MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Software Testing

UNIT IV

S.Y. Semester IV

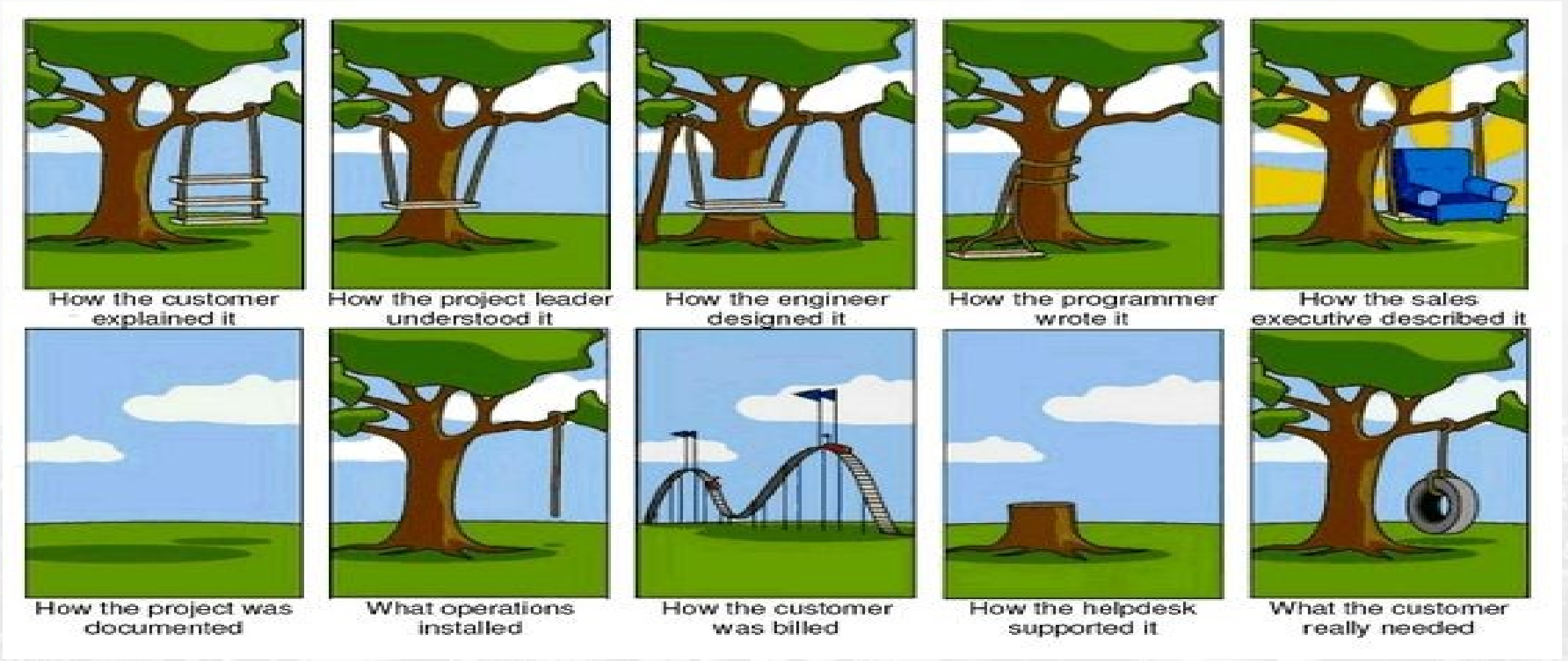
Overview

- 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



Why Software Testing?

- We all make mistakes; some trivial and some expensive or life-threatening!
- Murphy's first law : If anything can go wrong, it will!
- The primary purpose of software testing is to find out faults and failures from the application before, during and after the installation/deployment.
- Modern software development methodologies such as agile expect the involvement of testers at every step of the way:
  - user story refinement,
  - design reviews,
  - unit, integration and regression testing during development,
  - final verification before release

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.

