Software Testing

UNIT V

SY. Semester V

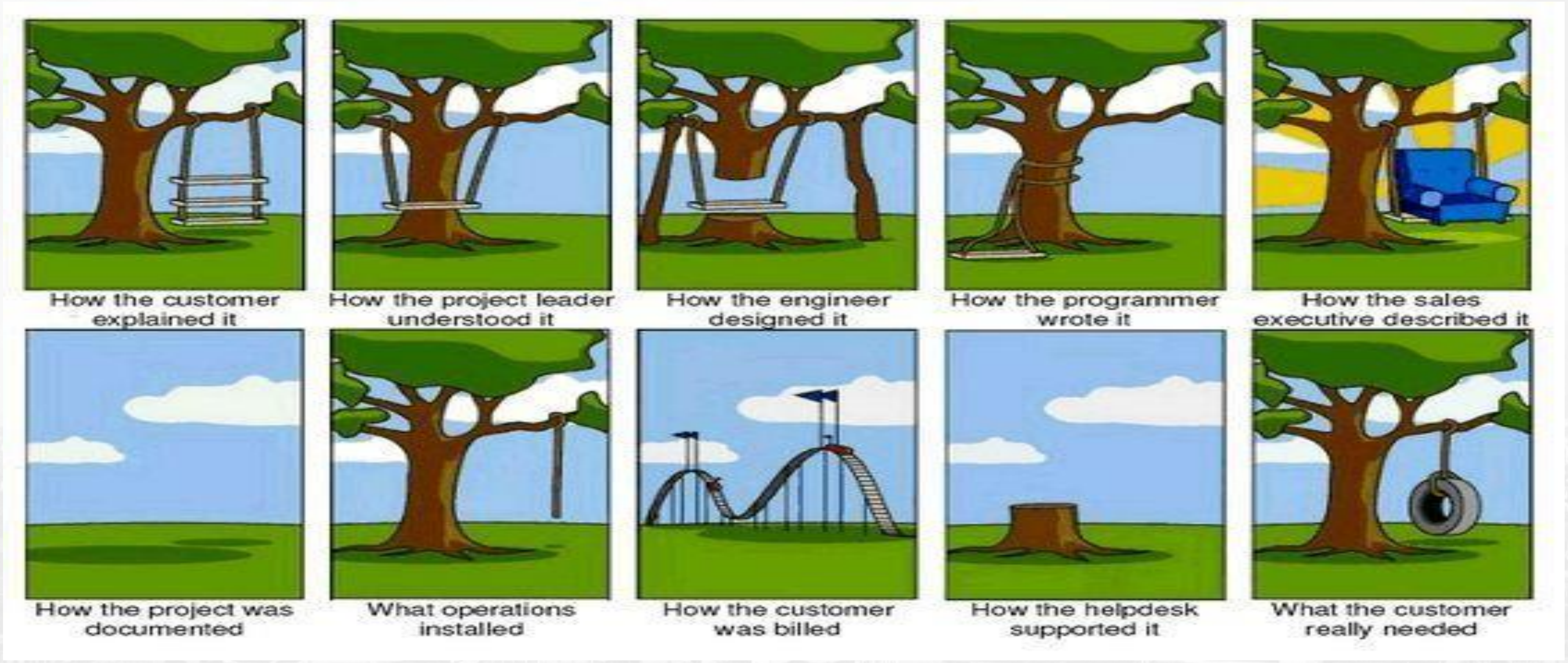
Overview

- Software testing is an important activity that:
  - Validates and verifies the requirements against the final or intermittent deliverables
  - Reviews the product architecture and design
  - Helps developers to improve their coding, design patterns and tests written on the basis of code
  - Executes the application with an intent to examine its behavior
  - Reviews the deployment environment and automation associated with it
  - Takes part in production activities

A little bit of History

- Software testing has been prevalent and important ever since the first piece of software was written in 1940s-50s!
- Earlier methods of software testing mainly involved debugging and fixing defects.
- Starting 1980s, software developers started looking at the testing activity beyond debugging.
- Starting 1990s, the testing activity transitioned into more comprehensive quality assurance(QA) process that covers the entire software development lifecycle.
- QA process involves planning, design, creation and execution of test cases along with support for existing test cases and environments.

Reality of Software Projects


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Purpose of testing

- There are two fundamental purposes of testing: verifying procurement specifications and managing risk.
- First, testing is about verifying that what was specified is what was delivered: it verifies that the product (system) meets the functional, performance, design, and implementation requirements identified in the procurement specifications.
- Second, testing is about managing risk for both the acquiring agency and the system's vendor/developer/integrator. The testing program is used to identify when the work has been "completed" so that the contract can be closed, the vendor paid, and the system shifted by the agency into the warranty and maintenance phase of the project

Need of software Testing

1. To gain customer confidence
2. To check software adaptability
3. To identify errors
4. To avoid extra costs
5. To accelerate software development
6. To avoid risks
7. To optimise business

- **1. To Gain Customer Confidence**
- Software testing makes sure that the software is user-friendly. That makes it capable of being used by the customers it is intended for. Those who specialise in software application testing are familiar with the needs of customers, and unless a software can satisfy a customer's needs, it would be a practically useless investment.

2. To Check Software Adaptability

- Given that there are many devices, operating systems, and browsers available today, it is necessary for software to be compatible with all platforms in order to offer users a smooth user experience.
- If the functionality of software is affected by the change of devices, it can count towards a negative user experience. Testing eliminates such errors in the performance while adding to the compatibility and adaptability of the software.

3. To Identify Errors

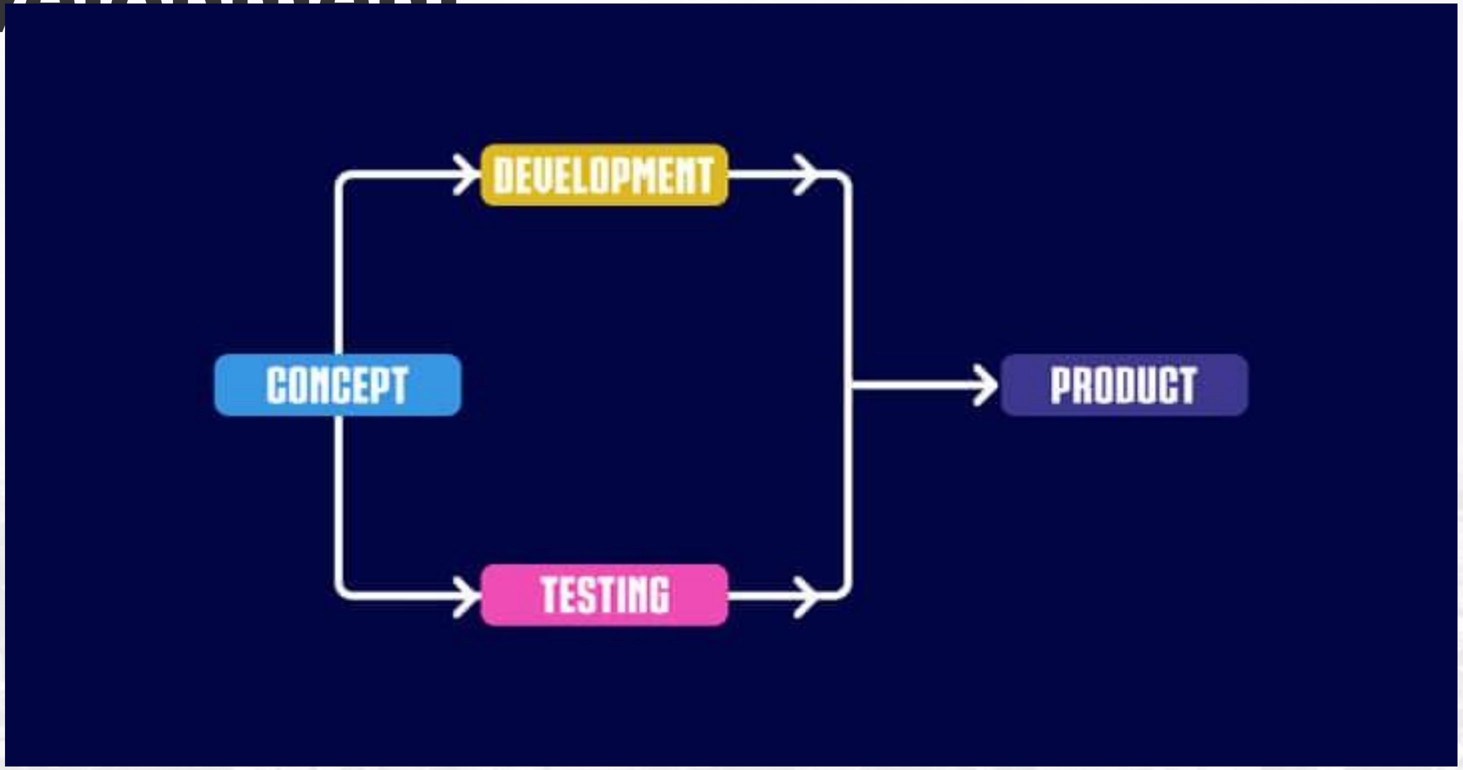
Regardless of how competent software developers and engineers may be, the possibility of glitches and bugs is always present in untested software. Testing will lead to better functioning of the product as hidden errors will be exposed and fixed.

Even a single bug can be damaging for the reputation of a software developing house. It can take a long time to regain customer confidence, so it's much better and ultimately more convenient to ensure testing is being achieved.

4. To Avoid Extra Costs

Tied to the problem of errors is the issue of the costs of reimbursing clients who have experienced glitchy software. These additional expenses can amount to a significant amount of damages, as not only has the client been dissatisfied with the product for which they have paid, but a client’s time that they could have invested elsewhere was also rendered worthless.

- **5. To Accelerate Software Development**



- Software testing and software development if run in parallel, can accelerate the software development process and make it more efficient. Staging the design process in a way that makes it certain that both software testing and software development are happening simultaneously takes care to avoid such pit-falls in software development.

6. To Avoid Risks

- The worst thing about bugs and glitches is that it indicates a software is not secure. Especially when it comes to software that is meant for organisations, errors or loopholes can lead to vulnerability.
- This can lead to huge losses of information to competitor businesses, and can also lead to a lot of communication errors within an organisation.

7. To Optimise Business

Testing allows the end-product to achieve a higher quality standard before being made live.

It also adds to the company’s brand image as well as its profitability through reduced support costs.

Essentially, every software developer’s goal is customer retention, and every customer’s goal is finding a service that’s reliable and worth their money. Providing effective software thus, allows a business to become entrenched in a software provider’s reputation.

Bugs Can be called as…..

- Defect
- Fault
- Problem
- Error
- Incident
- Anomaly
- Variance

- Failure
- Inconsistency
- Product Anomaly
- Product Incidence
- Feature :-)

Sources of Problems

- **Requirements Definition:** Erroneous, incomplete, inconsistent requirements.
- **Design:** Fundamental design flaws in the software.
- **Implementation:** Mistakes in chip fabrication, wiring, programming faults, malicious code.
- **Support Systems:** Poor programming languages, faulty compilers and debuggers, misleading development tools.

Sources of Problems (Cont’d)

- **Inadequate Testing of Software:** Incomplete testing, poor verification, mistakes in debugging.
- **Evolution:** Sloppy redevelopment or maintenance, introduction of new flaws in attempts to fix old flaws, incremental escalation to inordinate complexity.

Software Testing Principles

- Testing weeds out defects of different types at different levels
- Exhaustive Testing is not possible
  - There is always one more bug!
- Testing may demonstrate “defect clustering”;
  - Pareto principle : ~80% of the problems are found in 20% of the modules!
- Pesticide Paradox : Same old test suites and testing techniques will not yield new defects. The tests need to be regularly reviewed and revised.
- Absence of errors does not mean the software is usable and useful
- Early Testing (right from requirements phase) results in cheaper defect fixes.
- Testing is context dependent
  - The approach, methodologies, techniques and types of testing depend on the application domain.

1) Exhaustive testing is not possible

- Yes! Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application.
- And the million dollar question is, how do you determine this risk?
- To answer this let's do an exercise
- In your opinion, Which operation is most likely to cause your Operating system to fail?
- I am sure most of you would have guessed, Opening 10 different application all at the same time.
- So if you were testing this Operating system, you would realize that defects are likely to be found in multi-tasking activity and need to be tested thoroughly

2) Defect Clustering

- Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.
- By experience, you can identify such risky modules. But this approach has its own problems
- If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs.

3) Pesticide Paradox

- Repetitive use of the same pesticide mix to eradicate insects during farming will over time lead to the insects developing resistance to the pesticide Thereby ineffective of pesticides on insects. The same applies to software testing. If the same set of repetitive tests are conducted, the method will be useless for discovering new defects.
- To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.
- Testers cannot simply depend on existing test techniques. He must look out continually to improve the existing methods to make testing more effective. But even after all this sweat & hard work in testing, you can never claim your product is bug-free. To drive home this point, let’s see this video of the public launch of Windows 98
- You think a company like MICROSOFT would not have tested their OS thoroughly & would risk their reputation just to see their OS crashing during its public launch!

