefore, the tester's approach should be based on effective testing to adequately cover program logic and all conditions in the component level design.

- **Testing is not a single phase performed in SDLC.**

  Testing is not just an activity performed after the coding in SDLC. As discussed, the testing phase after coding is just a part of the whole testing process. Testing process starts as soon as the specifications for the system are prepared and it continues till the release of the product.

- **Destructive approach for constructive testing.**

  Testers must have the psychology that bugs are always present in the program and they must think about the technique of how to uncover them (this is their art of creativity). This psychology of being always suspicious about bugs is a negative/destructive approach. Successful testing is to discover more and more bugs, and not to show that the system does not contain any bugs.

- **Early testing is the best policy.**

  When is the right time to start the testing process? after coding, rather it starts as soon as requirement specifications are prepared. Moreover, the cost of bugs can be reduced tenfold, as bugs are harder to detect in later stages if they go undetected. Thus, the policy in testing is to start as early as possible.

- **Probability of existence of an error in a section of a program is proportional to the number of errors already found in that section.**

  Suppose the history of a software is that you found 50 errors in Module X, 12 in Module Y, and 3 in Module Z. The software was debugged but after a span of time, we fi nd some errors again and the software is given to a tester for testing. Where should the tester concentrate to fi nd the bugs? This principle says that the tester should start with Module X which has the history of maximum errors. Another way of stating it is that errors seem to come in clusters.

- **Testing strategy should start at the smallest module level and expand towards the whole program.**

  This principle supports the idea of incremental testing. Testing must begin at the unit or module level, gradually progressing towards integrated modules and finally the whole system. Testing cannot be performed directly on the whole system. It must start with the individual modules and slowly integrate the modules and test them. After this, the whole system should be tested

- **Testing should also be performed by an independent team.**

  When programmers develop the software, they test it at their individual modules. However, these programmers are not good testers of their own software. They are basically constructors of the software, but testing needs a destructive approach. Programmers always think positively that their code does not contain bugs. Therefore, it is always recommended to have the software tested by an independent testing team. Testers associated with the same project can also help in this direction, but this is not effective. For effective testing, the software may also be sent outside the organization for testing.

- **Everything must be recorded in software testing**

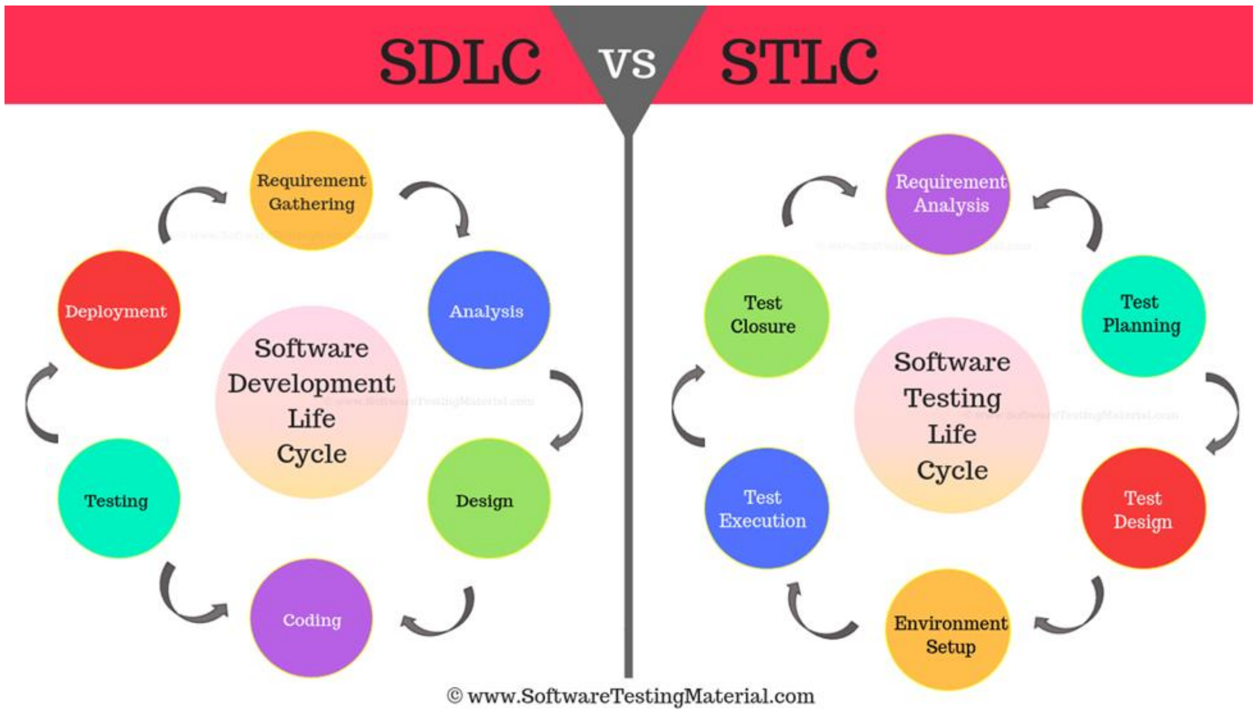
  It demands that every detail be recorded and documented. We must have the record of every test case run and the bugs reported. Even the inputs provided during testing and the corresponding outputs are to be recorded. Executing the test cases in a recorded and documented way can greatly help while observing the bugs. Moreover, observations can be a lesson for other projects. So the experience with the test cases in one project can be helpful in other projects.

