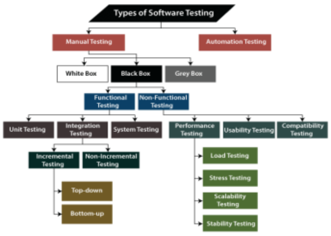
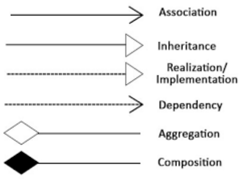
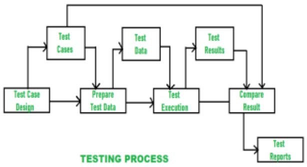


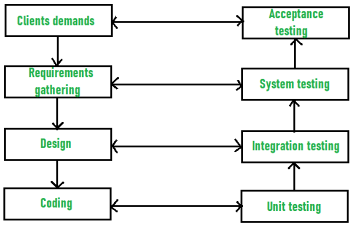




TESTING PROCESS

Software Design and Testing

Introduction to Software Testing



LEVELS OF TESTING

Outline

- Introduction and Evolution of s/w Testing,
- Definition and Goals of Testing,
- Effective and Exhaustive Testing,
- Software Testing Life Cycle (STLC),
- Testing Terminology and Methodology

We trust in GOD

Everything Else we TEST

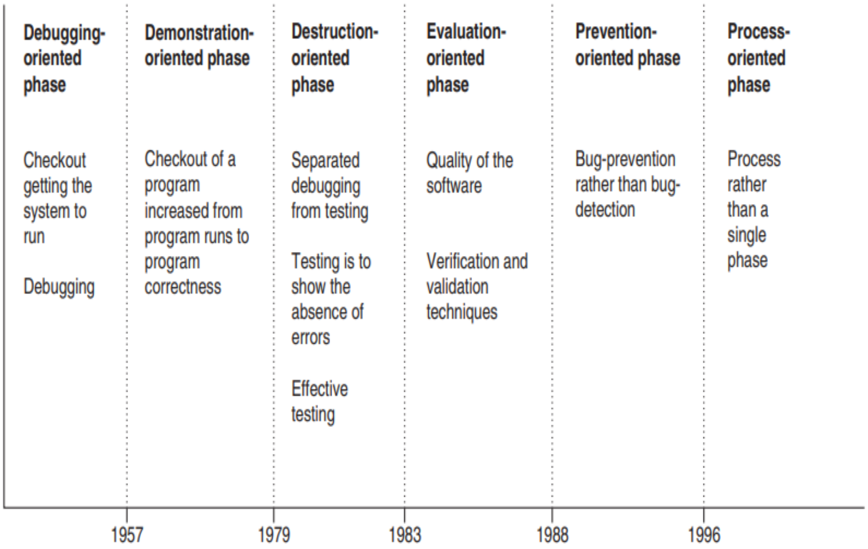
Software Testing

- 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 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.

Software Testing

- 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.
- ***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.
- By 1980, software professionals and organizations started emphasizing on quality.
- Organizations realized the importance of having quality assurance teams to take care of all testing activities for the project right from the beginning.
- In the 1990s, testing tools finally came into their own.

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

Evolution Of Software Testing

- 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.
- These three phases discuss the evolution in the earlier phases that we described. According to this classification, the current state-of-practice is Software Testing 3.0.
- **Software Testing 1.0**
  - In this phase, software testing was just considered a single phase to be performed 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.

Evolution Of Software Testing

- **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.

Evolution Of Software Testing

- **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.

Software Testing

Testing is the **process** of exercising a program with the specific **intent of finding errors** prior to delivery to the end user.

Don't view testing as a **"safety net"** that will catch all errors that occurred because of weak software engineering practice.

Effective Vs. Exhaustive Software Testing

- Exhaustive or complete software testing means that every statement in the program and every possible path combination with every possible combination of data must be executed.
- But soon, we will realize that exhaustive testing is out of scope. That is why the questions arise:
  - When are we done with testing? or
  - How do we know that we have tested enough?
- There may be many answers for these questions with respect to time, cost, customer, quality, etc.
- **Exhaustive or complete testing is not possible.**
- Therefore, we should concentrate on effective testing which emphasizes efficient techniques to test the software so that important features will be tested within the constrained resources.

Effective Vs. Exhaustive Software Testing

- Exhaustive Testing Examples
  - Suppose, a calculator accepts integers ranging from -1000000 to +1000000. The tester will try to input all the possible numbers in between the range. No matter how much time is consumed, the tester will test to its maximum ability to leave no scope. This will be exhaustive testing in this case.
  - Another example can be, a device can be accessed only through a code with combinations of numbers in pairs of 3. The maximum possible input combination in this case, will be 9 numbers from 0-9(no repetitions allowed) and 3 input permutations. So the number of test cases will look like- 9*9*9 which is possible to execute. Here we have achieved what we call exhaustive testing.

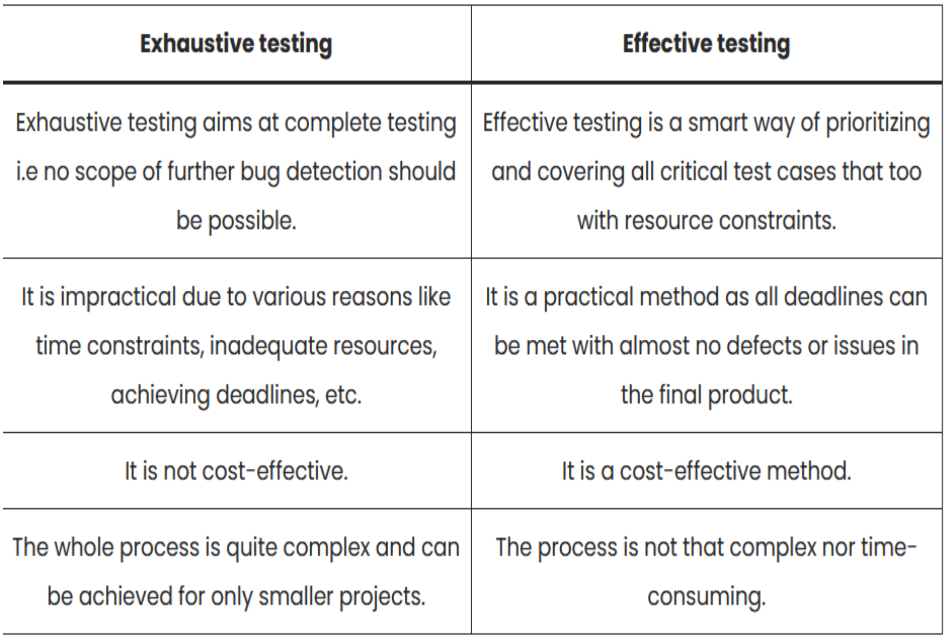
Effective Vs. Exhaustive Software Testing

- Why don’t we test everything ?