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

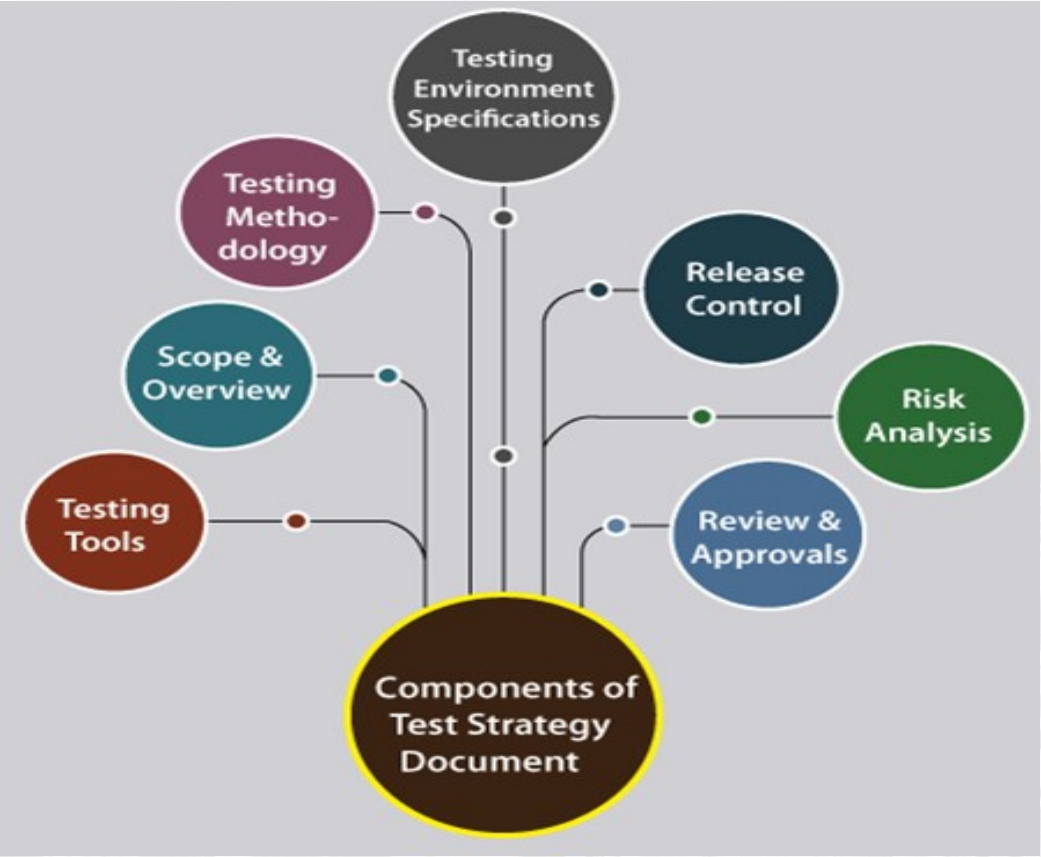
Software Testing : What is involved?

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.

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…

White Box Testing

- Test case design method that uses the software design document as a base and derives test cases that :
  - Guarantee that all independent paths within a module have been exercised at least once
  - Exercise all logical decisions on True and False conditions
  - Execute all loops at boundaries and within the operational bounds
  - Exercise internal data structures to ensure their validity.
- We need white box testing because :
  - Boundary conditions, irregular paths and exceptions need to be tested,
  - Some design errors may surface only after all logical paths are tested,
  - Typographical errors are random and may lead to incorrect behavior.

Black Box Testing

- Also called as Behavioral testing
- Focuses on functional requirements
- Complimentary to white box testing and may bring out a different set of errors
- Black box testing is required because it attempts to find out errors in following categories :
  - Incorrect or missing functions,
  - Interface errors,
  - Errors in data structures or database access,
  - Behavior and performance errors,
  - Initialization and termination errors
- Unlike White box testing, black box testing is started later in the SDLC.

Unit Testing

- Meant to test smallest unit of software program
- 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.
- A driver is a main program whereas a stub is a subprogram that may be called by the module or unit under test.
- Unit testing of highly cohesive components is easier because the components perform a very limited set of operations (or a single operation).

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.

System Testing

- Testing of the system as a whole
- Normally performed after integration testing
- Focuses on following:
  - External interfaces
  - Complex functionalities
  - Security
  - Performance
  - Installability
  - Smoothness of interaction (User experience)
  - Documentation etc.

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.

Validation Testing

- After integration test is complete, the software is completely assembled, packaged and made ready for delivery.
- Validation testing is performed on this package before it is released to the general public.
- Validation testing consists of :
  - Configuration review or audit : to make sure the software is packaged with all necessary artifacts
  - Alpha and Beta testing : Alpha testing done at developer's site with at least one customer "looking over the shoulder" of the development team; Beta testing done at one or more customer sites; giving those select customers an initial taste of the software.
- Customers record all problems encountered during validation testing; developers have a last chance to fix some of them before the software is declared as "GA" (general availability).

Basis Path Testing

- A type of “White Box” testing where all control paths in a given piece of code are explored.
- The steps include :
  - Construct the control flow graph.
  - Compute the cyclomatic complexity of the graph.
  - Identify the independent paths.
  - Design test cases based on the independent paths.

Equivalence Testing

- A variety of hypothesis tests where the “null” hypothesis is an interesting but common phenomenon whereas alternate hypothesis is a less occurring effect.
- Observed data is tested against the null hypothesis and its “equivalence bounds”.
- Normally used in clinical trials. E.g. to prove that a new cheaper drug works as effectively as an existing drug.