- **Invalid inputs and unexpected behaviour have a high probability of finding an error**

Whenever the software is tested, we test for valid inputs and for the functionality that the software is supposed to do. But thinking in a negative way, we must test the software with invalid inputs and the behaviour which is not expected in general. This is also a part of effective testing.

- **Testers must participate in specification and design reviews**

  Testers' role is not only to get the software and documents and test them. If they are not participating in other reviews like specification and design, it may be possible that either some specifications are not tested or some test cases are built for no specifications.



SOFTWARE TESTING LIFE CYCLE (STLC)

The testing process divided into a well-defined sequence of steps is termed as software testing life cycle (STLC). The major contribution of STLC is to involve the testers at early stages of development. This has a significant benefit in the project schedule and cost. The STLC also helps the management in measuring specific milestones.

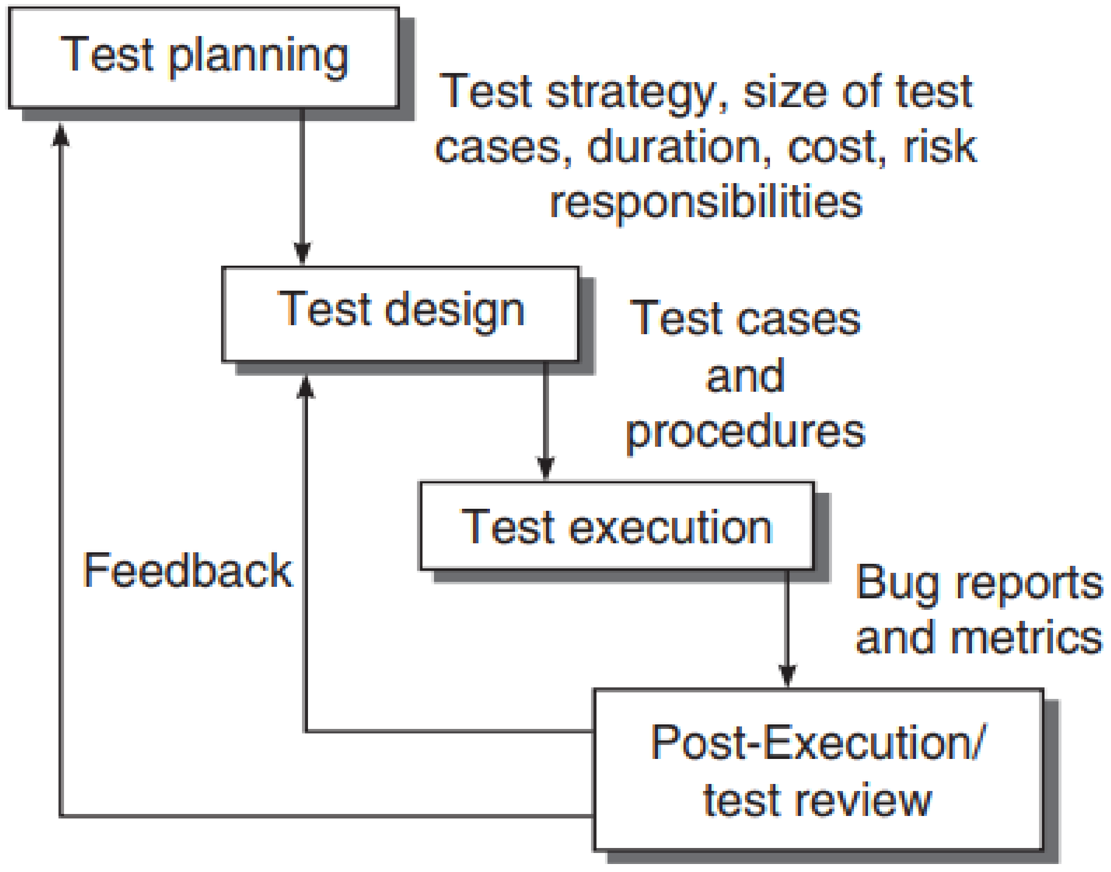

Test planning
Test strategy, size of test cases, duration, cost, risk responsibilities
Test design
Test cases and procedures
Test execution
Feedback
Bug reports and metrics
Post-Execution/ test review

Figure 2.8 Software testing life cycle

1. **Test Planning:**
   The goal of test planning is to take into account the important issues of testing strategy, viz. resources, schedules, responsibilities, risks, and priorities, as a roadmap. Test planning issues are in tune with the overall project planning. Broadly, following are the activities during test planning:

- Defining the test strategy.
- Estimate the number of test cases, their duration, and cost.
- Plan the resources like the manpower to test, tools required, documents required.
- Identifying areas of risks.
- Defining the test completion criteria.
- Identification of methodologies, techniques, and tools for various test cases.
- Identifying reporting procedures, bug classification, databases for testing, bug severity levels, and project metrics.

Based on the planning issues as discussed above, analysis is done for various testing activities. The major output of test planning is the test plan document. Test plans are developed for each level of testing. After analyzing the issues, the following activities are performed:

- Develop a test case format.
- Develop test case plans according to every phase of SDLC.
- Identify test cases to be automated (if applicable).
- Prioritize the test cases according to their importance and criticality.
- Define areas of stress and performance testing.
- Plan the test cycles required for regression testing.

2.Test Design:
One of the major activity in testing is the design of test cases. The test design is an important phase after test planning. It includes following activity:
Determine test objectives and their prioritization.
Preparing list of items to be tested.
Mapping items to test cases.
(There is only one rule in designing test cases: Cover all features, but do not make too many test cases)
Selection of test case design techniques. [Blackbox / Whitebox testing]
Creating test cases and test data. [Input required and Output Expected]

Setting up test environment and supporting tools. [H/W, testers, interfaces, OS etc.]
Creating test procedure specification. [Description of how the test cases will run. It is a sequence of steps]

Attributes of a good test cases:

1. A good test case is one that has been designed keeping in view the criticality and high-risk requirements in order to place a greater priority upon, and provide added depth for testing the most important function.
2. A good case should be designed such that there is a high probability of finding an error.
3. Test cases should not be overlapped or redundant.
4. It is possible to combine a series of tests into one test case.
5. A successful test case is one that has the highest probability of detecting an as – yet –undiscovered error.

3. Test Execution:

- In this phase, all test cases are executed including verification and validation.
- Verification test cases start at the end of each face of SDLC. Validation test cases start after the completion of a module.
- It is the decision of the test team to opt for automation or manual execution.
- Test results are documented in the incident reports, test logs, testing status, and test summary reports.
- ***Test incident*** is the report about any unexpected event during testing, which needs further investigation to resolve the bug.
- ***Test log*** is a record of the testing events that take place during the test.
- ***Test summary report*** is an evaluation report describing summary of all the tests and is prepared when the testing is over.