System has 20 screens
Average 4 menus / screen
Average 3 options / menu
Average of 10 fields / screen
2 types of input per field
Around 100 possible values
Approximate total for exhaustive testing
20 x 4 x 3 x 10 x 2 x 100 = 480,000 tests
Test length = 1 sec then test duration = 17.7 days
Test length = 10 sec then test duration = 34 weeks
Test length = 1 min then test duration = 4 years
Test length = 10 mins then test duration = 40 years!

It is not a matter of time. But time is money (salary is taken by hour. So second is valuable for software houses)

Effective Vs. Exhaustive Software Testing



Exhaustive testing	Effective testing
Exhaustive testing aims at complete testing i.e no scope of further bug detection should be possible.	Effective testing is a smart way of prioritizing and covering all critical test cases that too with resource constraints.
It is impractical due to various reasons like time constraints, inadequate resources, achieving deadlines, etc.	It is a practical method as all deadlines can be met with almost no defects or issues in the final product.
It is not cost-effective.	It is a cost-effective method.
The whole process is quite complex and can be achieved for only smaller projects.	The process is not that complex nor time-consuming.

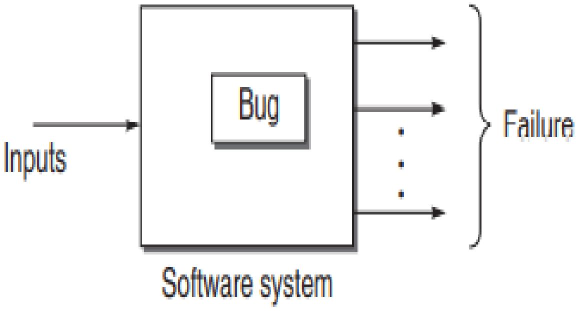
Software Testing Terminology and Methodology

- We tend to use the terms 'error', 'bug', 'defect', 'fault', 'failure', etc. interchangeably. But they don't mean the same.
- **Bugs** in general, are more important during software testing.
    - All the bugs do not have the same level of severity.
    - Some bugs are less important, as they are not going to affect the product badly.
    - On the other hand, bugs with high severity level will affect the product such that the product may not be released.
    - Therefore, bugs need to be classified according to their severity.
    - A bug has a complete cycle from its initiation to death.

Software Testing Terminology and Methodology

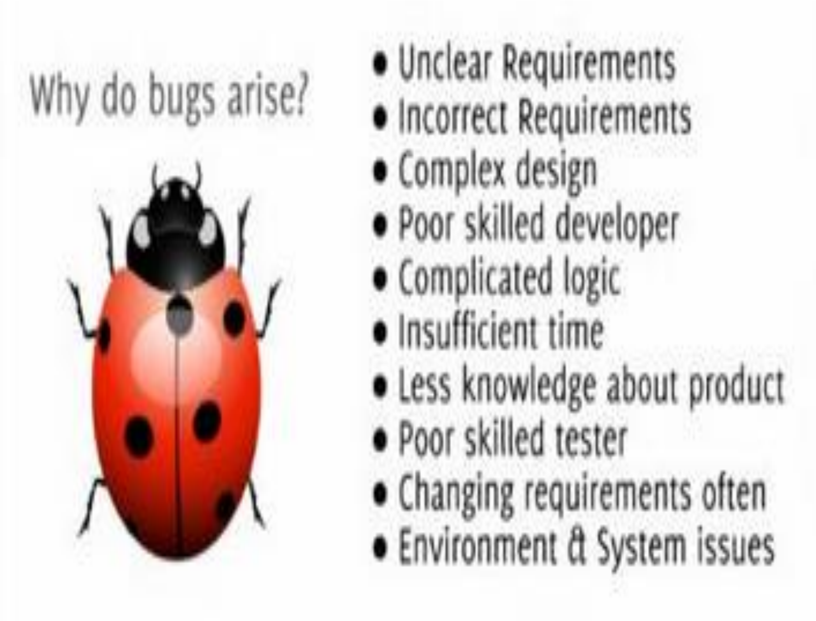
- **Failure :** When the software is tested, failure is the first term being used.
  - It means the inability of a system or component to perform a required function according to its specification.
  - In other words, when results or behaviour of the system under test are different as compared to specified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

Software Testing Terminology and Methodology



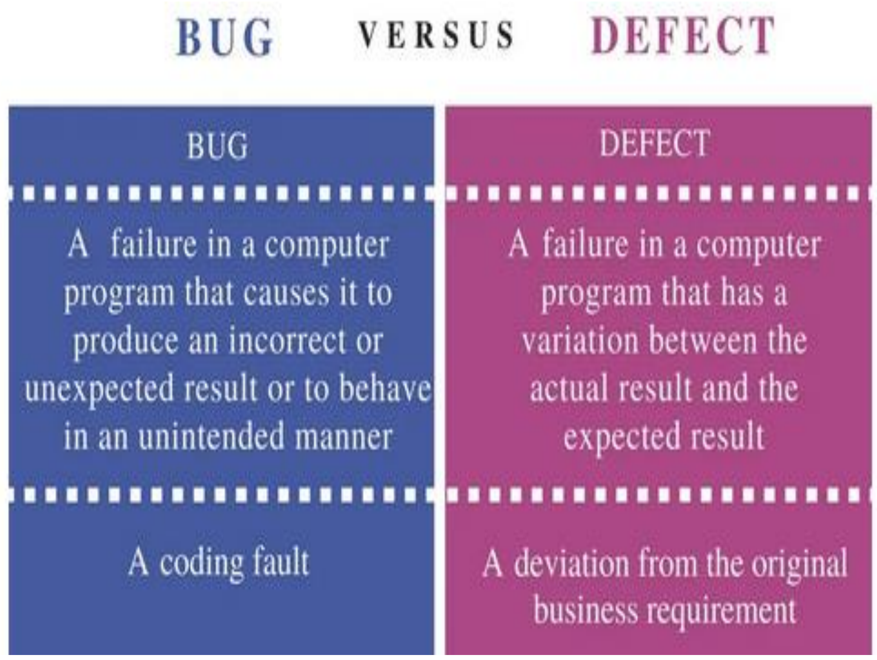
Software Testing Terminology and Methodology

- **Fault/Defect/Bug :**
  - 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.



Why do bugs arise?