- **4) Testing shows a presence of defects**
- Hence, testing principle states that – Testing talks about the presence of defects and don’t talk about the absence of defects. i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.
- But what if, you work extra hard, taking all precautions & make your software product 99% bug-free. And the software does not meet the needs & requirements of the clients.
- This leads us to our next principle, which states that- Absence of Error

- **5) Absence of Error – fallacy**
- It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not mere finding defects, but also to check that software addresses the business needs. The absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system build is unusable and does not fulfill the user's needs & requirements.
- To solve this problem, the next principle of testing states that

  Early Testing

6) Early Testing

- Early Testing – Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages. It is much cheaper to fix a Defect in the early stages of testing. But how early one should start testing? It is recommended that you start finding the bug the moment the requirements are defined. More on this principle in a later training tutorial.

7) Testing is context dependent

- Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application. All the developed software’s are not identical. You might use a different approach, methodologies, techniques, and types of testing depending upon the application type. For instance testing, any system at a retail store will be different than testing an ATM machine.

Goals of Software Testing

- Short term goals:
  - To find errors, gaps or missing requirements as compared to actual requirements
  - To ensure that the product matches the performance expectations
  - To ensure that the product **IS NOT** doing what it is **not supposed to do**!
- Long term goals:
  - Quality
  - Customer Satisfaction
  - Risk Management

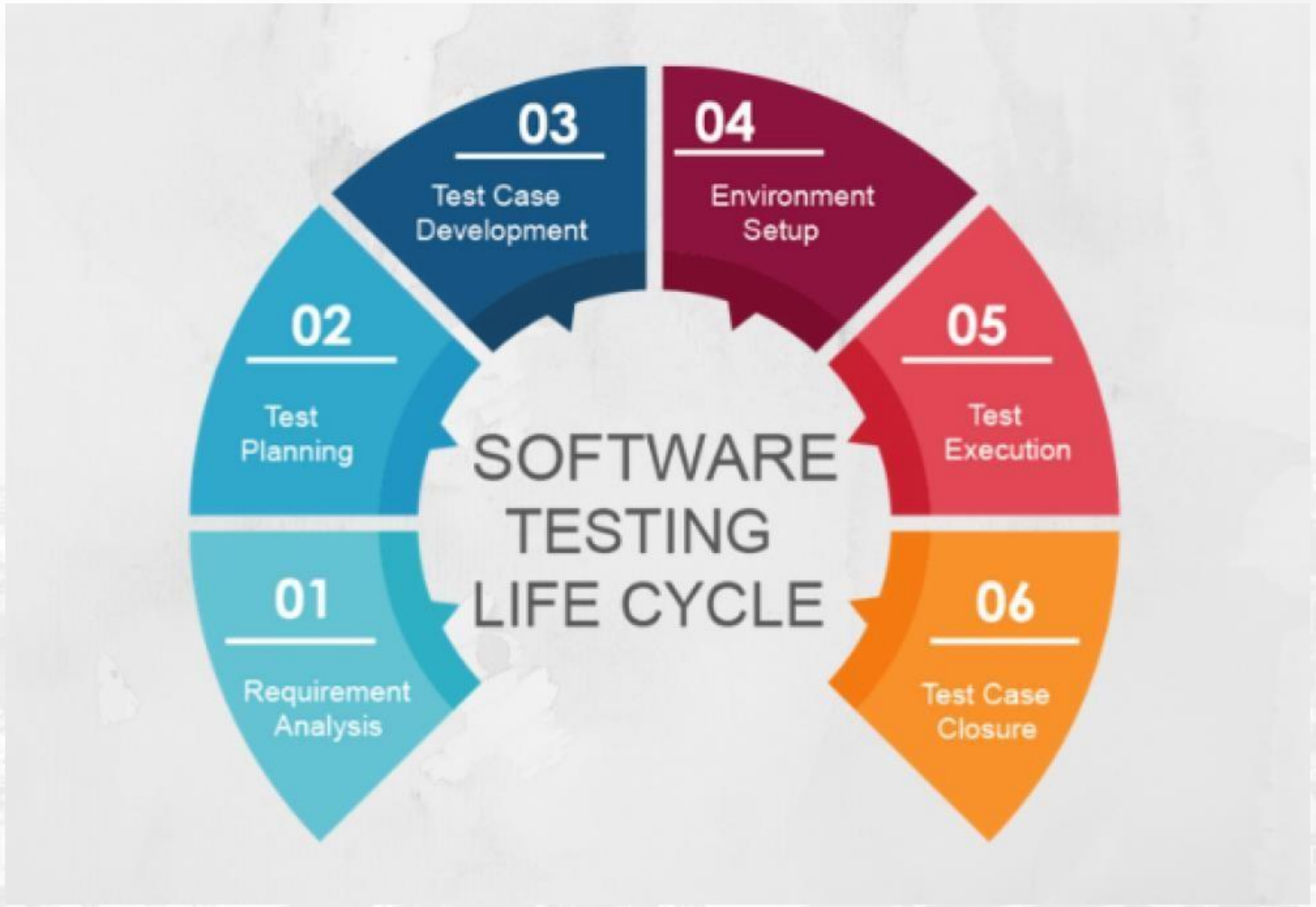
Software Testing Process

- Software testing is a **constructively destructive** process!
- Testers need to have a methodical, but negative approach.
- Testers need to establish a “**test to break**” attitude for successful testing and delivery of quality products.
- Testing is not only about finding and reporting defects but also about test planning, test execution and test coverage of all features.
- In traditional waterfall model, testing is done by independent team of testers, either after development is complete or along with the development process.
- Agile development methodologies promote “test-driven” software development where unit tests are written even before coding and are continuously updated and maintained by developers.

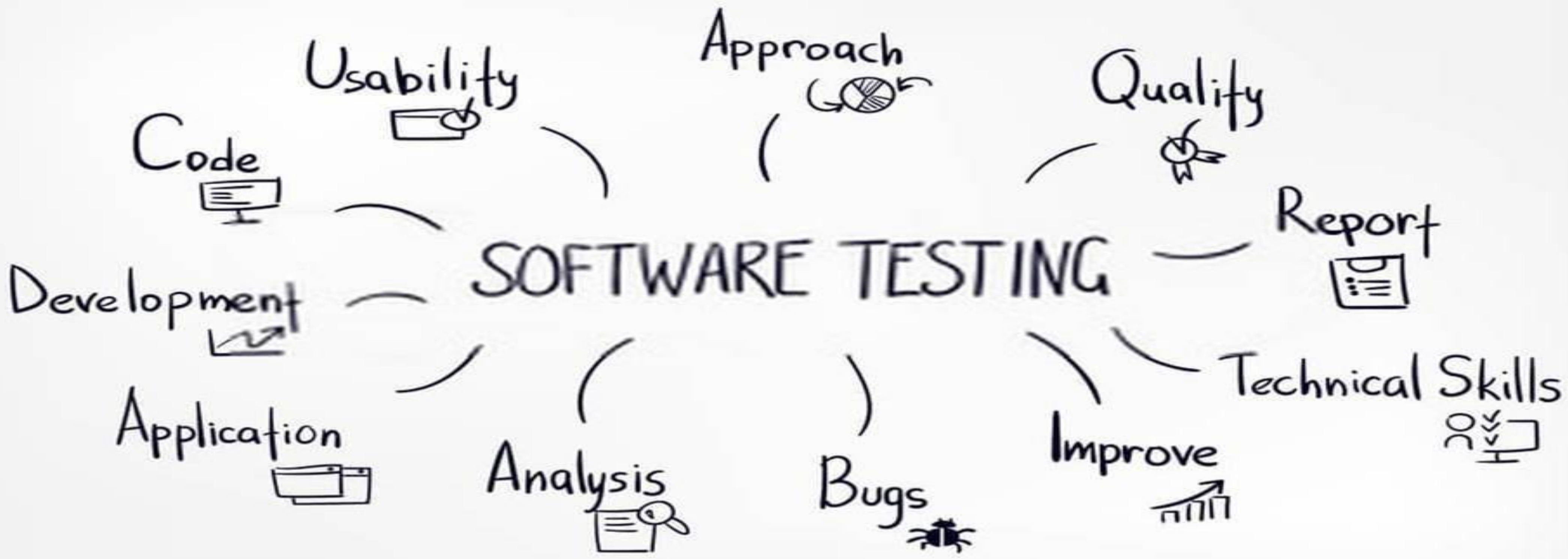
Testing Lifecycle

Like death and taxes,testing is both inevitable and unpleasant.

- Ed Yourdon



Software Testing : What is involved?

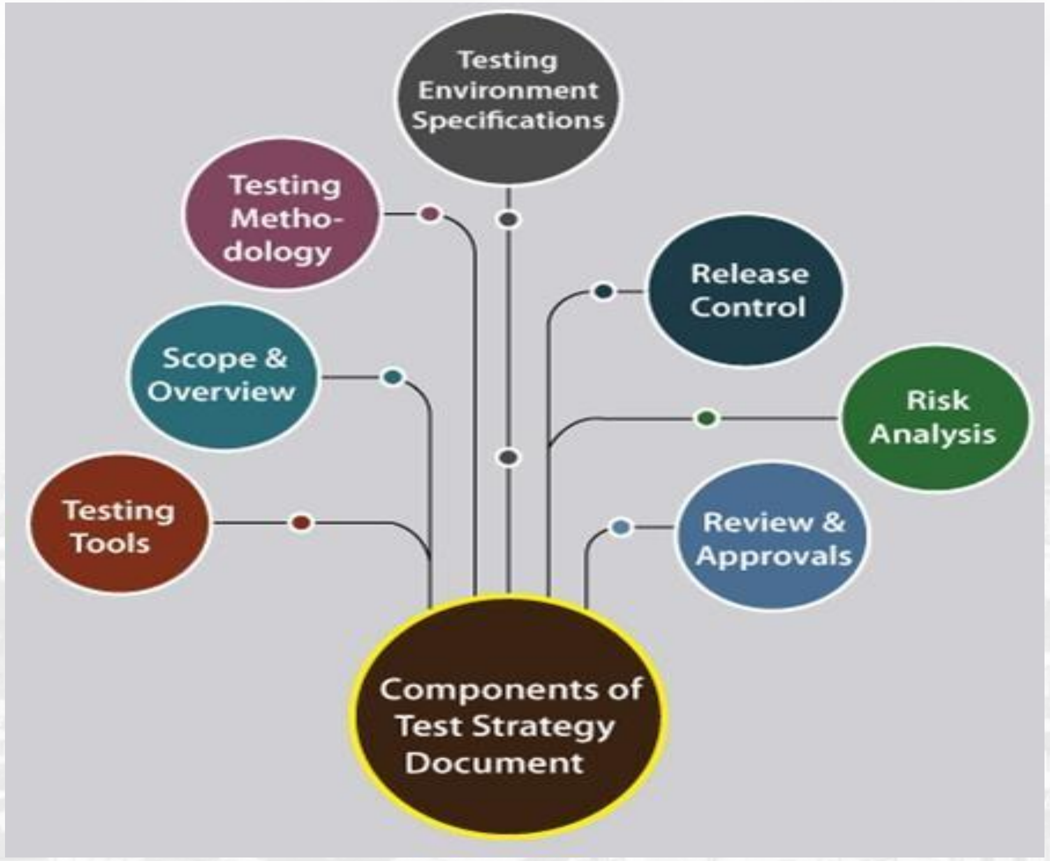


Software Testing Strategies

- It is important to devise a systematic strategy for testing. It avoids wastage of time and efforts.
- Some popular testing strategies are:
  - Effective technical reviews at every stage of SDLC
  - Component level testing that works “outward” toward integration testing
  - Different testing techniques for different software engineering approaches and at different times E.g. Unit testing at development time, smoke testing for nightly builds, regression testing for old features etc.
  - Testing done by developer and separate testing group
  - Testing and debugging as two separate activities

Test Strategy

- Test Strategy document is a high level document that explains the test methodologies, testing types, levels of testing and the overall approach.
- The objective is to make sure all stakeholders understand the purpose of testing cycle.



Verification and Validation

- Software testing consists of two widely used concepts:
  - Verification : Based on the assumptions made during earlier phases, does the software achieve its goals without any defects or gaps?
  - Validation : Was the software supposed to be built the way it is now built? Does it satisfy the high level requirements?
- Software teams need to ask the following questions :
  - Verification: “Are we building the product right?”
  - Validation: “Are we building the right product?”