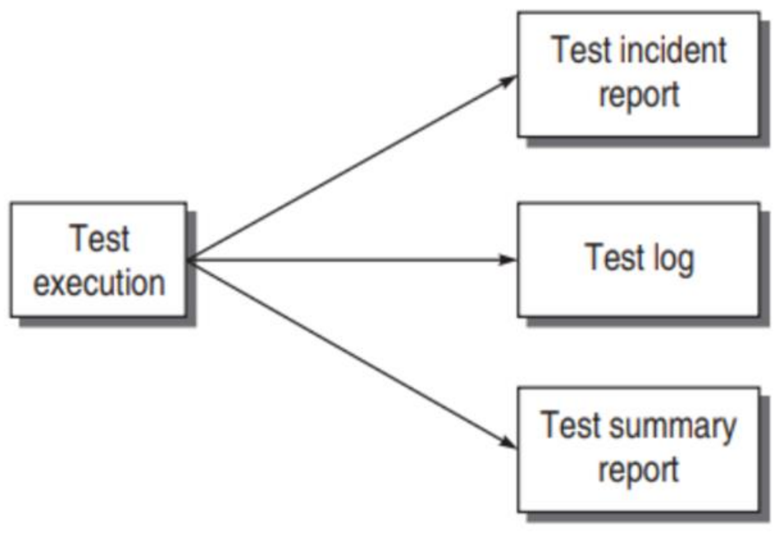


Figure 2.10 Documents in test execution

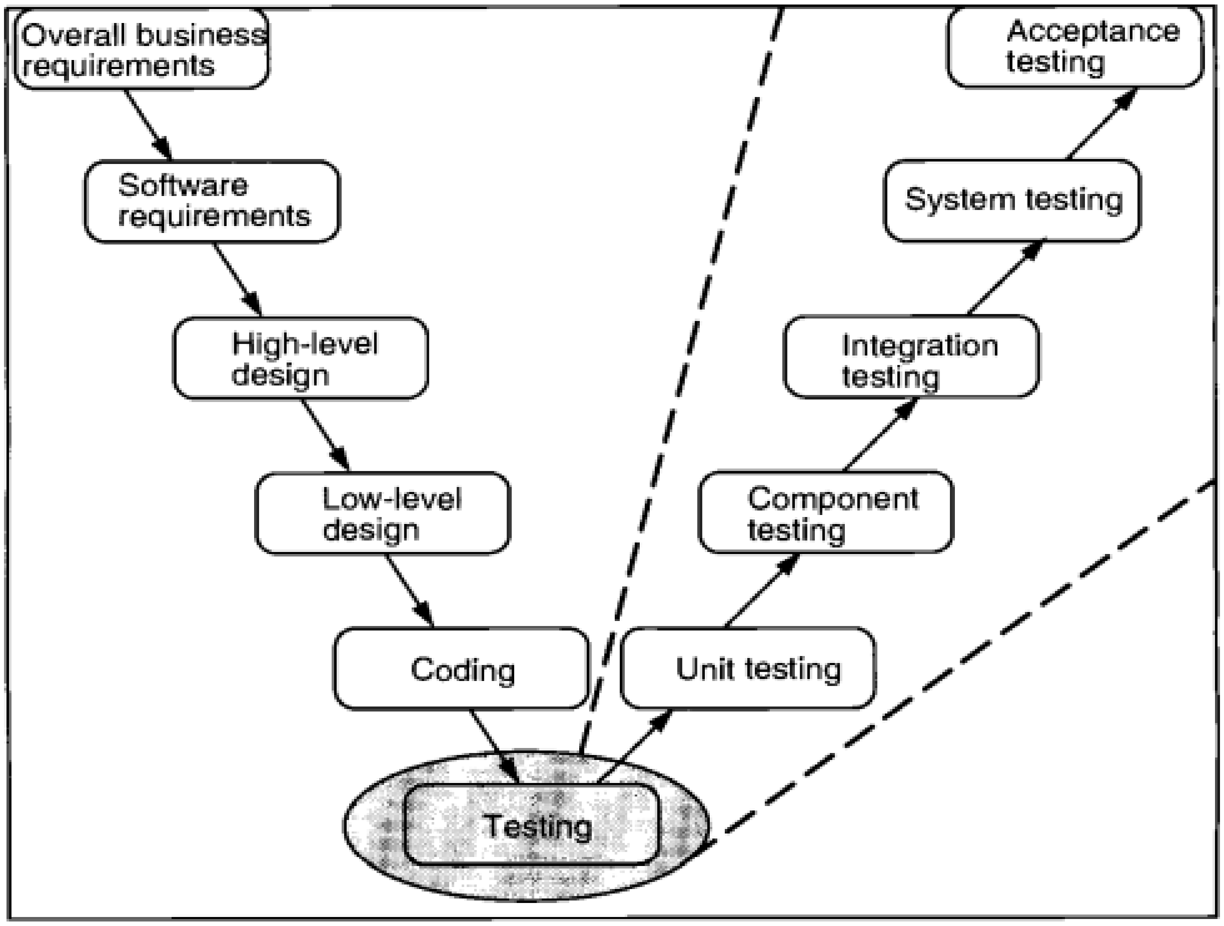
4.Post-Execution/Test Review:

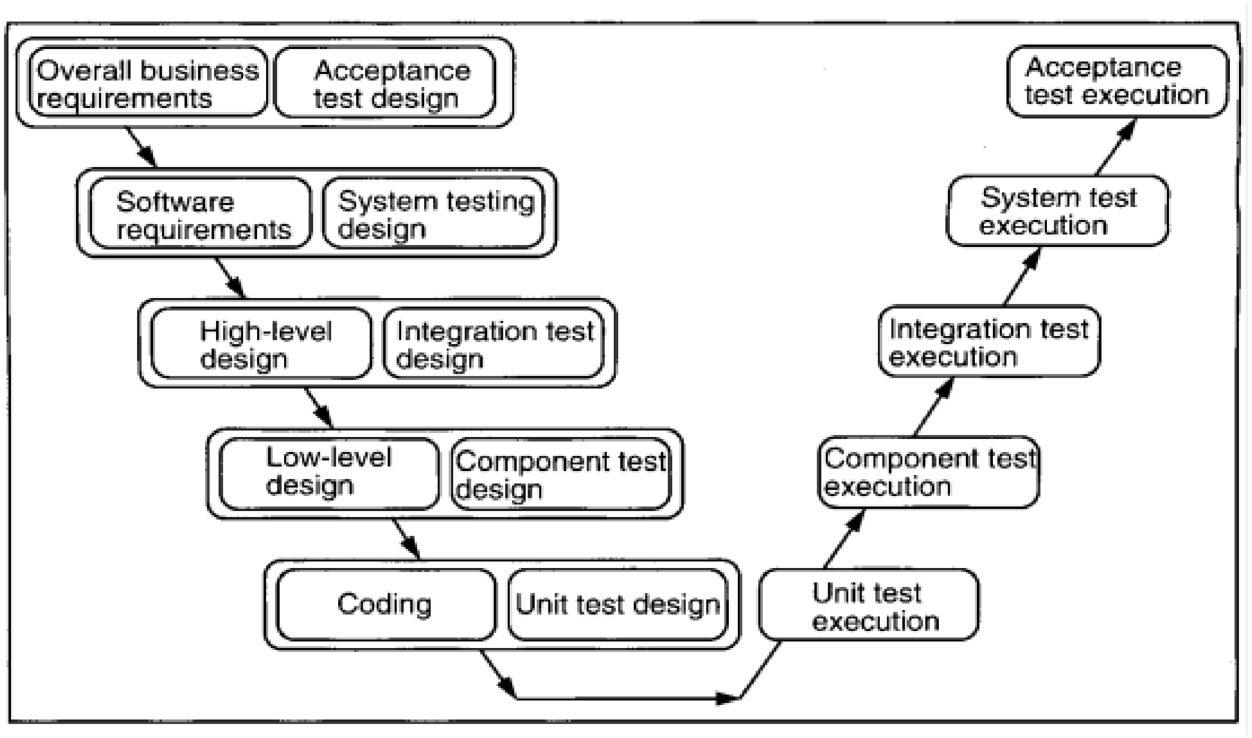
Understanding the bug: The developer analyses the bug reported and builds an understanding of its whereabouts.