- Unclear Requirements
- Incorrect Requirements
- Complex design
- Poor skilled developer
- Complicated logic
- Insufficient time
- Less knowledge about product
- Poor skilled tester
- Changing requirements often
- Environment & System issues

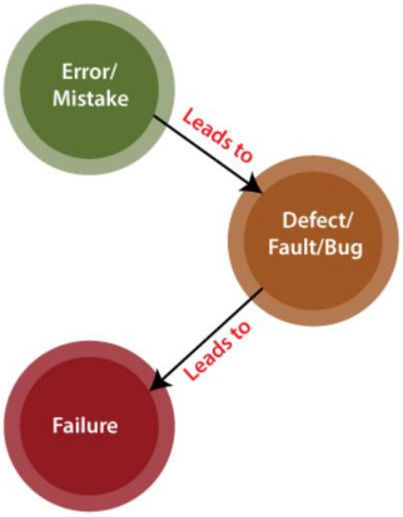


BUG VERSUS DEFECT

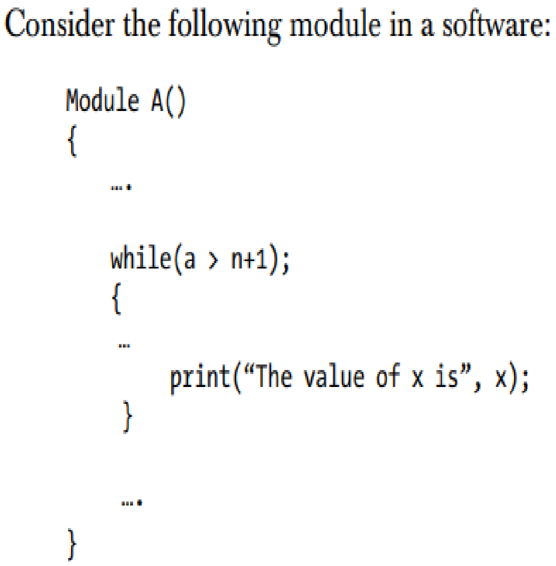
BUG	DEFECT
A failure in a computer program that causes it to produce an incorrect or unexpected result or to behave in an unintended manner	A failure in a computer program that has a variation between the actual result and the expected result
A coding fault	A deviation from the original business requirement

Software Testing Terminology and Methodology

- **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.



Software Testing Terminology and Methodology



Consider the following module in a software:

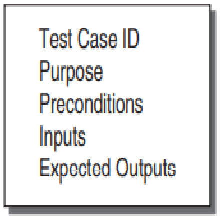
```
Module A()
{
    ….

    while(a > n+1);
    {
    …
         print("The value of x is", x);
     }

     ….
}
```

Software Testing Terminology and Methodology

- Suppose the module shown above is expected to print the value of x which is critical for the use of software.
- But when this module will be executed, the value of x will not be printed. It is a failure of the program.
- When we try to look for the reason of the failure, we find that in Module A(), the while loop is not being executed.
- A condition is preventing the body of while loop to be executed.
- This is known as bug/defect/fault.
- On close observation, we find a semicolon being misplaced after the while loop which is not its correct syntax and it is not allowing the loop to execute.
- This mistake is known as an error.

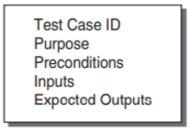
Software Testing Terminology and Methodology

- **Test case :** Test case is a well-documented procedure designed to test the functionality of a feature in the system.
  - A test case has an identity and is associated with a program behaviour.
  - The primary purpose of designing a test case is to find errors in the system.
  - For designing the test case, it needs to provide a set of inputs and its corresponding expected outputs.



Test Case ID
Purpose
Preconditions
Inputs
Expected Outputs

Software Testing Terminology and Methodology



Test Case ID
Purpose
Preconditions
Inputs
Expected Outputs

- Test case ID : is the identification number given to each test case.
- Purpose : defines why the case is being designed.
- Preconditions : for running the inputs in a system can be defined, if required, in a test case.
- Inputs : should not be hypothetical.
  - Actual inputs must be provided, instead of general inputs.
  - For example, if two integer numbers have to be provided as input, then specifically mention them as 23 and 56.
- Expected outputs : are the outputs which should be produced when there is no failure.
  - Test cases are designed based on the chosen testing techniques. They provide inputs when the system is executed.
  - After execution, observed results are compared with expected outputs mentioned in the test case.

Software Testing Terminology and Methodology

- **Testware :** The documents created during testing activities are known as testware.
  - Taking the analogy from software and hardware as a product, testware are the documents that a test engineer produces.
  - It may include test plans, test specifications, test case design, test reports, etc.
  - Testware documents should also be managed and updated like a software product.
- **Incident:** When a failure occurs, it may or may not be readily apparent to the user.
  - An incident is the symptom(s) associated with a failure that alerts the user about the occurrence of a failure.

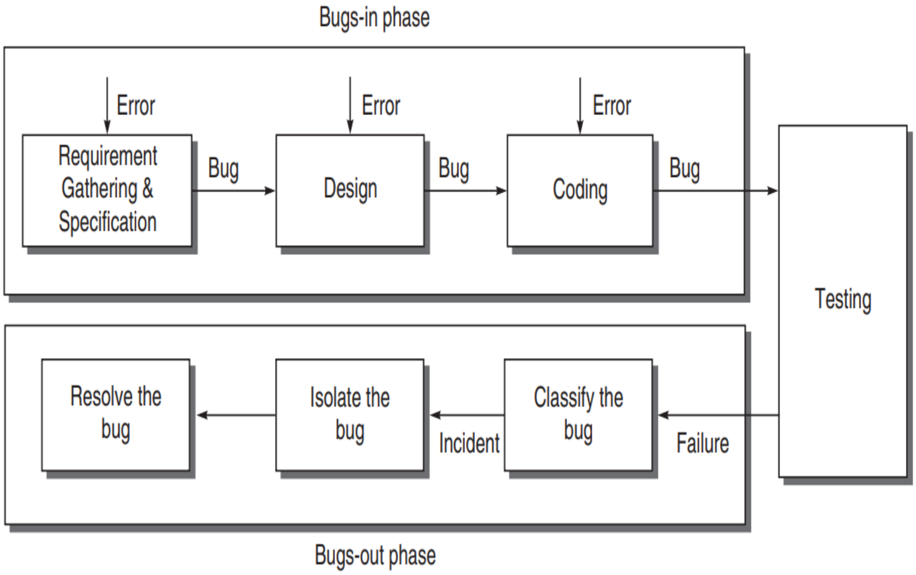
Software Testing Terminology and Methodology

- **Test oracle** : An oracle is the means to judge the success or failure of a test, i.e. to judge the correctness of the system for some test.
  - The simplest oracle is comparing actual results with expected results by hand.
  - This can be very time consuming, so automated oracles are sought.