V&V Activities

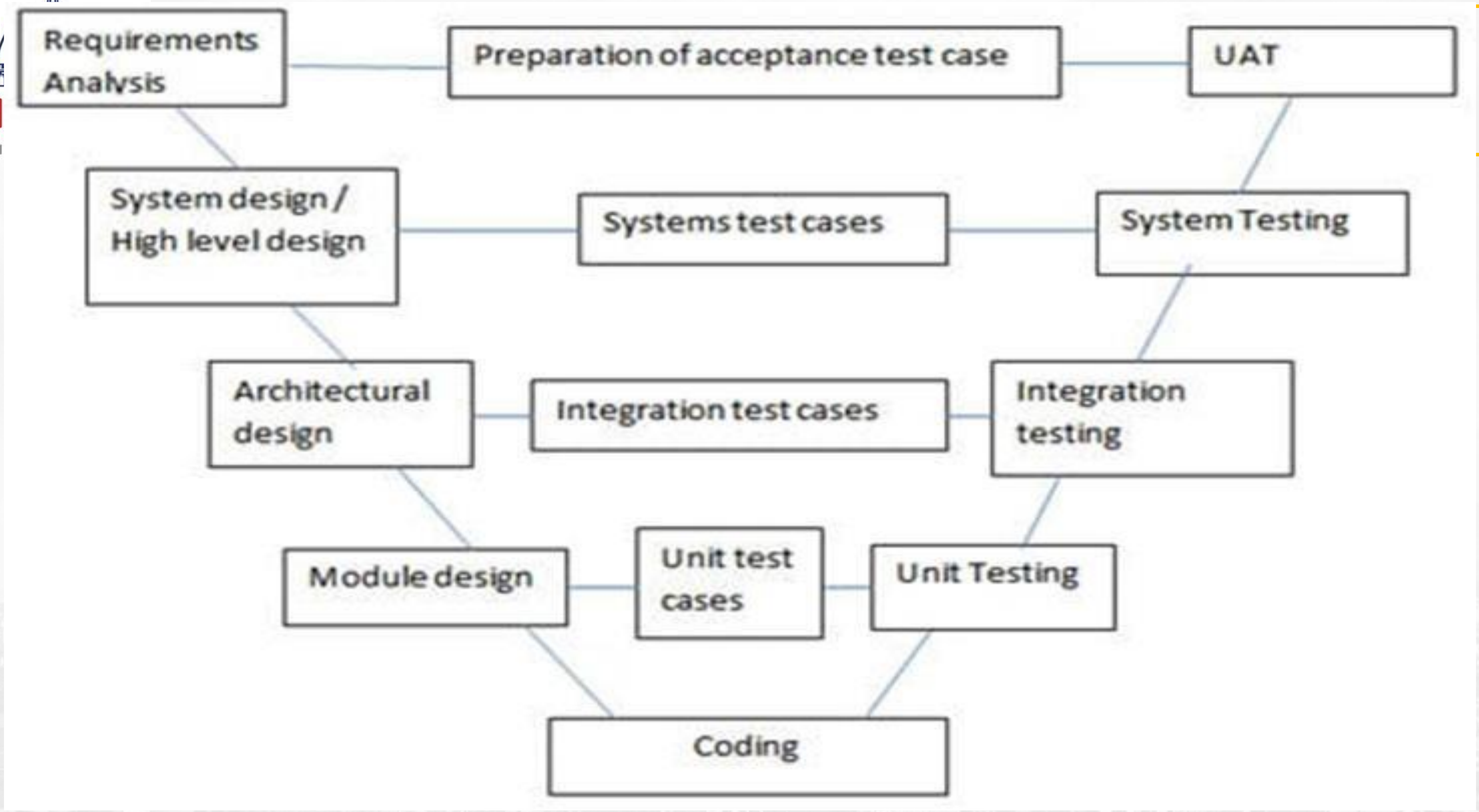
- V&V consists of a wide range of QA activities :
  - Technical Reviews
  - Configuration and Quality audits
  - Performance monitoring
  - Feasibility study
  - Documentation review
  - Algorithm analysis
  - A variety of testing such as usability, acceptance, installation etc.

V-Model

- In the V model, the development and QA activities are done simultaneously.

- There is no discrete phase called Testing, rather testing starts right from the requirement phase.

- The verification and validation activities go hand in hand.


Requirements Analysis
Preparation of acceptance test case
UAT
System design / High level design
Systems test cases
System Testing
Architectural design
Integration test cases
Integration testing
Module design
Unit test cases
Unit Testing
Coding

- In a typical development process, the left-hand side shows the development activities and the right hand side shows the testing activities.

- **Requirement analysis**: In this phase, the requirements are collected, analyzed and studied. Here how the system is implemented, is not important but, what the system is supposed to do, is important. Brain storming sessions/walkthrough, interviews are done to have the objectives clear.
- *Verification activities*: Requirements reviews.
- *Validation activities*: Creation of UAT (User acceptance test) test cases
- *Artifacts produced*: Requirements understanding document, UAT test cases.

- **System requirements /High-level design**: In this phase, the high-level design of the software is built. The team studies and investigates on how the requirements could be implemented. The technical feasibility of the requirements is also studied. The team also comes up with the modules that would be created/ dependencies, Hardware/software needs
- *Verification activities*: Design reviews

- *Validation activities*: Creation of System test plan and cases, Creation of traceability metrics

- *Artifacts produced*: System test cases, Feasibility reports, System test plan, Hardware-software requirements, and modules to be created, etc.

- **Architectural design:** In this phase, based on the high-level design, software architecture is created. The modules, their relationships, and dependencies, architectural diagrams, database tables, technology details are all finalized in this phase.
- *Verification activities*: Design reviews
- *Validation activities*: Integration test plan and test cases.
- *Artifacts produced*: Design documents, Integration test plan and test cases, Database table designs etc.

- **Module design/Low-level Design:** In this phase, each and every module of the software components are designed individually. Methods, classes, interfaces, data types etc are all finalized in this phase.
- *Verification activities*: Design reviews
- *Validation activities*: Creation and review of unit test cases.
- *Artifacts produced*: Unit test cases,

- **Implementation / Code**: In this phase, the actual coding is done.
- *Verification activities*: Code review, test cases review
- *Validation activities*: Creation of functional test cases.
- *Artifacts produced*: test cases, review checklist.

- Right-hand side demonstrates the testing activities or the Validation Phase. We will start from the bottom.
- **Unit Testing:** In this phase, all the unit test case, created in the Low-level design phase are executed.
- *Unit testing is a white box testing technique, where a piece of code is written which invokes a method (or any other piece of code) to test whether the code snippet is giving the expected output or not. This testing is basically performed by the development team. In case of any anomaly, defects are logged and tracked.
- *Artifacts produced*: Unit test execution results

- **Integration Testing**: In this phase, the integration test cases are executed which were created in the Architectural design phase. In case of any anomalies, defects are logged and tracked.
- *Integration Testing: Integration testing is a technique where the unit tested modules are integrated and tested whether the integrated modules are rendering the expected results. In simpler words, It validates whether the components of the application work together as expected.
- *Artifacts produced*: Integration test results.

- **Systems testing**: In this phase all the system test cases, functional test cases and nonfunctional test cases are executed. In other words, the actual and full fledge testing of the application takes place here. Defects are logged and tracked for its closure. Progress reporting is also a major part of this phase. The traceability metrics are updated to check the coverage and risk mitigated.
- *Artifacts produced*: Test results, Test logs, defect report, test summary report, and updated traceability matrices.

- **User Acceptance Testing**: Acceptance testing is basically related to business requirements testing. Here testing is done to validate that the business requirements are met in the user environment. Compatibility testing and sometimes nonfunctional testing (Load, stress, and volume) testing are also done in this phase.
- *Artifacts produced*: UAT results, Updated Business coverage matrices

- **When To Use The V Model?**
- V model is applicable when:
- The requirement is well defined and not ambiguous
- Acceptance criteria are well defined.
- Project is short to medium in size.
- Technology and tools used are not dynamic.

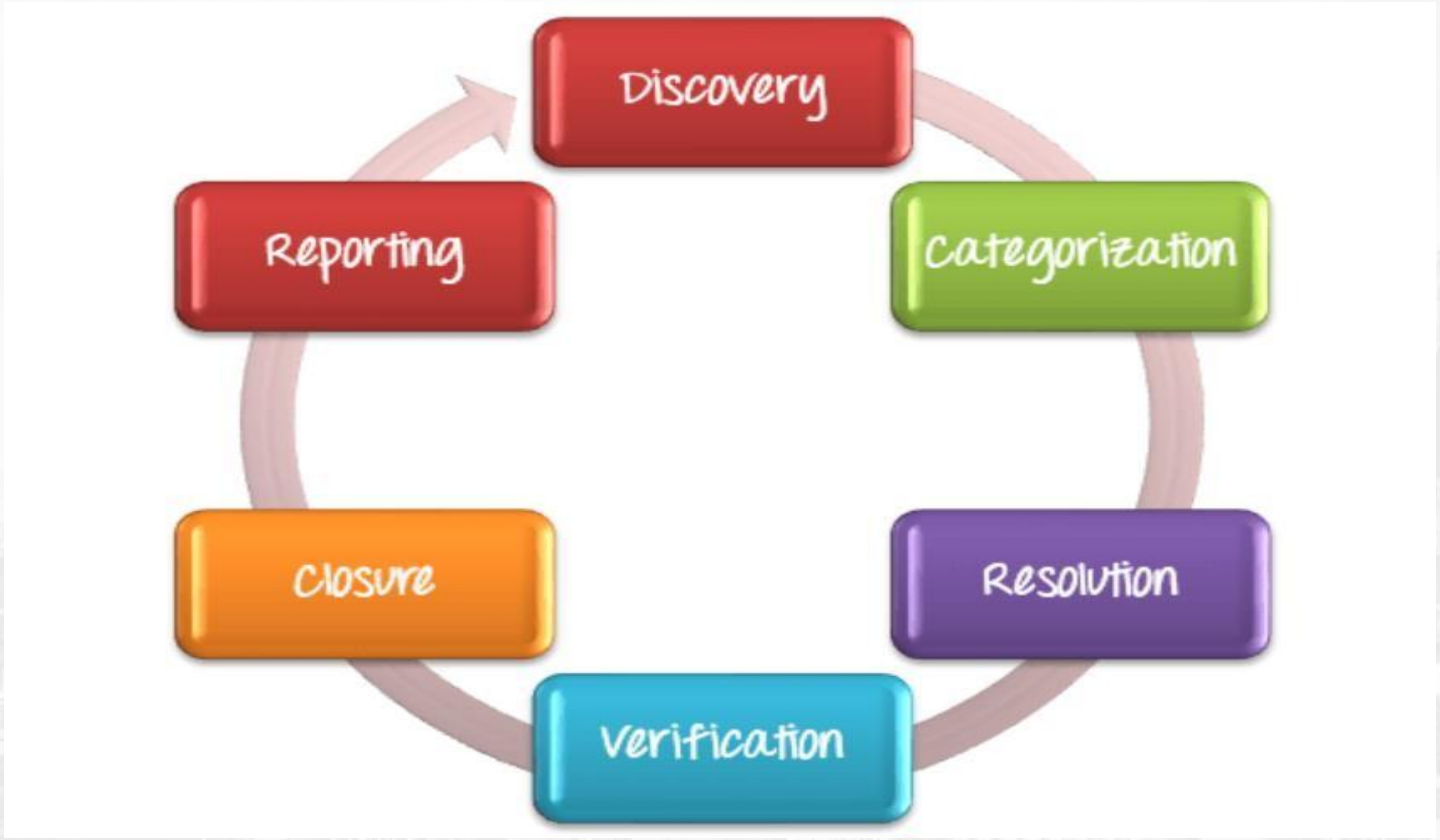
Pros and Cons of using V model

PROS	CONS
- Development and progress is very organized and systematic	-Not suitable for bigger and complex projects
- Works well for smaller to medium sized projects.	- Not suitable if the requirements are not consistent.
- Testing starts from beginning so ambiguities are identified from the beginning.	- No working software is produced in the intermediate stage.
- Easy to manage as each phase has well defined objectives and goals.	- No provision for doing risk analysis so uncertainty and risks are there.