Reproducing the bug :Next, he confirms the bug by reproducing the bug and the failure that exists. This is necessary to cross-check failures. However, some bugs are not reproducible which increases the problems of developers.

Analyzing the nature and cause of the bug: After examining the failures of the bug, the developer starts debugging its symptoms and tracks back to the actual location of the error in the design.

V-testing:


Overall business requirements
Software requirements
High-level design
Low-level design
Coding
Testing
Unit testing
Component testing
Integration testing
System testing
Acceptance testing



Testing Life Cycle Model:

- Verification and Validation V&V(are the building blocks of a testing process. Life cycle involves continues testing of the system during the development process.
- In V-testing concept, as the development team attempts to implement the software, the testing team concurrently starts checking the software.
- When the project starts, both the system development and the system test process begin.
- The team that is developing the system begins the system development process and the team that is conducting the system test begins planning the system test process.

V-Testing Life Cycle Model:

- In V-Testing concept, as the development team attempts to implement the software, the testing team concurrently starts checking the software.
- When the project starts, both the system development and the system test process begin.
- The team that is developing the system begins the system development process and the team that is conducting the system test begins planning the system test process as shown in the figure given below.
- The V&V process involves:

1. Verification of every step of SDLC and,
2. Validation of the verified system at the end

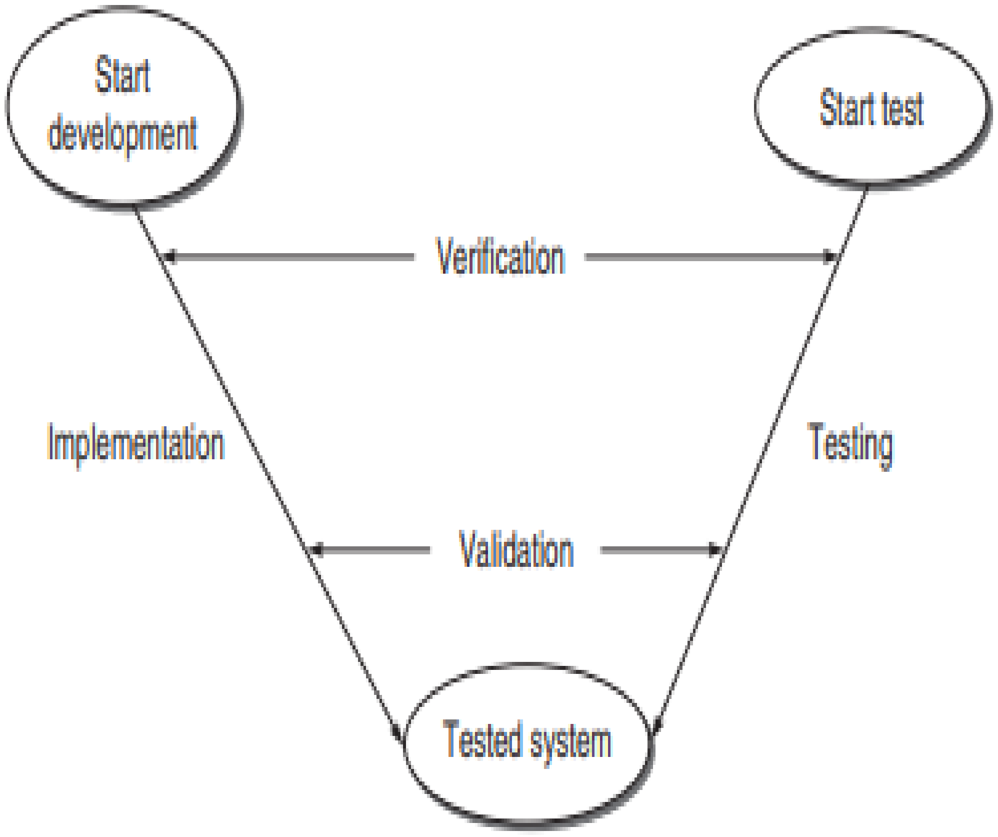


Figure 2.12 V-testing model

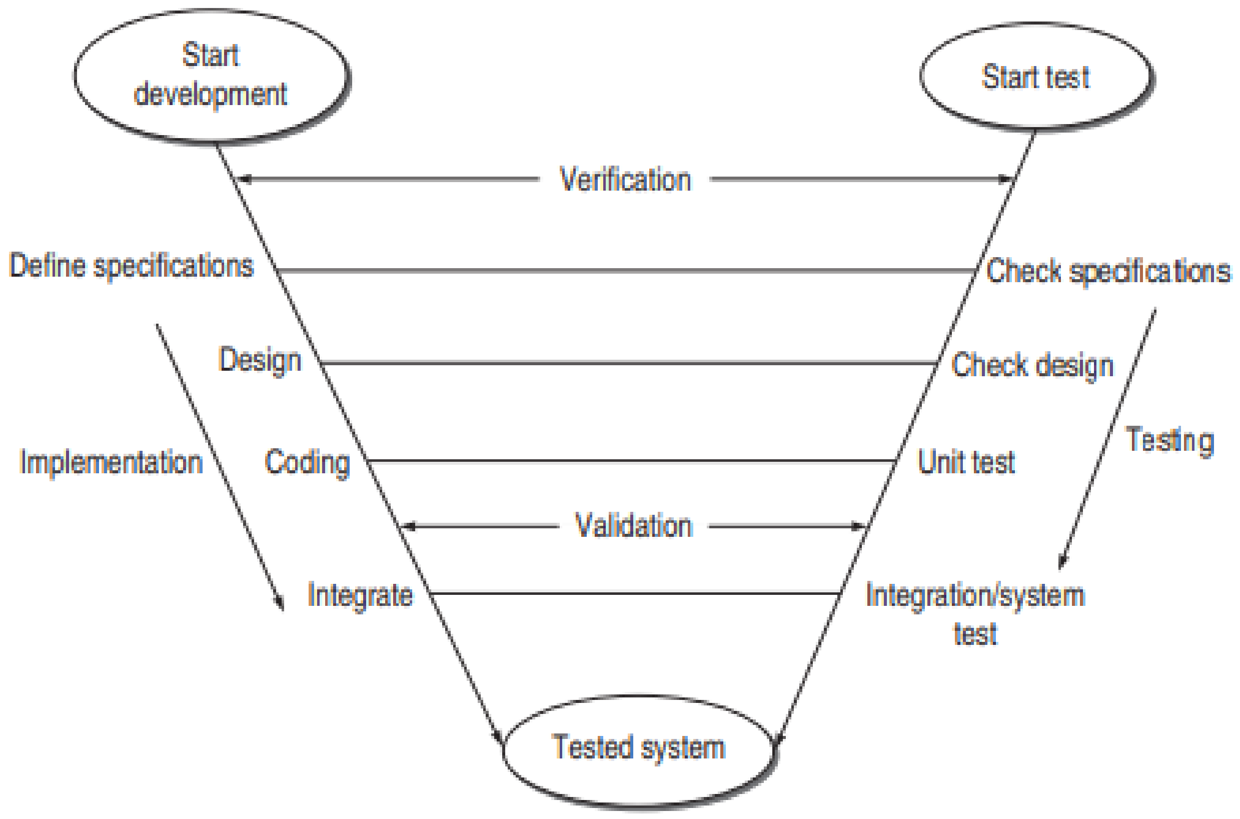


Figure 2.13 Expanded V-testing model

<u>VERIFICATION:</u>

- Verification is a set of activities that ensures correct implementation of specific functions in a software.
- <u>What is the need of verification?</u>

1. If the verification is not performed at early stages, there is always a chance of mismatch between the required product and the delivered product.
2. Verification exposes more errors.
3. Early verification decreases the cost of fixing bugs.
4. Early verification enhances the quality of software.