Bug Life Cycle

- **Difference in Bug, Defect and Error**
  - Bug is an error found BEFORE the application goes into production (By Tester).
  - Defect is an error found AFTER application goes into production. (By Customer)
  - Anything that is not confirming to expected behavior and it may be at any stage is Error.
  - An error can be a defect or Bug and deviated from Customer Requirement.
- **What is Bug Life Cycle?** The duration between the first time bug is found ("New") and closed successfully ("Closed"), rejected, postponed or deferred is called as "Bug Life Cycle".
- The whole life cycle of a bug can be classified into two phases:
  - (i) bugs-in phase and (ii) bugs-out phase

Bug Life Cycle



Bug-In Phase

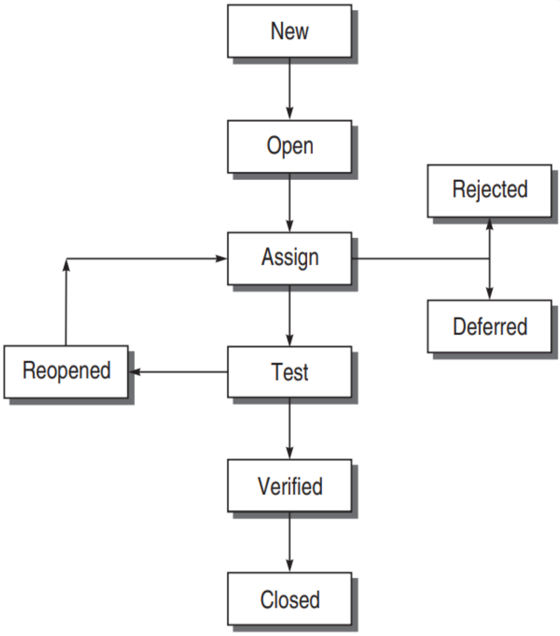
- This phase is in where the errors and bugs are introduced in the software.
- Whenever people commit mistake, it creates errors on a specific location of the software and consequently, when this error goes unnoticed, it causes some conditions to fail, leading to a bug in software.
- This bug is carried out in subsequent phases of SDLC, if not detected.
- A phase may have its own errors as well as bugs received from the previous phase.

Bug-Out Phase

- If failures occurs while testing a software product, we come to conclusion that it is affected by bugs.
- In bug-out phase we observe failures the following activities are carried out to free from bugs.
  - **Bug classification:** Classify bug whether it can be critical or catastrophic in nature or it may have no adverse effect on the output behavior.
  - **Bug isolation:** bug isolation is the activity by which we locate the module in which the bug appears. Incidents observed in failures help in this activity.
  - **Bug resolution:** Once we have isolate the bug, we back-trace the design to pinpoint the location of the error. A bug is resolved when we have found exact location of its occurrence.

Stages of Bug

- New
- Open
- Assign
- Deferred
- Rejected
- Test
- Reopened
- Verified
- Closed



Stages of Bug

- **New** : when the bug is posted for the first time, its state will be "NEW". This means that the bug is not yet approved.
- **Open**: After a tester has posted a bug, the leas of the tester approves that thee bug is genuine and he changes the state as "OPEN". Leas of tester can set the bug status as: Open can Assign the bug to developer or bug may be deferred until next release.
- **Assign** : Once the lead changes the state as "OPEN", he assigns the bug to corresponding developers or developer team. The state of the bug now is changed to "ASSIGN".
    - Developer can set bug status as:
        - Won't fix
        - Couldn't reproduce
        - Need more information or
        - Fixed.

Stages of Bug

- **Deferred** : If the bug is not related to current build or cannot be fixed in this release or bug is not important to fix immediately then the project manager can set the bug status as deferred.
- **Rejected** : If the developer feels that the bug is not genuine, he rejects the bug. The state of the bug is changed to “REJECTED”.
- **Test** : If the bug status set by developer is either “Need more info” or fixed the QA responds with specific action.
- **Retest**: Once the developer fixes the bug, he has to assign the bug to the testing team for next round of testing. Before he releases the software with bug fixed, he changes the state of bug to “RETEST”. It specifies that the bug has been fixed and is released to testing team.
- **Duplicate**: If the bug is repeated twice or two bugs mention the same concept of the bug, then one bug status is changed to “DUPLICATE”

Stages of Bug

- **Reopened** : If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “REOPENED”. The bug traverses the life cycle once again.
- **Closed** : Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to “CLOSED”. This state means that the bug is fixed, tested and approved.

The Severity of Bug

- **Critical /Show Stopper** : An item that prevents further testing of the product or function under test can be classified as Critical Bug. No workaround is possible for such bugs. Examples of this includes a missing menu option or security permission required to access a function under test.
- **Major / High** : A defect that does not function as expected/designed or cause other functionality to fail to meet requirements can be classified as Major Bugs. The work around can be provided for such bugs. Examples of this includes inaccurate calculations, the wrong field being updated, etc.
- **Average / Medium** : The defects which do not conform to standards and conventions can be classified as Medium Bugs. Easy workarounds exists to achieve functionality objectives. Examples include matching visual and text links which lead to different end points.

The Severity of Bug

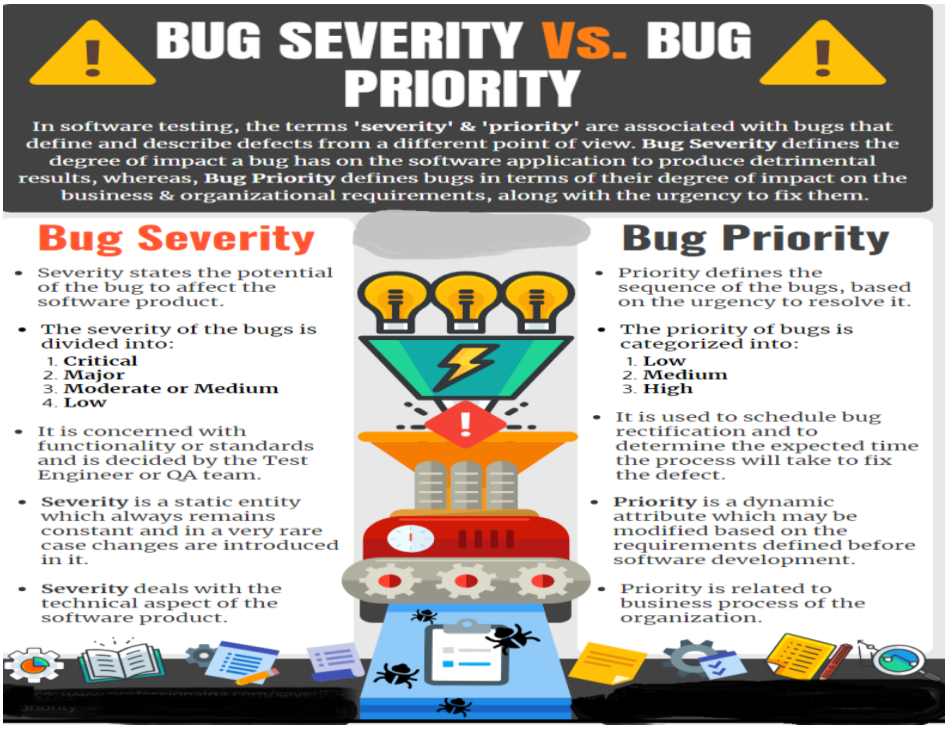
- **Minor / Low** : Cosmetic defects which does not affect the functionality of the system can be classified as Minor Bugs.