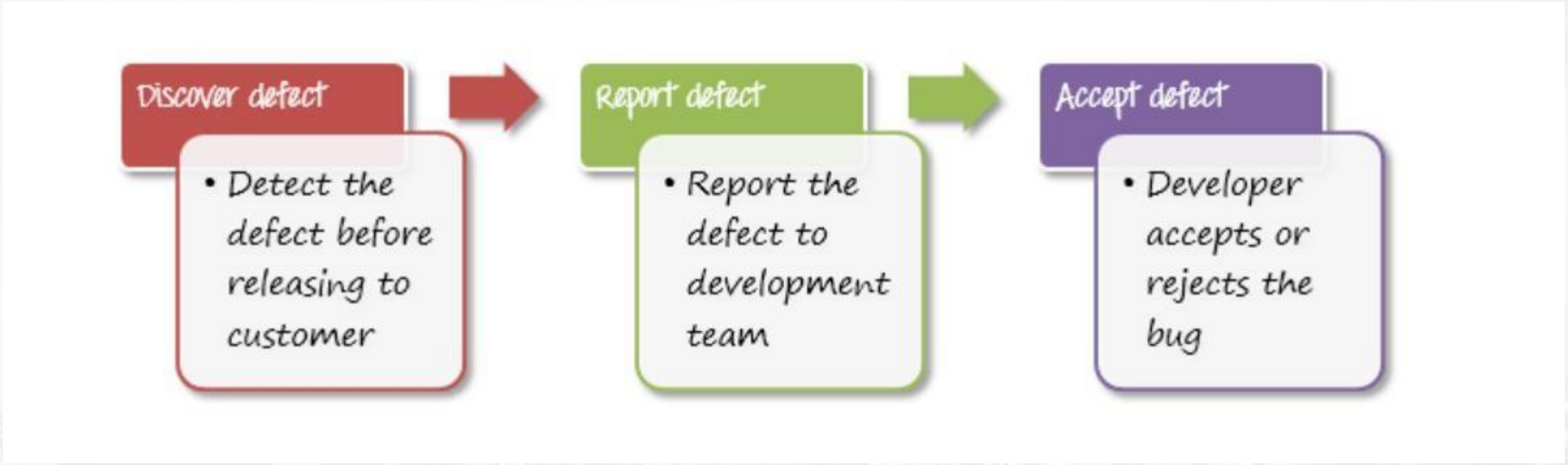
Defect Management Process in Software Testing

- Defect Management is a systematic process to identify and fix bugs. A defect management cycle contains the following stages

1) Discovery of Defect
2) Defect Categorization
3) Fixing of Defect by developers
4) Verification by Testers
5) Defect Closure
6) Defect Reports at the end of project

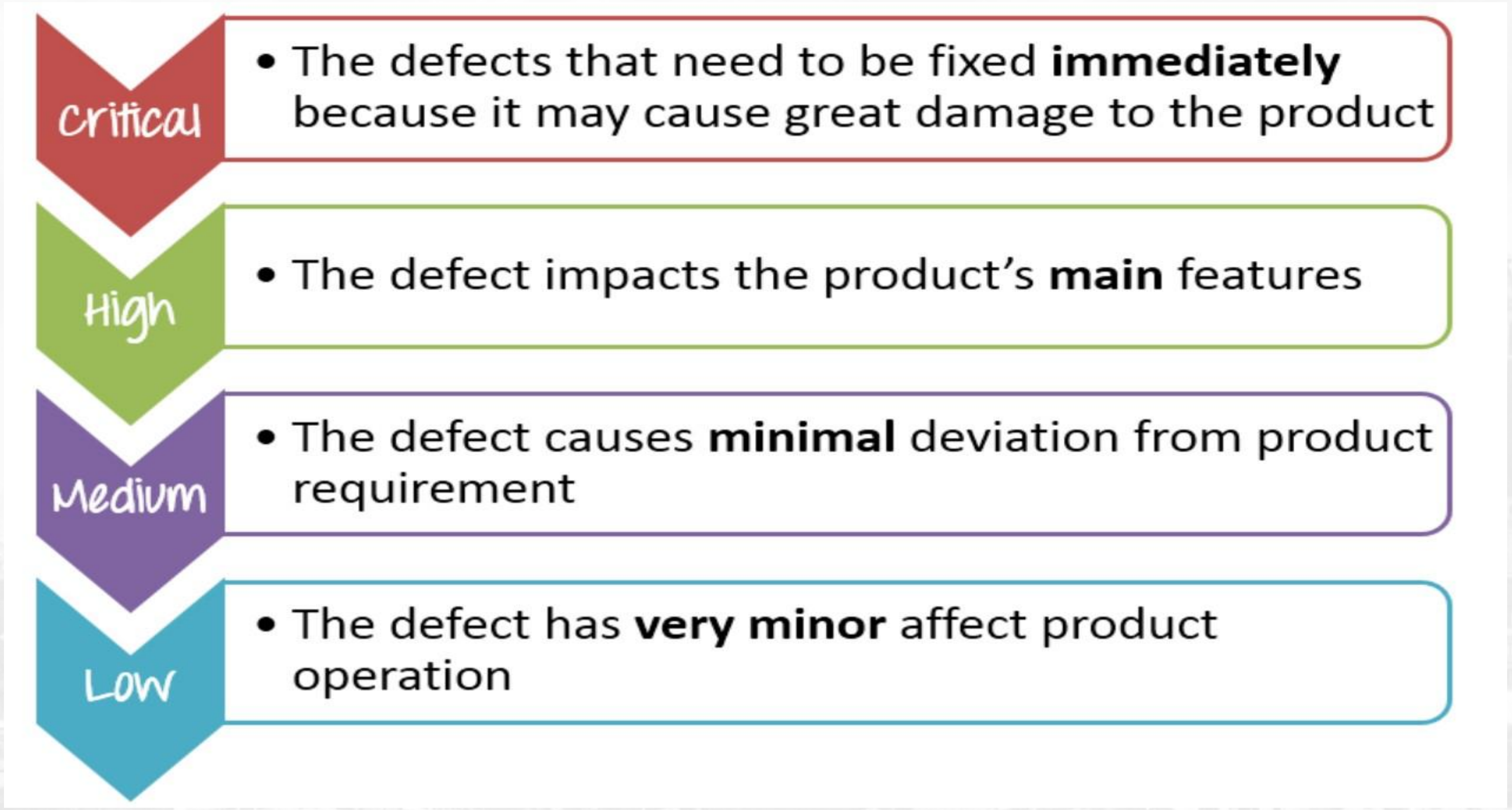

Discovery
Categorization
Resolution
Verification
Closure
Reporting

Discovery



Discover defect
• Detect the defect before releasing to customer
Report defect
• Report the defect to development team
Accept defect
• Developer accepts or rejects the bug

Categorization



- The defects that need to be fixed **immediately** because it may cause great damage to the product

- The defect impacts the product's **main** features

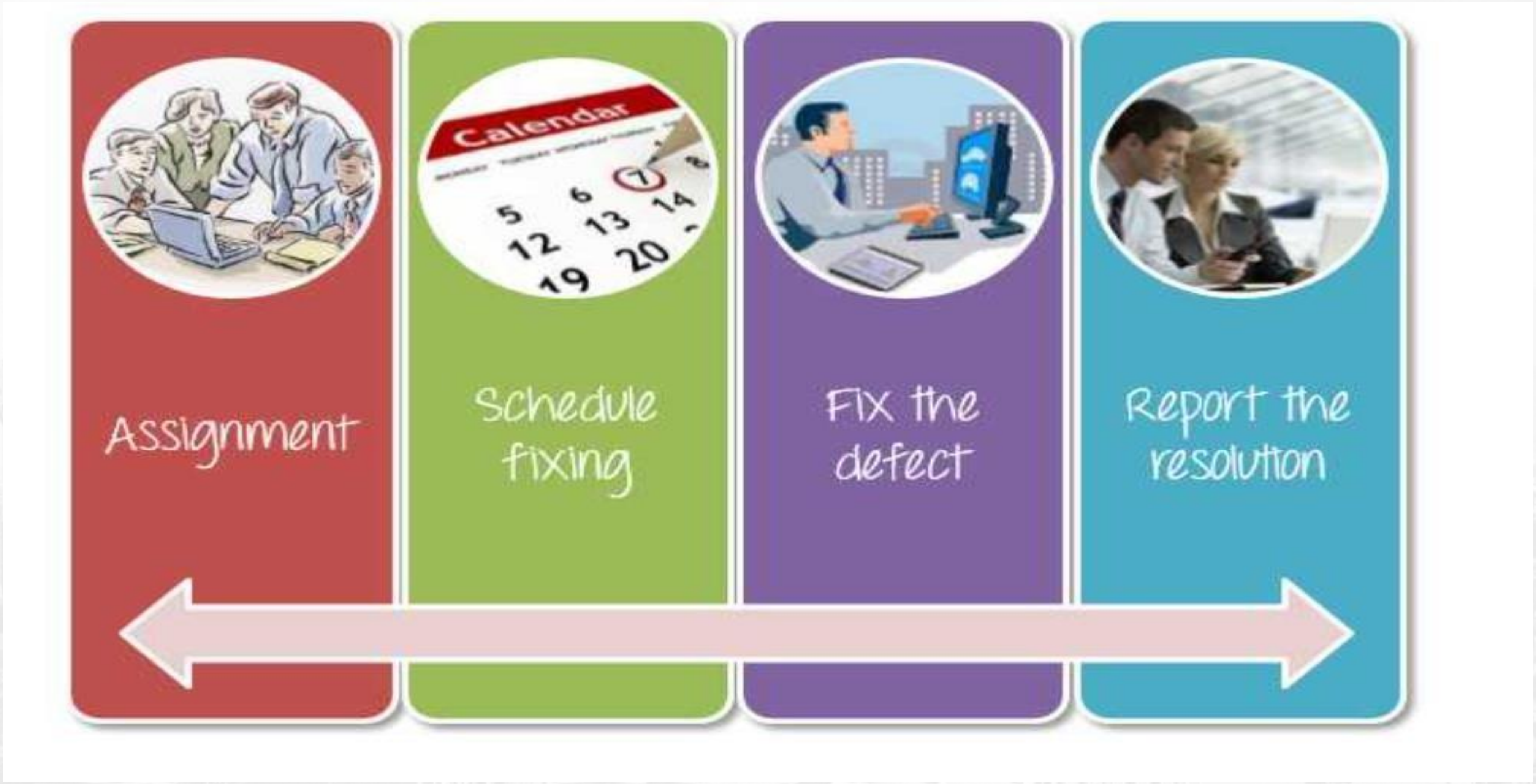
- The defect causes **minimal** deviation from product requirement

- The defect has **very minor** affect product operation

Testing Terms

- In software testing and Quality Assurance literature, a number of terms are used to describe different forms of testing.
  - White Box
  - Black Box
  - Unit
  - Regression
  - Integration
  - System
  - Performance
  - Smoke
  - Stress and so on...

Defect Resolution



- **Verification**

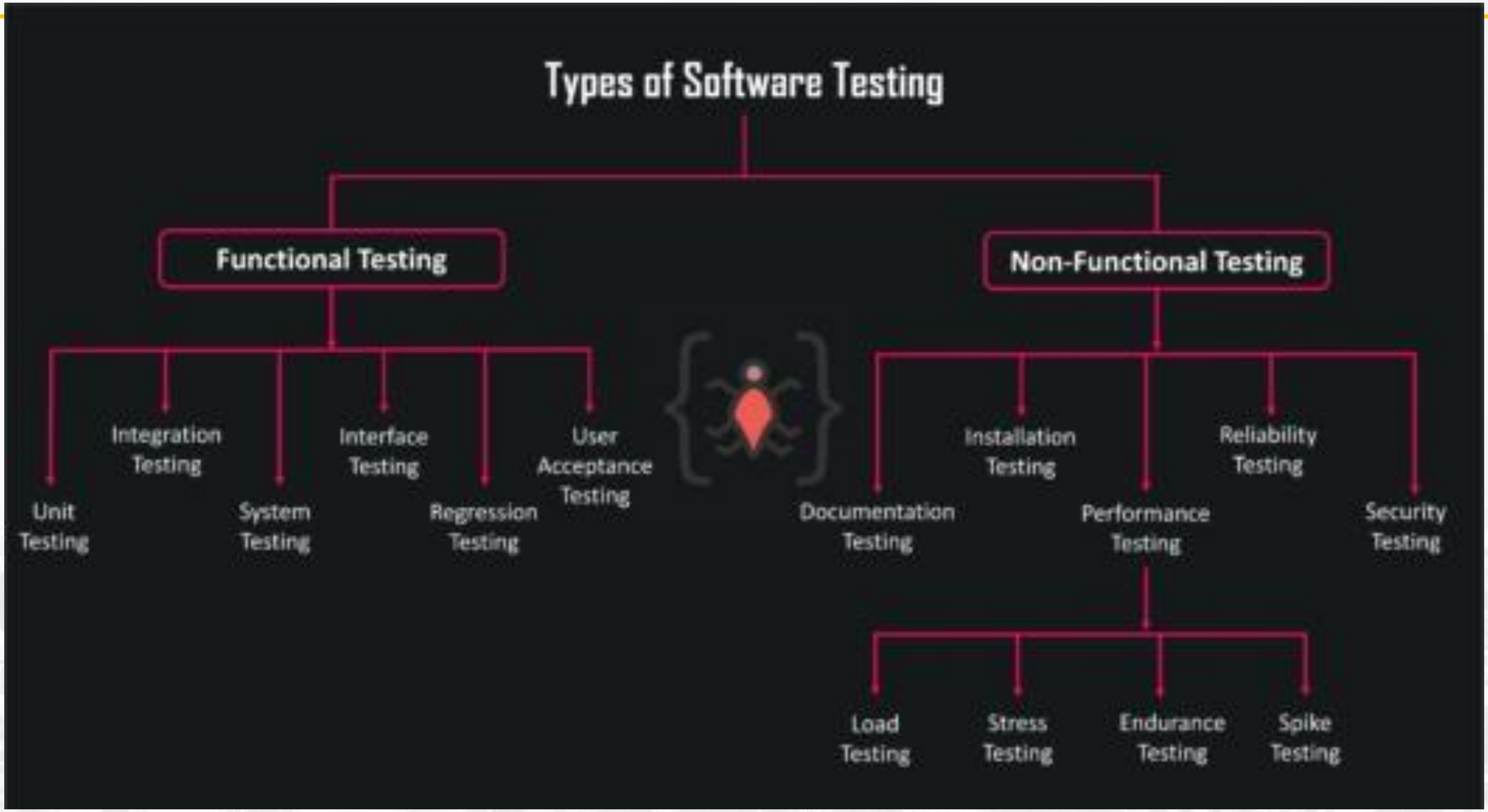
  After the development team **fixed** and **reported** the defect, the testing team **verifies** that the defects are actually resolved.