- <u>What is the goal of verification?</u>

1. Everything must be verified
2. Results of verification may not be binary

3. Even implicit quantities must be verified.

VALIDATION ACTIVITIES:

Validation involves the following three activities, which are also known as the three levels of validation testing.

Unit Testing: It is major validation efforts performed on the smallest module of the system. If avoided, many bugs becomes latent bugs and are released in the final product. Unit testing is a basic level of testing which cannot be overlooked and confirms the behavior of a single module according to its functional specification.

Integration Testing: It is a validation technique which combines all unit-tested modules and performs a test on their aggregation. Integration testing is needed because of interfacing. Unit modules are not independent, and are related to each other by interfacing between unit modules.

System Testing: This testing level focuses on testing the entire integrated system. It incorporates many types of testing, as the full system can have various users in different environments.

Software Testing Techniques:

STATIC TESTING:

It is a technique for assessing the structural characteristics of source code, design specifications or any notational representation that conforms to well-defined syntactic rules .It is called as static because we never execute the code in this technique. For example, the structure of code is examined by the teams but the code is not executed.

DYNAMIC TESTING:

All the methods that execute the code to test a software are known as dynamic testing techniques. In this technique, the code is run on a number of inputs provided by the user and the corresponding results are checked. This type of testing is further divided into two parts: (A) Black-box testing and (B) White-box testing.

[A] Black-box testing: This technique takes care of the inputs given to a system and the output is received after processing in the system. What is being processed in the system? How does the system perform these operations? Black-box testing is not concerned with these questions. It checks the functionality of the system only.

That is why the term black-box is used. It is also known as functional testing. It is used for system testing under validation.

[B] White-box testing: This technique complements black-box testing. Here, the system is not a black box. Every design feature and its corresponding code is checked logically with every possible path execution. So, it takes care of the structural paths instead of just outputs. It is also known as structural testing and is used for unit testing under verification.

Difference between Static & Dynamic Testing:

Static Testing	Dynamic Testing
In Static testing code is not executed	In Dynamic testing code is always executed
It means to review and to examine the software	It means running and then testing the software
Cost of the product is reduced as it always starts early in software testing life cycle	This testing increases the cost of project as it is started late
Static testing is done as phase verification	Dynamic testing is done as phase validation
Static testing techniques are formal technique review, inspection, walkthrough	In dynamic testing various techniques like white box and black box are used
It does not take time as its purpose is to check the item	It takes time because it executes the software code and we need to run the test cases.

So, to summarize we can say that:

1. Software testing is required to check the reliability of the software
2. Software testing ensures that the system is free from any bug that can cause any kind of failure
3. Software testing ensures that the product is in line with the requirement of the client
4. It is required to make sure that the final product is user friendly
5. At the end software is developed by a team of human developers all having different viewpoints and approach. Even the smartest person has the tendency to make an error. It is not possible to create software with zero defects without incorporating software testing in the development cycle.
6. No matter how well the software design looks on paper, once the development starts and you start testing the product you will definitely find lots of defects in the design.

You cannot achieve software quality without software testing. Even if testers are not involved in actual coding they should work closely with developers to improve the quality of the code. For best results it is important that software testing and coding should go hand in hand.

ASSIGNMENT-1:

1. What is software testing? List and explain goals of software testing.
2. What are the principles of software testing? Explain them.
3. Explain Software Testing Life Cycle (STLC).
4. What is V-testing life cycle model? How is it used? Explain in detail.
5. State the difference between static and dynamic testing?

MCQs:

No. of objectives
1.1 to 1.6
2.1 to 2.14 (All)

Submission on :17/01/2023