The Priority of Bug

- A priority classification of a software error is based on the importance and urgency of resolving the error.
  - **Immediate** : The bug should be resolved immediately.
  - **High** : This bug should be resolved as soon as possible in the normal course of development activity, before the software is released.
  - **Medium** : This bug should be repaired after serious bug have been fixed.
  - **Low** : It can be resolved in a future major system revision or not be resolved at all.

Severity Vs Priority

- **What is Severity?** Bug Severity defines the degree of impact of a bug on a software application to produce detrimental results. Greater the severity, larger would be the impact on the functionalities.
- **What is Priority?** Unlike, bug severity, bug priority defines a bug in the terms of its degree of impact on the business & organizational needs and requirements along with the urgency to fix that bug. Higher the priority, sooner the bug will be resolved.



BUG SEVERITY Vs. BUG PRIORITY

In software testing, the terms 'severity' & 'priority' are associated with bugs that define and describe defects from a different point of view. **Bug Severity** defines the degree of impact a bug has on the software application to produce detrimental results, whereas, **Bug Priority** defines bugs in terms of their degree of impact on the business & organizational requirements, along with the urgency to fix them.

Bug Severity

- Severity states the potential of the bug to affect the software product.
- The severity of the bugs is divided into:
  1. **Critical**
  2. **Major**
  3. **Moderate or Medium**
  4. **Low**
- It is concerned with functionality or standards and is decided by the Test Engineer or QA team.
- **Severity** is a static entity which always remains constant and in a very rare case changes are introduced in it.
- **Severity** deals with the technical aspect of the software product.

Bug Priority

- Priority defines the sequence of the bugs, based on the urgency to resolve it.
- The priority of bugs is categorized into:
  1. **Low**
  2. **Medium**
  3. **High**
- It is used to schedule bug rectification and to determine the expected time the process will take to fix the defect.
- **Priority** is a dynamic attribute which may be modified based on the requirements defined before software development.
- Priority is related to business process of the organization.

[redacted]

[redacted]

Why do bug occur

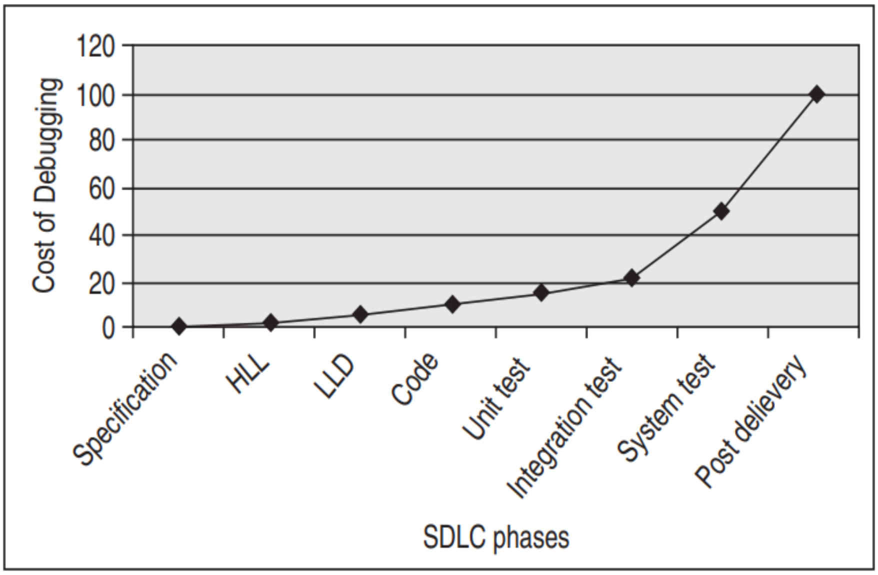
- Miscommunication or No Communication
- Software Complexity
- Lack of Designing Experience/Faulty Design Logic
- Coding/Programming Errors
- Ever-Changing Requirements
- Time Pressures (Unrealistic Time Schedule)
- Egotistical or Overconfident People
- Poorly Documented Code
- Software Development Tools (Third-party Tools and Libraries)
- Obsolete Automation Scripts or Over-Reliance on Automation
- Lack of Skilled Testers
- Absence or Inadequate Version Control Mechanism
- Frequent Releases

Why do bug occur

- Insufficient Training for Staff
- Changes at the Eleventh Hour (Last-Minute Changes)
- Ineffective Testing Life Cycle
- Not Automating Repetitive Test Cases and depending on the testers for manual verification every time.
- Not tracking the development and test execution progress continuously.
- The incorrect design leads to issues being carried out in all phases of the Software Development Cycle.
- Any wrong assumption(s) made during coding and testing stages.

Bugs Affect Economics Of Software Testing

- The cost of bug is equal to detection cost + correction cost.