- **Closure**

  Once a defect has been resolved and verified, the defect is changed status as **closed**. If not, you have send a notice to the development to check the defect again.

Defect Reporting

- **Defect Reporting** in software testing is a process in which test managers prepare and send the defect report to the management team for feedback on defect management process and defects' status. Then the management team checks the defect report and sends feedback or provides further support if needed. Defect reporting helps to better communicate, track and explain defects in detail.
- The management board has right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report them the current defect situation to get feedback from them.


Types of Software Testing
Functional Testing
Non-Functional Testing
Unit Testing
Integration Testing
System Testing
Interface Testing
Regression Testing
User Acceptance Testing
Documentation Testing
Installation Testing
Performance Testing
Reliability Testing
Security Testing
Load Testing
Stress Testing
Endurance Testing
Spike Testing

Levels of Testing:

There are basically following testing in level of testing

- Unit Testing,
- Integration Testing
- System Testing
- Acceptance Testing

Various types of testing come under these levels.

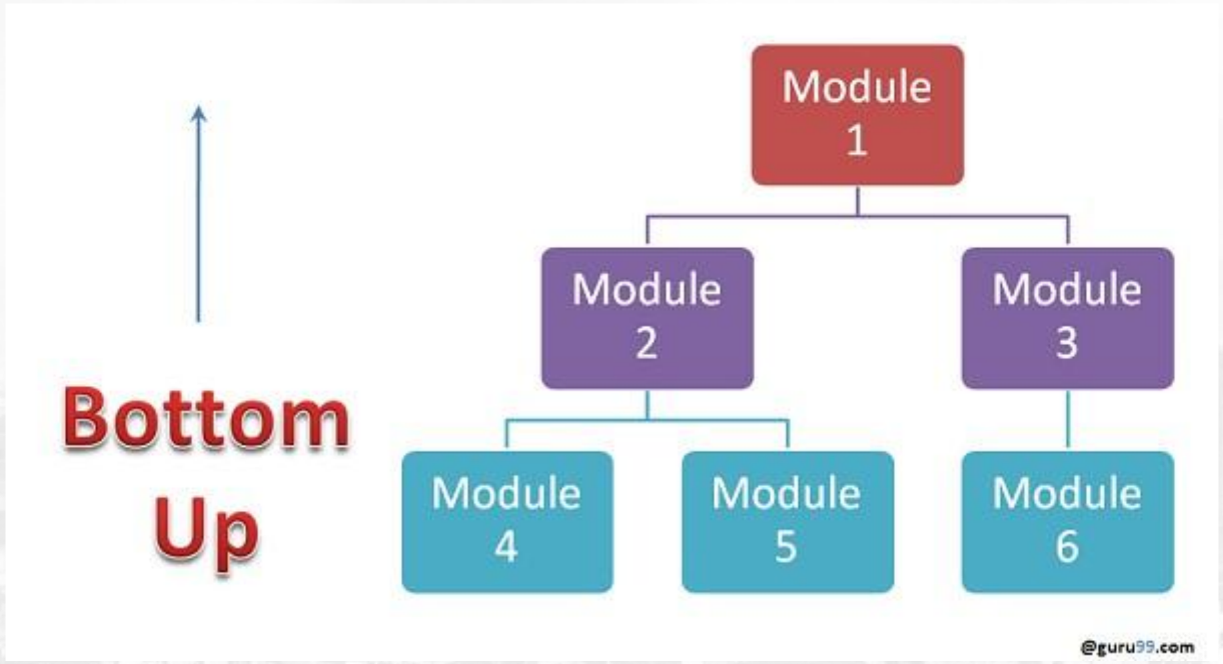
Unit Testing

- In this type of testing, errors are detected individually from every component or unit by individually testing the components or units of software to ensure that if they are fit for use by the developers. It is the smallest testable part of the software.
- The module interface is tested to ensure that information properly flows into and out of the unit under test.
- Local data structure is tested for its integrity.
- Boundary conditions are tested.
- A *driver* and a *stub* software must be developed for each unit.

Integration Testing

- A systematic way to test a program structure that consists of more than one interfacing components
- An incremental way of integration helps in identifying and isolating issues. It is easy to fix those issues if small number of components are integrated at one time.
- 2 ways to do integration testing :
  - Top down integration : Start with main program, integrate its subordinates (immediate subroutines, classes and methods) and traverse the tree of subordinates one level at a time. You can adapt “***depth first***” or “***breadth first***” approach while integrating all the subroutines. ***Regression testing*** is performed to make sure that new errors are not introduced.
  - Bottom-up integration : Start with the most atomic component, integrate other components at the same level, write a driver program to drive the functionality of integrated components, run integration tests on a ***cluster*** of components and then go one level up. Like this, drivers are removed and all components are integrated.

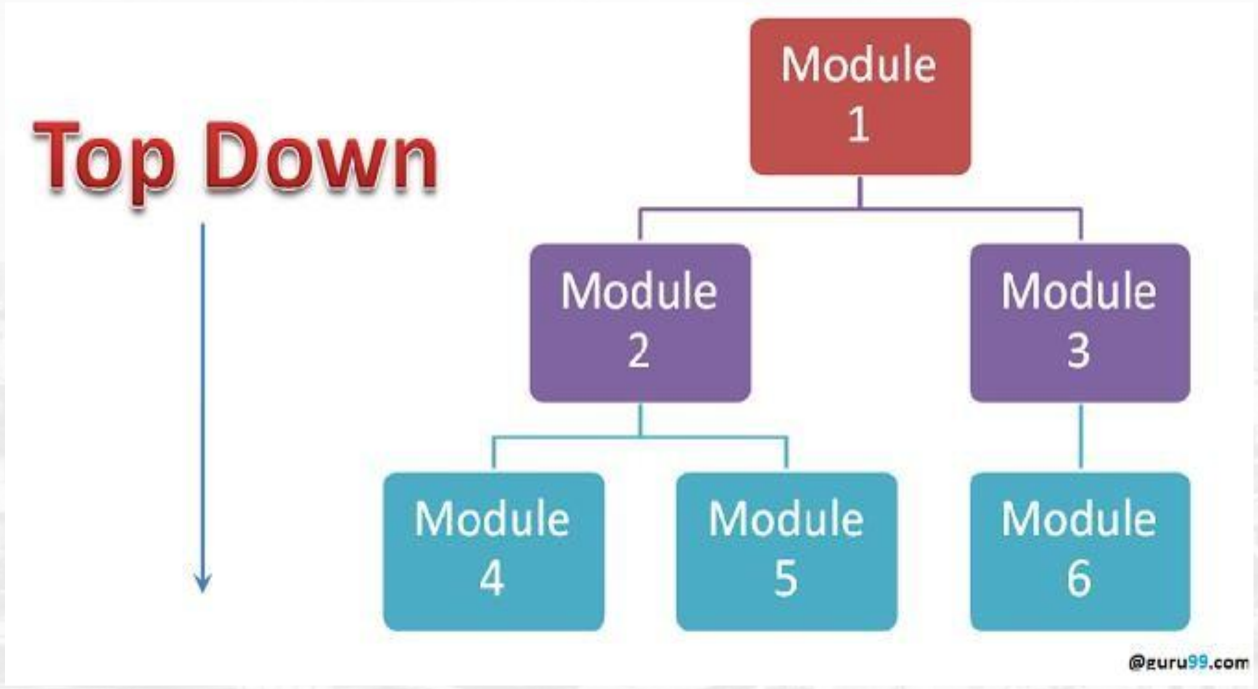
Bottom-up Integration



Cntd..

Top-down Integration:

- In Top to down approach, testing takes place from top to down following the control flow of the software system.



System Testing

- In system testing, complete and integrated Softwares are tested i.e. all the system elements forming the system is tested as a whole to meet the requirements of the system.
- Focuses on following:
  - External interfaces
  - Complex functionalities
  - Security
  - Performance
  - Installability
  - Smoothness of interaction (User experience)
  - Documentation etc.

Apha and Beta Testing

- Alpha Testing is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance tests.
- Beta Testing is performed by real users of the software application in a real environment. Beta testing is one type of User Acceptance Testing.

Alpha Testing	Beta Testing
Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black-box testing.
Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Alpha testing is performed at the developer's site.	Beta testing is performed at the end-user of the product.
Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.
Alpha testing may require a long execution cycle.	Beta testing requires only a few weeks of execution.
Developers can immediately address the critical issues or fixes in alpha testing.	Most of the issues or feedback collected from the beta testing will be implemented in future versions of the product.
Multiple test cycles are organized in alpha testing.	Only one or two test cycles are there in beta testing.

Performance Testing

- **Performance Testing** is a type of software testing that ensures software applications to perform properly under their expected workload. It is a testing technique carried out to determine system performance in terms of sensitivity, reactivity and stability under a particular workload.
- Performance testing is a type of software testing that focuses on evaluating the performance and scalability of a system or application. The goal of performance testing is to identify bottlenecks, measure system performance under various loads and conditions, and ensure that the system can handle the expected number of users or transactions.

Types of performance testingLoad testing: Load testing simulates a real-world load on the system to see how it performs under stress. It helps identify bottlenecks and determine the maximum number of users or transactions the system can handle.

- **Stress testing:** Stress testing is a type of load testing that tests the system's ability to handle a high load above normal usage levels. It helps identify the breaking point of the system and any potential issues that may occur under heavy load conditions.
- **Spike testing:** Spike testing is a type of load testing that tests the system's ability to handle sudden spikes in traffic. It helps identify any issues that may occur when the system is suddenly hit with a high number of requests.
- **Soak testing:** Soak testing is a type of load testing that tests the system's ability to handle a sustained load over a prolonged period of time. It helps identify any issues that may occur after prolonged usage of the system.
- **Endurance testing:** This type of testing is similar to soak testing, but it focuses on the long-term behavior of the system under a constant load.

Performance Testing Attributes:

- **Speed:**

  It determines whether the software product responds rapidly.
- **Scalability:**

  It determines amount of load the software product can handle at a time.
- **Stability:**

  It determines whether the software product is stable in case of varying workloads.
- **Reliability:**

  It determines whether the software product is secure or not

Advantages of Performance Testing :

- Performance testing ensures the speed, load capability, accuracy and other performances of the system.
- **Identifying bottlenecks**: Performance testing helps identify bottlenecks in the system such as slow database queries, insufficient memory, or network congestion. This helps developers optimize the system and ensure that it can handle the expected number of users or transactions.
- **Improved scalability:** By identifying the system's maximum capacity, performance testing helps ensure that the system can handle an increasing number of users or transactions over time. This is particularly important for web-based systems and applications that are expected to handle a high volume of traffic.
- **Improved reliability:** Performance testing helps identify any potential issues that may occur under heavy load conditions, such as increased error rates or slow response times. This helps ensure that the system is reliable and stable when it is deployed to production.
- **Reduced risk:** By identifying potential issues before deployment, performance testing helps reduce the risk of system failure or poor performance in production.
- **Cost-effective:** Performance testing is more cost-effective than fixing problems that occur in production. It is much cheaper to identify and fix issues during the testing phase than after deployment.
- **Improved user experience**: By identifying and addressing bottlenecks, performance testing helps ensure that users have a positive experience

Disadvantages of Performance Testing :

- Sometimes, users may find performance issues in the real time environment.
- Team members who are writing test scripts or test cases in the automation tool should have high-level of knowledge.
- Team members should have high proficiency to debug the test cases or test scripts.
- Low performances in the real environment may lead to lose large number of users
- Performance testing also has some disadvantages, which include:
- Resource-intensive: Performance testing can be resource-intensive, requiring significant hardware and software resources to simulate a large number of users or transactions. This can make performance testing expensive and time-consuming.
- Complexity: Performance testing can be complex, requiring specialized knowledge and expertise to set up and execute effectively. This can make it difficult for teams with limited resources or experience to perform performance testing.
- Limited testing scope: Performance testing is focused on the performance of the system under stress, and it may not be able to identify all types of issues or bugs. It's important to combine performance testing with other types of testing such as functional testing, regression testing, and acceptance testing.