Bug Classification Based On Criticality

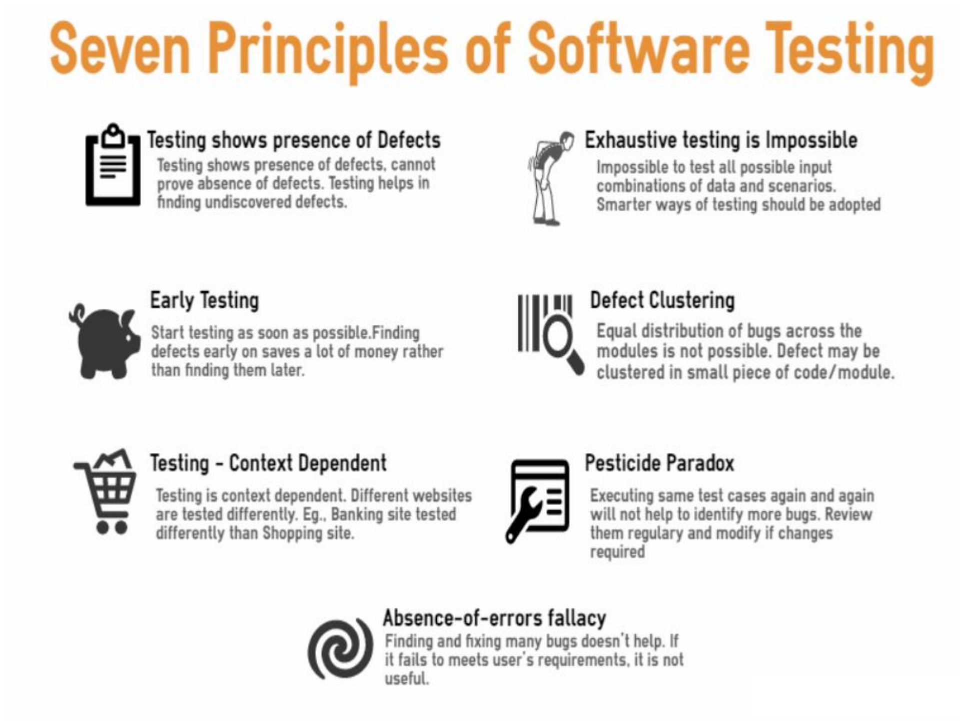
- Bugs can be classified based on the impact they have on the software under test.
- This classification can be used for the prioritization of bugs, as all bugs cannot be resolved in one release.
- **Critical Bugs** : This type of bugs has the worst effect such that it stops or hangs the normal functioning of the software.
  - The person using the software becomes helpless when this type of bug appears.
  - For example, in a sorting program, after providing the input numbers, the system hangs and needs to be reset.
- **Major Bug** : This type of bug does not stop the functioning of the software but it causes a functionality to fail to meet its requirements as expected.
  - For example, in a sorting program, the output is being displayed but not the correct one.

Bug Classification Based On Criticality

- **Medium Bugs** : Medium bugs are less critical in nature as compared to critical and major bugs.
  - If the outputs are not according to the standards or conventions, e.g. redundant or truncated output, then the bug is a medium bug.
- **Minor Bugs** : These types of bugs do not affect the functionality of the software.
  - These are just mild bugs which occur without any effect on the expected behaviour or continuity of the software.
  - For example, typographical error or misaligned printout.

Bug Classification Based On SDLC

- Requirements and Specifications Bugs
- Design Bugs
  - Control flow bugs
  - Logic bugs
  - Processing bugs
  - Data flow bugs
  - Error handling bugs
  - Race condition bugs
  - Boundary-related bugs
  - User interface bugs
- Coding Bugs
- Interface and Integration Bugs
- System Bugs
- Testing Bugs



Seven Principles of Software Testing

Testing shows presence of Defects
Testing shows presence of defects, cannot prove absence of defects. Testing helps in finding undiscovered defects.

Exhaustive testing is Impossible
Impossible to test all possible input combinations of data and scenarios. Smarter ways of testing should be adopted

Early Testing
Start testing as soon as possible.Finding defects early on saves a lot of money rather than finding them later.

Defect Clustering
Equal distribution of bugs across the modules is not possible. Defect may be clustered in small piece of code/module.

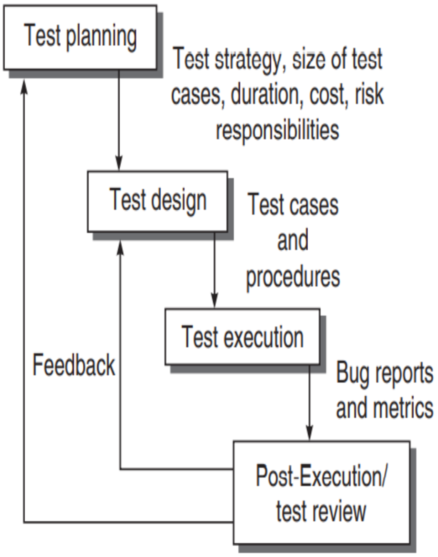
Testing - Context Dependent
Testing is context dependent. Different websites are tested differently. Eg., Banking site tested differently than Shopping site.

Pesticide Paradox
Executing same test cases again and again will not help to identify more bugs. Review them regulary and modify if changes required

Absence-of-errors fallacy
Finding and fixing many bugs doesn't help. If it fails to meets user's requirements, it is not useful.

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
Bug reports and metrics
Post-Execution/ test review
Feedback

47

Software Testing Life Cycle (STLC)

- **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.
- 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

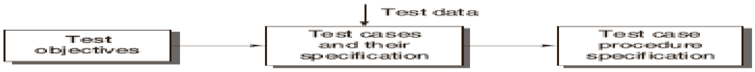
48

Software Testing Life Cycle (STLC)

- **Test Planning :** Based on the planning issues, 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

Software Testing Life Cycle (STLC)

- **Test Design :** One of the major activities in testing is the design of test cases.
- However, this activity is not an intuitional process; rather it is a well-planned process.
- The test design is an important phase after test planning.
- It includes the following critical activities.
  - Determining the test objectives and their prioritization.
  - Preparing list of items to be tested
  - Mapping items to test cases
  - Selection of test case design techniques
  - Creating test cases and test data
  - Setting up the test environment and supporting tools



Test data

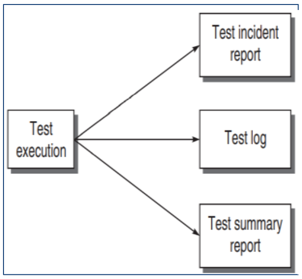
Test objectives

Test cases and their specification

Test case procedure specification

Software Testing Life Cycle (STLC)

- **Test Execution :** In this phase, all test cases are executed including verification and validation.
- Verification test cases are started at the end of each phase of SDLC.
- Validation test cases are started after the completion of a module.
- It is the decision of the test team to opt for automation or manual execution.



Test Execution Level	Person Responsible
Unit	Developer of the module
Integration	Testers and Developers
System	Testers, Developers, End-users
Acceptance	Testers, End-users

Software Testing Life Cycle (STLC)

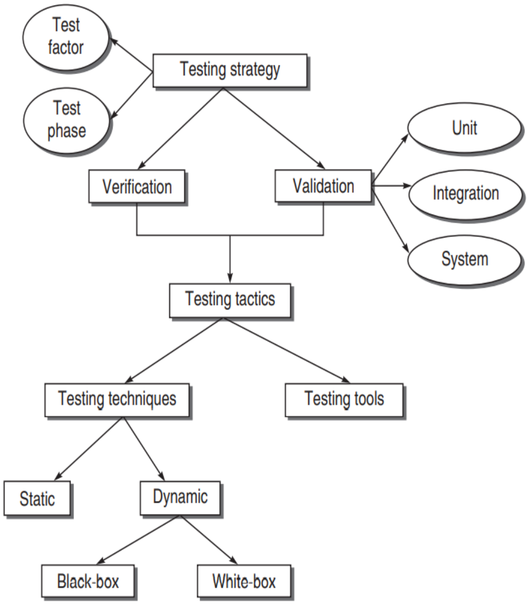
- **Post-Execution/Test Review :** As we know, after successful test execution, bugs will be reported to the concerned developers.
- This phase is to analyse bug-related issues and get feedback so that maximum number of bugs can be removed.
- This is the primary goal of all test activities done earlier.
- As soon as the developer gets the bug report, he performs the following activities:
  - Understanding the bug
  - Reproducing the bug
  - Analysing the nature and cause of the bug
- After this, the results from manual and automated testing can be collected.
- The final bug report and associated metrics are reviewed and analysed for overall testing process.

Software Testing Life Cycle (STLC)

- **Post-Execution/Test Review**
- The following activities can be done:
  - ***Reliability analysis*** to establish whether the s/w meets the predefined reliability goals or not. If so, the product can be released and if not, then time and resources are required to reach reliability goals.
  - ***Coverage Analysis:*** used as an alternative to stop testing.
  - ***Overall Defect Analysis:*** to identify risk areas and focus on quality improvement.

Software Testing Methodology

- Software testing methodology is the organization of software testing by means of which the test strategy and test tactics are achieved, as shown in Fig.



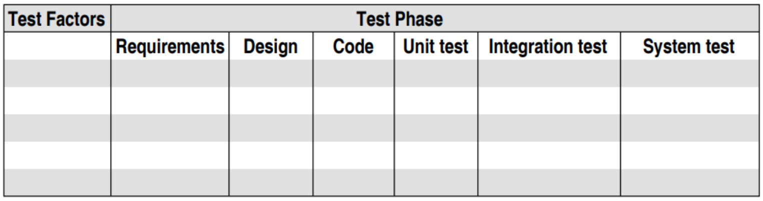
Software Testing Methodology

- **SOFTWARE TESTING STRATEGY**
- Testing strategy is the planning of the whole testing process into a well-planned series of steps.
- When a test strategy is being developed, risk reduction is addressed first.
- Two components:

1. **Test Factors**: Test factors are risk factors or issues related to the system under development.
   - Risk factors need to be selected and ranked according to a specific system under development.
   - The testing process should reduce these test factors to a prescribed level.
2. **Test Phase** : It refers to the phases of SDLC where testing will be performed. E.g. Strategies different for spiral and iterative waterfall.

Software Testing Methodology

- **Test Strategy Matrix :**
- A test strategy matrix identifies the concerns that will become the focus of test planning and execution.
- In this way, this matrix becomes an input to develop the testing strategy.
- The matrix is prepared using test factors and test phase
  - Select and rank test factors
  - Identify system development phases
  - Identify risks associated with the system under development



Test Factors	Test Phase					
	Requirements	Design	Code	Unit test	Integration test	System test

Software Testing Methodology

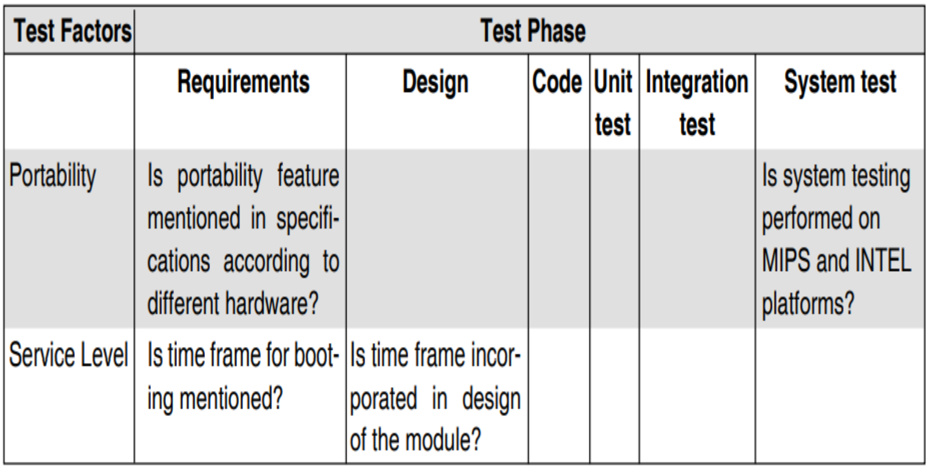
- **Test Strategy Matrix :**
- **Select and rank test factors** : Based on the test factors list, the most appropriate factors according to specific systems are selected and ranked from the most significant to the least. These are rows of a matrix.
- **Identify system development phases:** Different phases according to the adopted development model are listed as columns of the matrix. These are called test phases.
- **Identify risks associated with the system under development and its corresponding phase:** In horizontal column under each test phases, the test concern with the strategy used to address this concern is entered.

Software Testing Methodology

- **Test Strategy Matrix :**
- **Identify risks associated with the system under development and its corresponding phase:**
- The risks may include any circumstance, events, actions that may prevent the test program from being implemented or executed according to schedule, such as late budget approval, delayed arrival of test equipment, or late availability of the software application.
- Plan the test strategy for every risk identified so that risks are mitigated.

Software Testing Methodology

- Let’s take a project as an example.
- Suppose a new operating system has to be designed, which needs a test strategy.



Test Factors	Test Phase					
	Requirements	Design	Code	Unit test	Integration test	System test
Portability	Is portability feature mentioned in specifications according to different hardware?					Is system testing performed on MIPS and INTEL platforms?
Service Level	Is time frame for booting mentioned?	Is time frame incorporated in design of the module?				

Software Testing Methodology

- **Development Of Test Strategy**
- When the project under consideration starts and progresses, testing too starts from the first step of SDLC.
- Therefore, the test strategy should be such that the testing process continues till the implementation of project.
- Moreover, the rule for development of a test strategy is that testing ‘begins from the smallest unit and progresses to enlarge’.
- **This means the testing strategy should start at the component level and finish at the integration of the entire system.**
- Thus, a test strategy includes testing the components being built for the system, and slowly shifts towards testing the whole system.
- This gives rise to two basic terms—**Verification and Validation—** the basis for any type of testing.

Software Testing Methodology

- **Development Of Test Strategy**
- It can also be said that the testing process is a combination of verification and validation.
- The purpose of verification is to check the software with its specification at every development phase such that any defect can be detected at an early stage of testing and will not be allowed to propagate further.
- That is why verification can be applied to all stages of SDLC.
- So verification refers to the set of activities that ensures correct implementation of functions in a software.
- However, as we progress down to the completion of one module or system development, the scope of verification decreases.
- The validation process starts replacing the verification in the later stages of SDLC.

Software Testing Methodology

- **Development Of Test Strategy**
- Validation is a very general term to test the software as a whole in conformance with customer expectations.
- **According to Boehm**
  - Verification is 'Are we building the product right?'
  - Validation is 'Are we building the right product?'