Security Testing

- **Security Testing** is a type of software testing that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders. It ensures that the software system and application are free from any threats or risks that can cause a loss. Security testing of any system is focused on finding all possible loopholes and weaknesses of the system which might result in the loss of information or repute of the organization. Security testing is a type of software testing that focuses on evaluating the security of a system or application. The goal of security testing is to identify vulnerabilities and potential threats, and to ensure that the system is protected against unauthorized access, data breaches, and other security-related issues.

Principle of Security Testing:

- Below are the six basic principles of security testing:
- Confidentiality
- Integrity
- Authentication
- Authorization
- Availability
- Non-repudiation

White Box Testing

- **White box testing** techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. White Box Testing is also known as transparent testing, open box testing.

Working process of white box testing:

- **Input:** Requirements, Functional specifications, design documents, source code.
- **Processing:** Performing risk analysis for guiding through the entire process.
- **Proper test planning:** Designing test cases so as to cover the entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
- **Output:** Preparing final report of the entire testing process.

Testing techniques:

- **Statement coverage:** In this technique, the aim is to traverse all statement at least once. Hence, each line of code is tested. In case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.
- **Branch Coverage:** In this technique, test cases are designed so that each branch from all decision points are traversed at least once. In a flowchart, all edges must be traversed at least once.

- **Condition Coverage:** In this technique, all individual conditions must be covered as shown in the following example:
- READ X, Y
- IF(X == 0 || Y == 0)
- PRINT ‘0’
- #TC1 – X = 0, Y = 55
- #TC2 – X = 5, Y = 0

- **Multiple Condition Coverage:** In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let’s consider the following example:
  - READ X, Y
  - IF(X == 0 || Y == 0)
  - PRINT ‘0’
  - #TC1: X = 0, Y = 0
  - #TC2: X = 0, Y = 5
  - #TC3: X = 55, Y = 0
  - #TC4: X = 55, Y = 5

- **Basis Path Testing:** In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

- **Loop Testing:** Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.
  - **Simple loops:** For simple loops of size n, test cases are designed that:
    - Skip the loop entirely
    - Only one pass through the loop
    - 2 passes
    - m passes, where m < n
    - n-1 ans n+1 passes
  - **Nested loops:** For nested loops, all the loops are set to their minimum count and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.
  - **Concatenated loops:** Independent loops, one after another. Simple loop tests are applied for each. If they're not independent, treat them like nesting.

- Testers can identify defects that cannot be detected through other testing techniques.

Advantages of white box testing include:

- Testers can create more comprehensive and effective test cases that cover all code paths.
- Testers can ensure that the code meets coding standards and is optimized for performance.

Disadvantages to white box testing, such as:

- Testers need to have programming knowledge and access to the source code to perform tests
- Testers may focus too much on the internal workings of the software and may miss external issues.
- Testers may have a biased view of the software since they are familiar with its internal workings. Overall, white box testing is an important technique in software engineering, and it is useful for identifying defects and ensuring that software applications meet their requirements and specifications at the code level
- It is very expensive.

Black Box Testing

- **Black box** testing is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products.

Black box testing can be done in the following ways:

- **1. Syntax-Driven Testing –** This type of testing is applied to systems that can be syntactically represented by some language. For example- compilers, language that can be represented by a context-free grammar. In this, the test cases are generated so that each grammar rule is used at least once.
- **2. Equivalence partitioning –** It is often seen that many types of inputs work similarly so instead of giving all of them separately we can group them and test only one input of each group. The idea is to partition the input domain of the system into several equivalence classes such that each member of the class works similarly, i.e., if a test case in one class results in some error, other members of the class would also result in the same error.
- **3. Boundary value analysis –** Boundaries are very good places for errors to occur. Hence if test cases are designed for boundary values of the input domain then the efficiency of testing improves and the probability of finding errors also increases. For example – If the valid range is 10 to 100 then test for 10,100 also apart from valid and invalid inputs.

- **4. Cause effect Graphing –** This technique establishes a relationship between logical input called causes with corresponding actions called the effect. The causes and effects are represented using Boolean graphs. The following steps are followed:
  - Identify inputs (causes) and outputs (effect).
  - Develop a cause-effect graph.
  - Transform the graph into a decision table.
  - Convert decision table rules to test cases.

- **5. Requirement-based testing –** It includes validating the requirements given in the SRS of a software system.
- **6. Compatibility testing –** The test case result not only depends on the product but is also on the infrastructure for delivering functionality. When the infrastructure parameters are changed it is still expected to work properly.

Black Box Testing Type

- The following are the several categories of black box testing:
- Functional Testing
- Regression Testing
- Nonfunctional Testing (NFT)
- **Functional Testing:** It determines the system’s software functional requirements.

  Regression Testing: It ensures that the newly added code is compatible with the existing code. In other words, a new software update has no impact on the functionality of the software. This is carried out after a system maintenance operation and upgrades.

  Nonfunctional Testing: Nonfunctional testing is also known as NFT. This testing is not functional testing of software. It focuses on the software’s performance, usability, and scalability.

Advantages of Black Box Testing:

- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user’s or client’s point of view.
- Test cases are easily reproducible.
- It is used in finding the ambiguity and contradictions in the functional specifications.

Disadvantages of Black Box Testing:

- There is a possibility of repeating the same tests while implementing the testing process.
- Without clear functional specifications, test cases are difficult to implement.
- It is difficult to execute the test cases because of complex inputs at different stages of testing.

Basis Path Testing

- **Basis Path Testing** is a white box testing technique based on the control structure of a program or a module. Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing. Therefore, by definition, Basis path testing is a technique of selecting the paths in the control flow graph, that provide a basis set of execution paths through the program or module. Since this testing is based on the control structure of the program, it requires complete knowledge of the program's structure. To design test cases using this technique, four steps are followed :

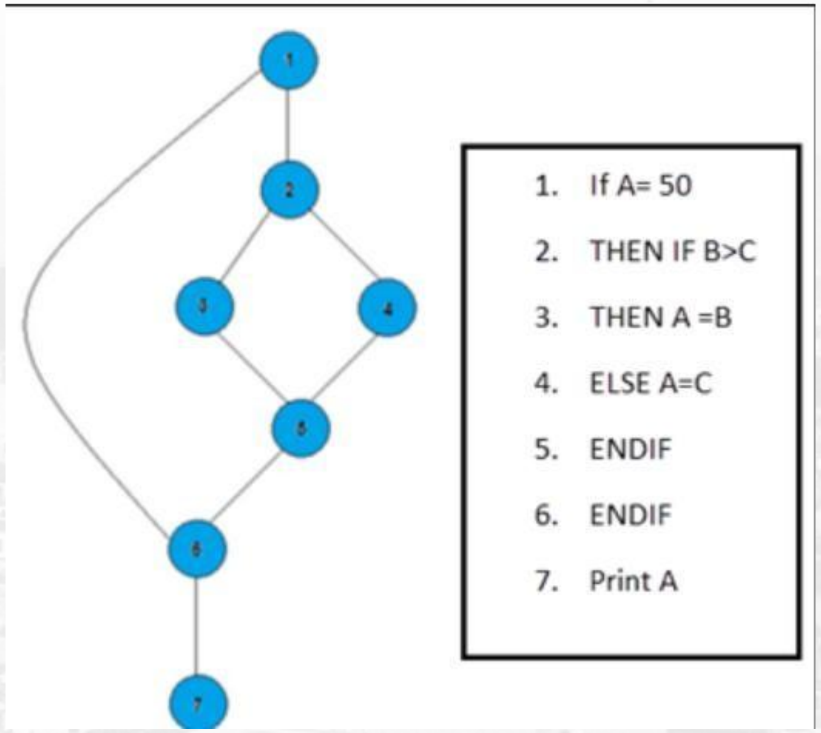
1. Construct the Control Flow Graph
2. Compute the Cyclomatic Complexity of the Graph
3. Identify the Independent Paths
4. Design Test cases from Independent Paths

Continued …

Control Flow Graph – A control flow graph (or simply, flow graph) is a directed graph which represents the control structure of a program or module. A control flow graph (V, E) has V number of nodes/vertices and E number of edges in it. A control graph can also have :

- **Junction Node –** a node with more than one arrow entering it.
- **Decision Node –** a node with more than one arrow leaving it.
- **Region –** area bounded by edges and nodes (area outside the graph is also counted as a region.).

- **Cyclomatic Complexity –** The cyclomatic complexity V(G) is said to be a measure of the logical complexity of a program. It can be calculated using three different formulae :
- V(G) = P+1
- V(G)= 2+1
- For example
- **Path 1**: 1,2,3,5,6, 7
- **Path 2**: 1,2,4,5,6, 7
- **Path 3**: 1, 6, 7

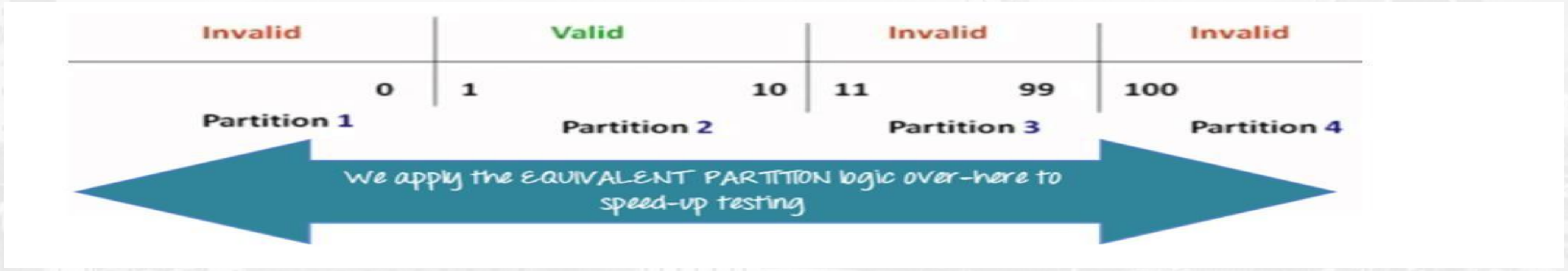


1. If A= 50
2. THEN IF B>C
3. THEN A =B
4. ELSE A=C
5. ENDIF
6. ENDIF
7. Print A

Equivalence Partitioning

- **Equivalence Partitioning** or Equivalence Class Partitioning is type of black box testing technique which can be applied to all levels of software testing like unit, integration, system, etc. In this technique, input data units are divided into equivalent partitions that can be used to derive test cases which reduces time required for testing because of small number of test cases.
- It divides the input data of software into different equivalence data classes.
- You can apply this technique, where there is a range in the input field.
- **Example 1: Equivalence and Boundary Value**
- Let’s consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.
- While value 11 to 99 are considered invalid for order and an error message will appear, **“Only 10 Pizza can be ordered”**

- The divided sets are called Equivalence Partitions or Equivalence Classes. Then we pick only one value from each partition for testing. The hypothesis behind this technique is **that if one condition/value in a partition passes all others will also pass**. Likewise**, if one condition in a partition fails, all other conditions in that partition will fail**.



Regression Testing

Each time a new module is developed, the software changes.

Regression testing is a way to make sure that the addition of new functionality has not broken existing features and functionality.

Regression test suite consists of test cases that are already developed and executed every time a major feature or module is added to the application.

Regression testing is normally done using a suite of automated test cases that includes :

- Sample of test cases that exercise all software functions
- Additional test cases that focus on functions that may be affected because of the new module or changes
- Tests that focus on software components that have been changed.

Regression tests should be chosen wisely to cover only affected features and modules.

Smoke Testing

- A suite of tests capable of “burning” the system generating “smoke” out of a nightly build
- Smoke tests are executed on nightly builds to make sure that the changes made during the previous day did not break any major functionality or the build does not contain any “showstopper” defects.
- The smoke test should
  - exercise the whole system from end to end.
  - expose major problems.
  - Be thorough enough to guarantee stability of the build for further testing.