Verification
is about,
"Are we building
the product right?"
The software should
conform to its
specification
Validation
is about,
"Are we building
the right product?"
The software should
do what the user
really requires

Software Testing Methodology

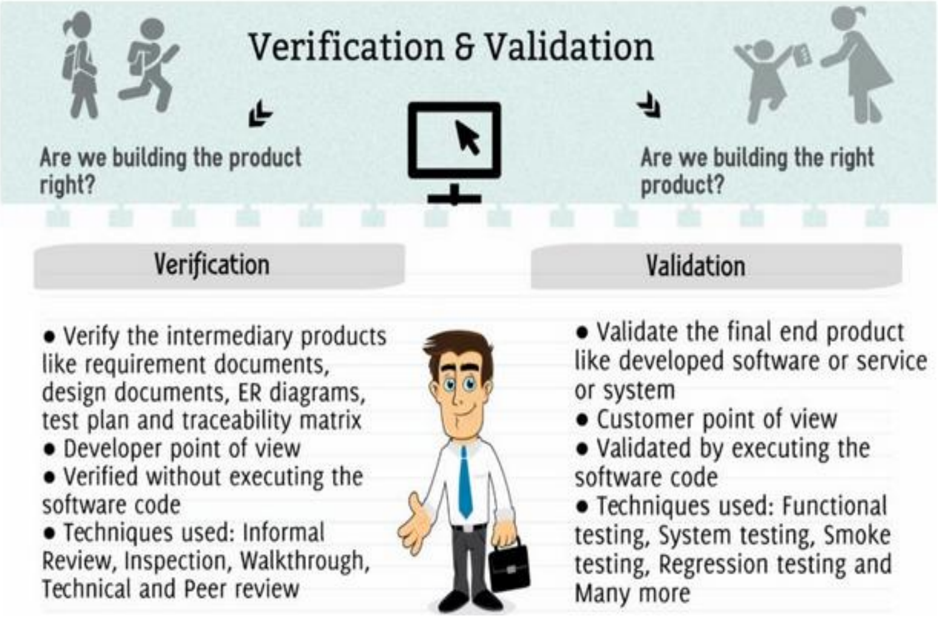
- **Development Of Test Strategy**
- **Verification**: “Are we building the product right?” is to check the software with its specification at every development phase such that any defect can be detected at any early stage of testing and will not be allowed to propagate further.
- Verification refers to the set of activities that ensures correct implementation of functions in a software.
- **Validation**: “Are we building the right product?”. When different modules of a system is integrated, the system built after integration. This is validation testing.
- Validation has the following three activities which are also known as the three levels of validation testing.

Software Testing Methodology

- **Development Of Test Strategy**
- **Unit Testing**: it is a major validation effort performed on the smallest module of the system.
  - Unit testing is a basic level of testing which cannot be overlooked, and confirms the behavior of a single module according to its functional specifications.
- **Integration** Testing: it is a validation technique which combines all unit tested modules and performs a test on their aggregation?
  - When one module is combined with another in an integrated environment, interfacing between units must be tested.
  - We integrate the units according to the design and availability of units. Therefore, tester must be aware of system design.

Software Testing Methodology

- **Development Of Test Strategy**
- **System Testing**: This testing level focuses on testing the entire integrated system.
  - It incorporates many types of testing. The purpose is to test the validity for specific users and environments.
  - The validity of the whole system is checked against the requirement specification.



Verification & Validation

Are we building the product right?

Are we building the right product?

Verification

- Verify the intermediary products like requirement documents, design documents, ER diagrams, test plan and traceability matrix
- Developer point of view
- Verified without executing the software code
- Techniques used: Informal Review, Inspection, Walkthrough, Technical and Peer review

Validation

- Validate the final end product like developed software or service or system
- Customer point of view
- Validated by executing the software code
- Techniques used: Functional testing, System testing, Smoke testing, Regression testing and Many more

Testing Tactics

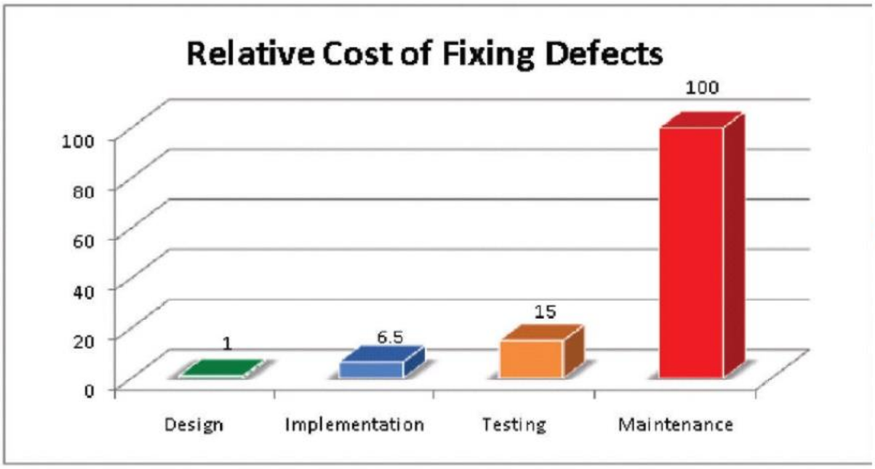
- **Software testing techniques**: effective testing is a real challenge.
- Design effective test cases with most of the testing domains covered detecting the maximum number of bugs.
- Actual methods for designing test cases, software testing technique, implement the test cases on the software.
- These techniques are categorized into two parts.
- **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 syntactical rules.
- **Dynamic Testing**: All the methods that execute the code to test a software are known as dynamic testing techniques.
- The code is run on a number of inputs provided by the user and the corresponding results are checked. Further divided into sub two categories: Black box testing and White box testing.

Testing Tactics

- **Black box testing**: It checks only the functionality of system.
  - This techniques received inputs given to system and the output is received after processing in the system.
  - It is also known as functional testing. It is used for system testing under validation.
- **White box testing**: Every design feature and its corresponding code is checked logically with every possible path execution.
  - So it takes care of structural paths instead of just outputs.
  - It is known as structural testing and also used for unit testing under verification.

Testing Tactics

- **Black box testing**: It checks only the functionality of system.
  - This techniques received inputs given to system and the output is received after processing in the system.
  - It is also known as functional testing. It is used for system testing under validation.
- **White box testing**: Every design feature and its corresponding code is checked logically with every possible path execution.
  - So it takes care of structural paths instead of just outputs.
  - It is known as structural testing and also used for unit testing under verification.



IBM System Science Institute Relative Cost of Fixing Defects

THANKS