Graph based Testing

- Graph-based testing first builds a graph model for the program under test, and then tries to cover certain elements in the graph model.
  - Graph is one of the most widely used structures for abstraction.
  - Graph is a well-defined, well-studied structure and is one of the most fundamental data structures in computer science

Major Steps

- Step 1: Build a graph model – What information to be captured, and how to represent those information?
- Step 2: Identify test requirements (TR) – A test requirement is a structural entity in the graph model that must be covered during testing
- Step 3: Select test paths to cover those requirements
- Step 4: Derive test data so that those test paths can be executed

Graph Models

- Control Flow Graph: Captures information about how the control is transferred in a program.
- Data Flow Graph: Augments a CFG with data flow information
- Dependency graph: Captures the data/control dependencies among program statements
- Cause-effect graph: Modeling relationships among program input conditions, known as causes, and output conditions, known as effects

Graph Based Testing

- Control Flow Testing and Data Flow Testing
- Control flow testing focuses on the transfer of control, while data flow testing focuses on the definitions of data and their subsequent use.
- Control flow coverage is defined in terms of nodes, edges, and paths; data flow coverage is defined in terms of def, use, and du-path.

Other Testing Terms

- Recovery Testing : What happens if the system fails or crashes? Is the data recovered properly? Does the system return to its original state?
- Security Testing : a series of tests that tries to break into the system by improper means.
- Stress Testing : a series of tests that executes the system in a way that demands resources in very high magnitude, frequency or volume. How does the system respond to such scenarios? E.g. millions of simultaneous users, gigabytes of memory, thousands of transactions etc.
- Performance Testing : Important especially in case of real time and embedded systems; performance of a fully integrated system is tested using different parameters.

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

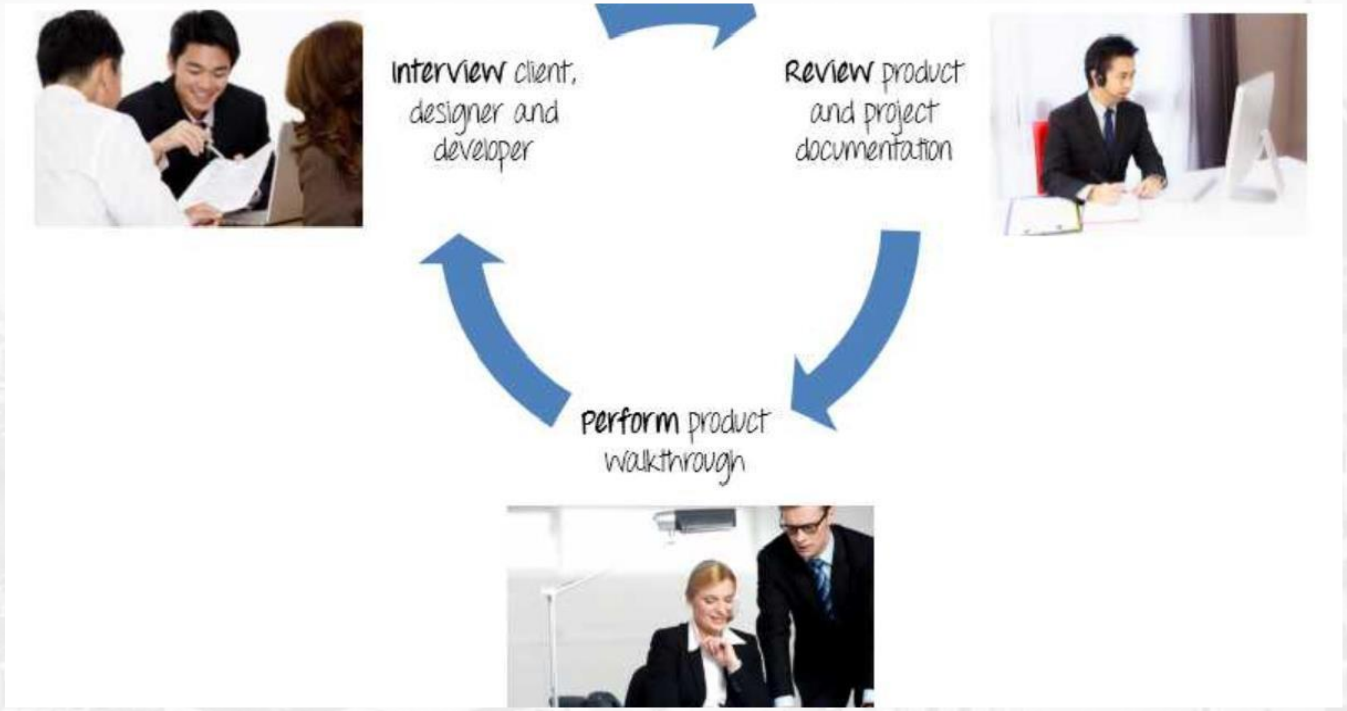
- Making Test Plan document has multiple benefits
  - Help people outside the test team such as developers, business managers, customers **understand** the details of testing.
  - Test Plan **guides** our thinking. It is like a rule book, which needs to be followed.
  - Important aspects like test estimation, test scope, Test Strategy are **documented** in Test Plan, so it can be reviewed by Management Team and re-used for other projects.

How to write a Test Plan

- Follow the seven steps below to create a test plan as per IEEE 829
  - Analyze the product
  - Design the Test Strategy
  - Define the Test Objectives
  - Define Test Criteria
  - Resource Planning
  - Plan Test Environment
  - Schedule & Estimation
  - Determine Test Deliverables

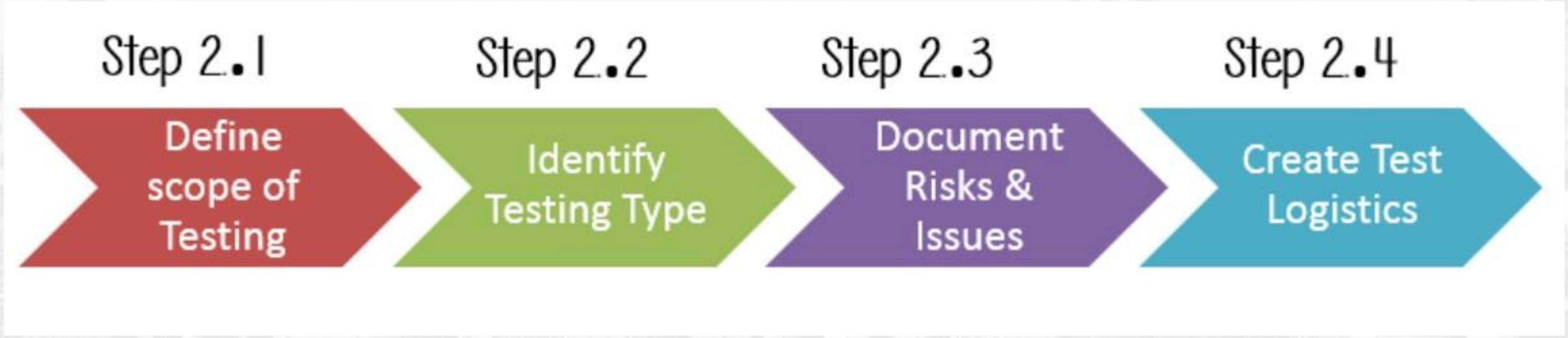
Continued …

- **Step 1) Analyze the product**
  - How can you test a product **without** any information about it? The answer is **Impossible.** You must learn a product **thoroughly** before testing it.


Interview client, designer and developer
Review product and project documentation
Perform product walkthrough

Continued …

- **Step 2) Develop Test Strategy**
- A Test Strategy document, is a high-level document, which is usually developed by Test Manager. This document defines:
  - The project's **testing objectives** and the means to achieve them
  - Determines testing **effort** and **costs**



Continued …

- **Step 3) Define Test Objective**
- Test Objective is the overall goal and achievement of the test execution. The objective of the testing is finding as many software defects as possible; ensure that the software under test is **bug free** before release.
- To define the test objectives, you should do 2 following steps
  - List all the software features (functionality, performance, GUI…) which may need to test.
  - Define the **target** or the **goal** of the test based on above features

Continued …

- **Step 4) Define Test Criteria**
- Test Criteria is a standard or rule on which a test procedure or test judgment can be based. There're 2 types of test criteria as following
- **Suspension Criteria:** Specify the critical suspension criteria for a test. If the suspension criteria are met during testing, the active test cycle will be **suspended** until the criteria are **resolved**.
- **Exit Criteria:** It specifies the criteria that denote

  a **successful** completion of a test phase. The exit criteria are the targeted results of the test and are necessary before proceeding to the next phase of development.

Continued …

- **Step 5) Resource Planning**
- Resource plan is a **detailed summary** of all types of resources required to complete project task. Resource could be human, equipment
- **Step 6) Plan Test Environment**
- A testing environment is a setup of software and hardware on which the testing team is going to execute test cases. The test environment consists of **real business** and **user** environment, as well as physical environments, such as server, front end running environment and materials needed to complete a project

- **Step 7) Schedule & Estimation**
- Making schedule is a common term in project management. By creating a solid schedule in the Test Planning, the Test Manager can use it as tool for monitoring the project progress, control the cost overruns.
- **Step 8) Test Deliverables**
- Test Deliverables is a list of all the documents, tools and other components that has to be developed and maintained in support of the testing effort.

Continued …

- There are different test deliverables at every phase of the software development lifecycle.
- Test deliverables are provided **before** testing phase.
  - Test plans document.
  - Test cases documents
  - Test Design specifications.
- Test deliverables are provided **during** the testing
  - Test Scripts
  - Simulators.
  - Test Data
  - Test Traceability Matrix
  - Error logs and execution logs.
- Test deliverables are provided **after** the testing cycles is over.

Test Scenario

- Defines test cases that cover end to end functionality of a module or part of the project.
- Compels the testers to think from user’s perspective and list down all steps required to complete certain functionality.

E.g. ATM Money Withdrawal scenario contains following test cases:

i. Swipe ATM card

ii. Enter correct PIN

iii. Select “Withdrawal” menu

iv. Enter amount

Each of these steps may lead to multiple test cases.

Test Case

- A section of the test plan document that explains:
  - Feature/functionality to be tested
  - Required input(s)
  - Expected output
  - Additional steps
  - Preconditions (system state before test execution)
  - Postconditions(system state after test execution)

–Error/Exception conditions

Etc.

- Writing correct, complete and enough number of test cases ensures consistency of test execution, better test coverage and less dependence on certain team members.

Types of Test Cases

- Functional
- Integration
- System
- Performance
- Security

And so on and so forth

Characteristics of a good test case

- Accurate
- Essential
- Economical
- Traceable
- Repeatable
- Reusable
- Neither too simple nor too complex
- Has reasonable probability of catching an error

Test case Attributes

- Test case ID : should be unique with easily understandable naming convention. E.g. UI001, DB002 etc.
- Description : Brief and concise
- Preconditions : Steps to be completed before the test case is executed. E.g. ATM PIN change test case requires previous PIN
- Steps : Write each step with proper numbering and description. The steps should be written from end user's perspective.
- Test data : Gather all required test data to provide as input. E.g. ATM PIN change requires card number and previous PIN.
- Expected Result : What should be returned to the user as a result of this test case? E.g. "Login successful", "PIN changed successfully" etc.
- Actual Result : After the test case is executed, what is the actual output?

Test case : Other Attributes

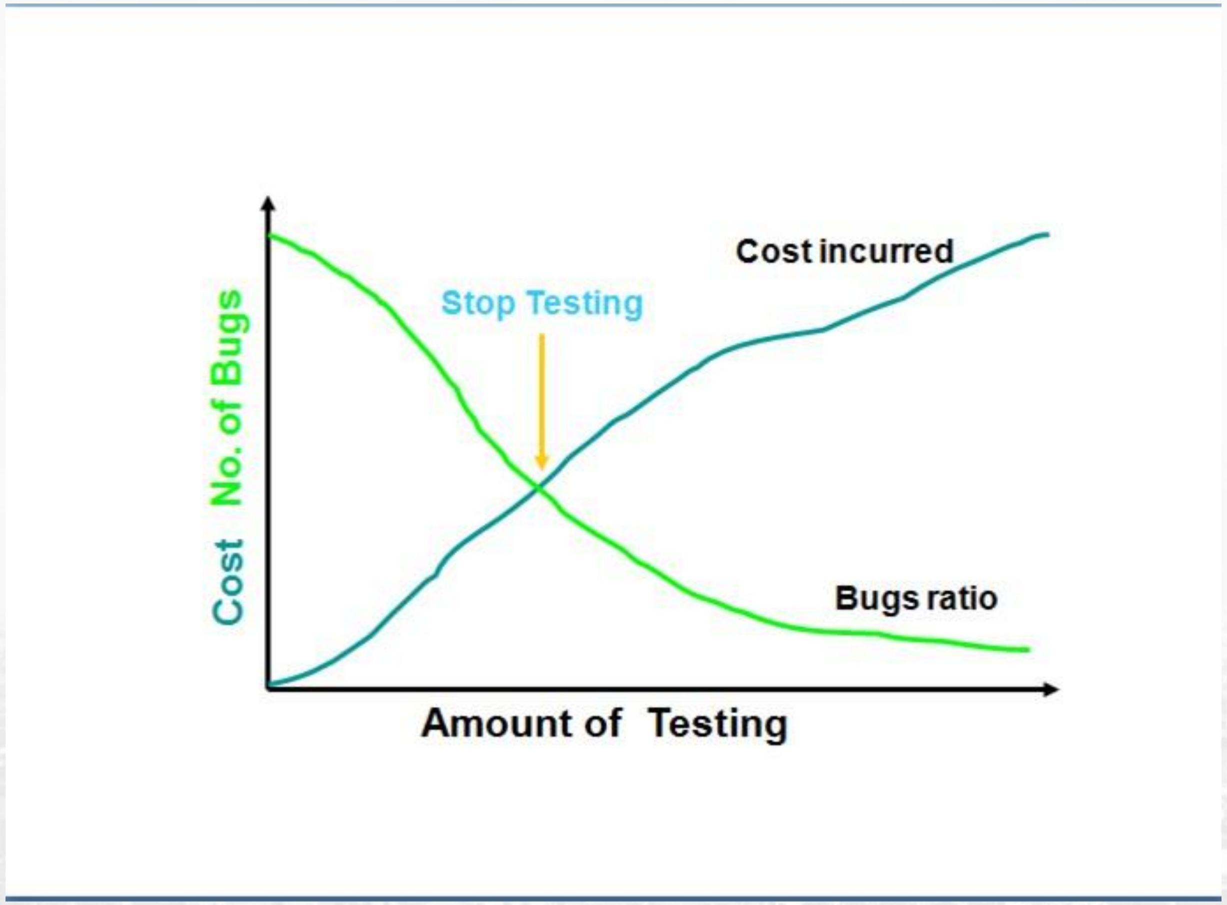
Other important fields of a test case template:

- **Project Name:** Name of the project the test cases belongs to
- **Module Name:** Name of the module the test cases belongs to
- **Reference Document:** Mention the path of the reference documents (if any such as Requirement Document, Test Plan, Test Scenarios, etc.,)
- **Created By:** Name of the Tester who created the test cases
- **Date of Creation:** When the test cases were created
- **Reviewed By:** Name of the Tester who reviewed the test cases
- **Date of Review:** When the test cases were reviewed
- **Executed By:** Name of the Tester who executed the test case

Test Case Execution Flow

- Generate Test Plan, Test Scenarios and Test Cases->Execute Test cases->Verify Results->Store Test Logs->Generate Error Report for failed test cases->Track defects
- Errors injected into the system lead to faults which in turn may lead to system failure
- It is always in the best interest of everyone to detect the errors earlier in the testing cycle.

When to stop testing?



Test manager considers following factors:

1. Testing and release deadlines
2. Pass percentage of test cases
3. Requirements, functionality and code coverage reaches a certain point.
4. Bug rate falls below a certain level.
5. Alpha or Beta testing period ends.
6. Testing budget is depleted.

Software Test Metrics

- Test Metrics quantify and justify the amount of testing performed in the project.
- Some important test metrics are:
  - Test Coverage = Number of units tested/total size of the system
  - Test cost (in %) = (Cost of testing/total cost)*100
  - Effectiveness of testing to business= loss due to problems / total resources processed by the system
  - Quality of testing = number of defects found during testing /(no of defects found during testing + no. of acceptance defects found after delivery) * 100
  - Achieving budget = Actual cost of testing /Budgeted cost of testing
  - Defect Density = number of defects/size

Test cases

A test case is typically defined as a document that lays out the following:

- Test data
- Procedures/inputs
- Scenarios
- Descriptions
- Testing environment
- Expected results
- Actual results

Cntd.

Check Login Functionality there many

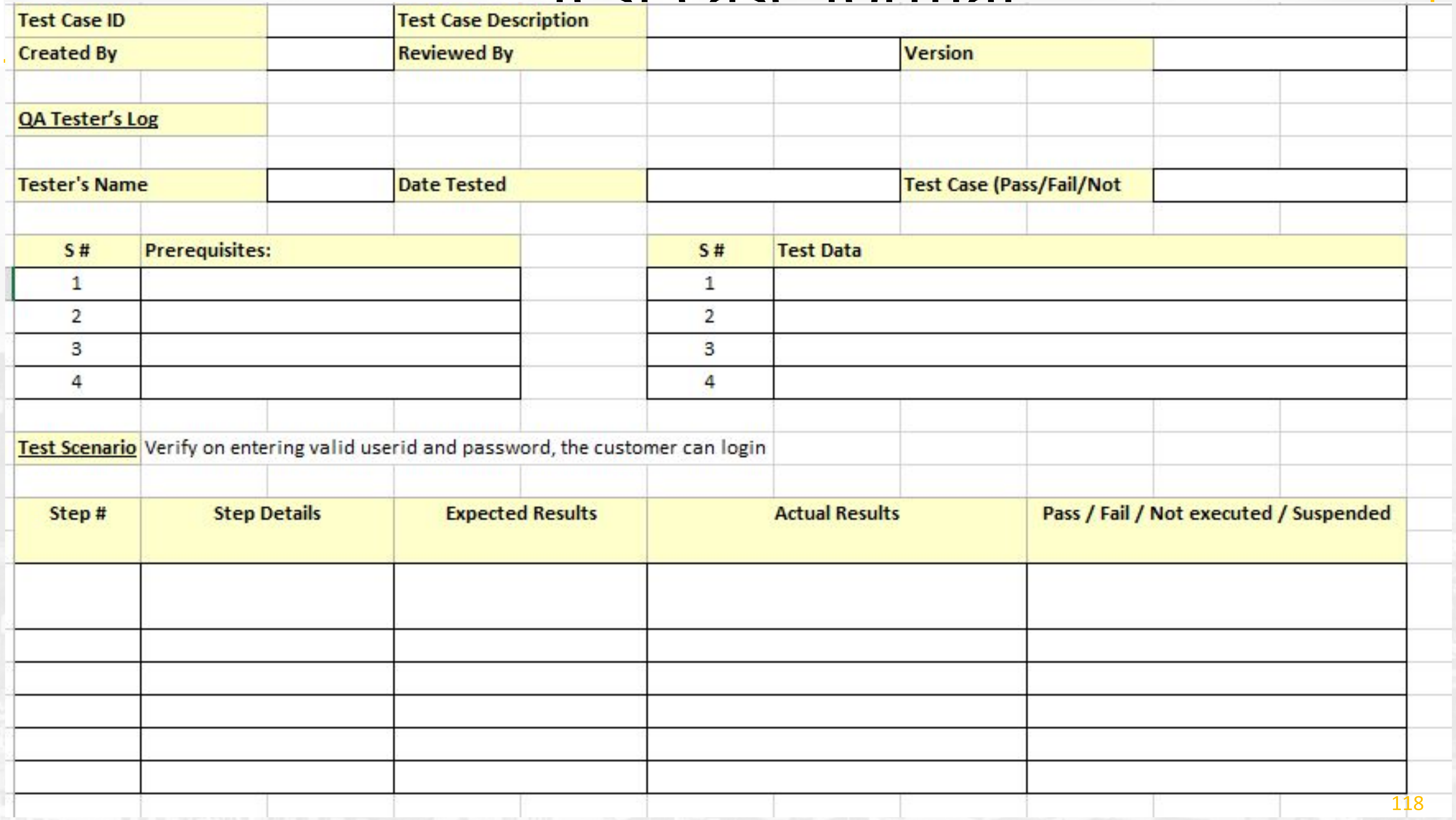
- Test Case 1: Check results on entering valid User Id & Password
- Test Case 2: Check results on entering Invalid User ID & Password
- Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more possible test cases are:

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass/Fail

Cntd.

Test Case ID #	Test Case Scenario	Test Step	Test Data	Expected Result	Actual Result	Pass/Fail
	Customer Login with valid Data	Enter UserId Enter Password Click Submit	Password = pass99	Login into an application		
TU02	Check Customer Login with invalid Data	Go to site Enter UserId Enter Password Click Submit	Userid = guru99 Password = glass99	User should not Login into an application	As Expected	Pass

Test case format



Test Case ID		Test Case Description			
Created By		Reviewed By		Version	

QA Tester's Log

Tester's Name		Date Tested		Test Case (Pass/Fail/Not	

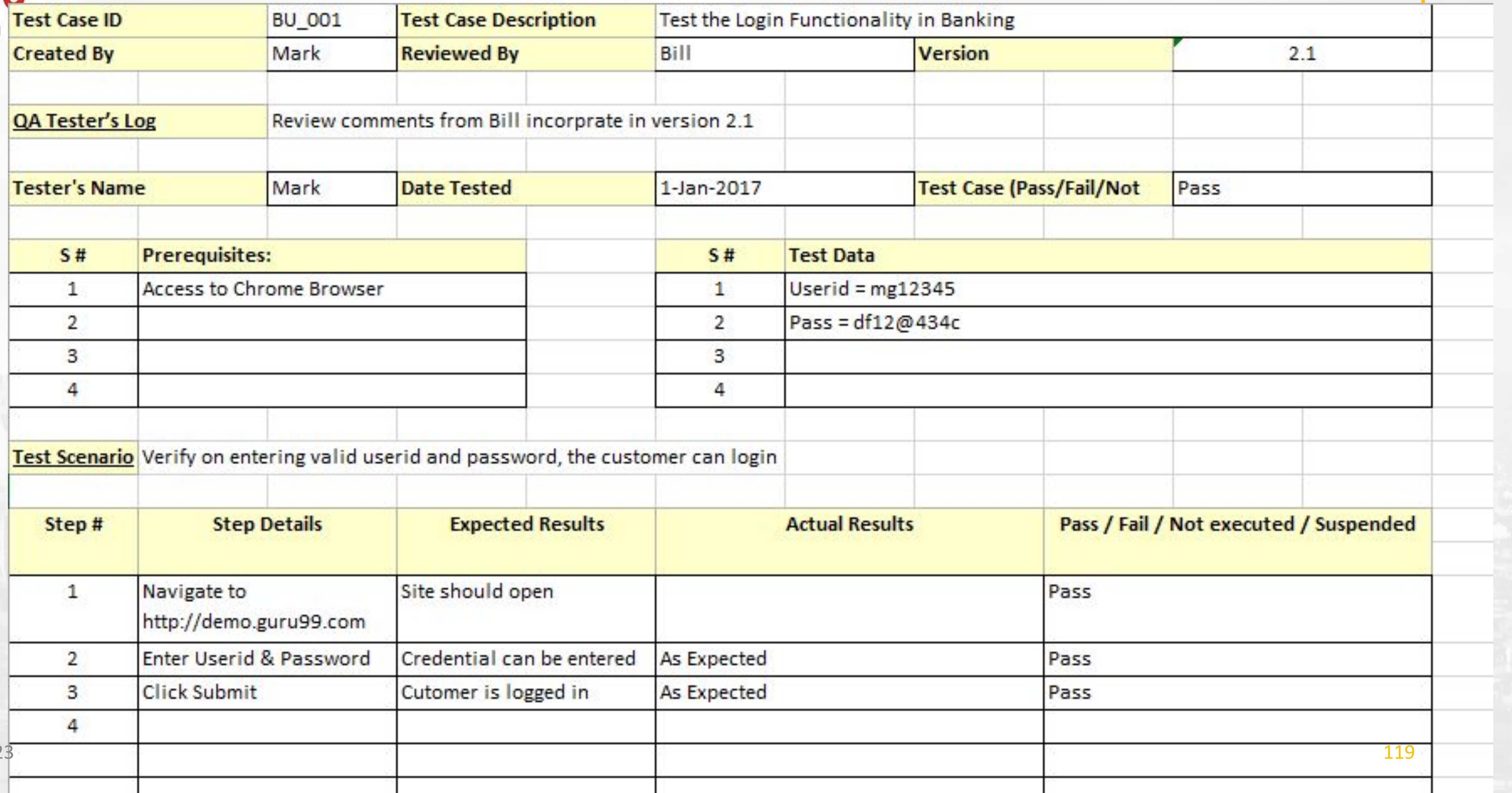
S #	Prerequisites:
1	
2	
3	
4	

S #	Test Data
1	
2	
3	
4	

Test Scenario Verify on entering valid userid and password, the customer can login

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended

Test Case format (example)



Test Case ID	BU_001	**Test Case Description**	Test the Login Functionality in Banking		
Created By	Mark	**Reviewed By**	Bill	**Version**	2.1

QA Tester's Log	Review comments from Bill incorprate in version 2.1

Tester's Name	Mark	**Date Tested**	1-Jan-2017	**Test Case (Pass/Fail/Not**	Pass

S #	**Prerequisites:**
1	Access to Chrome Browser
2	
3	
4	

S #	**Test Data**
1	Userid = mg12345
2	Pass = df12@434c
3	
4	

Test Scenario	Verify on entering valid userid and password, the customer can login

Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Navigate to http://demo.guru99.com	Site should open		Pass
2	Enter Userid & Password	Credential can be entered	As Expected	Pass
3	Click Submit	Cutomer is logged in	As Expected	Pass
4				