Graph based Testing

- The requirements are translated into a graph model.
- From the graph all paths to cover are identified.
- Test data required to cover the paths is generated.
- Test cases are executed using the test data.

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

- Formal document used to define scope of testing and related activities
- Uses the information from SRS, use case documents and other product related documents.
- Developed by QA Manager and/or QA leads
- Dynamic document that evolves during project implementation

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
v. Collect cash
vi. Respond to the prompt for "do you want to print the receipt?"
vii. Collect ATM card

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?
- Status : Based on the difference between expected and actual result, the status should be set to "Pass" or "Fail".

Test case : Other Attributes

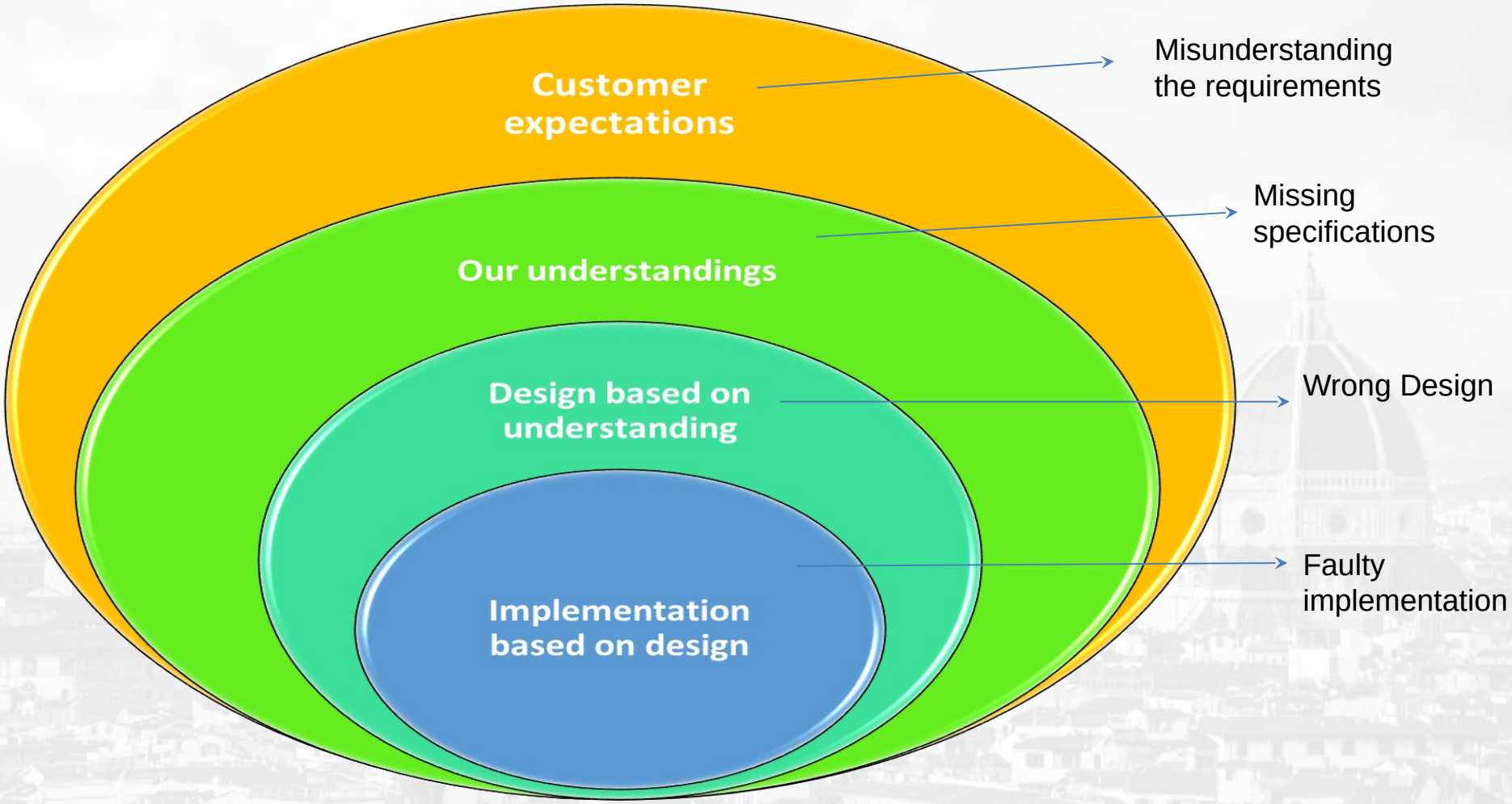
Other important fields of a test case template:

- **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
- **Date of Execution:** When the test case was executed
- **Comments:** Include any other information which helps the team

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.

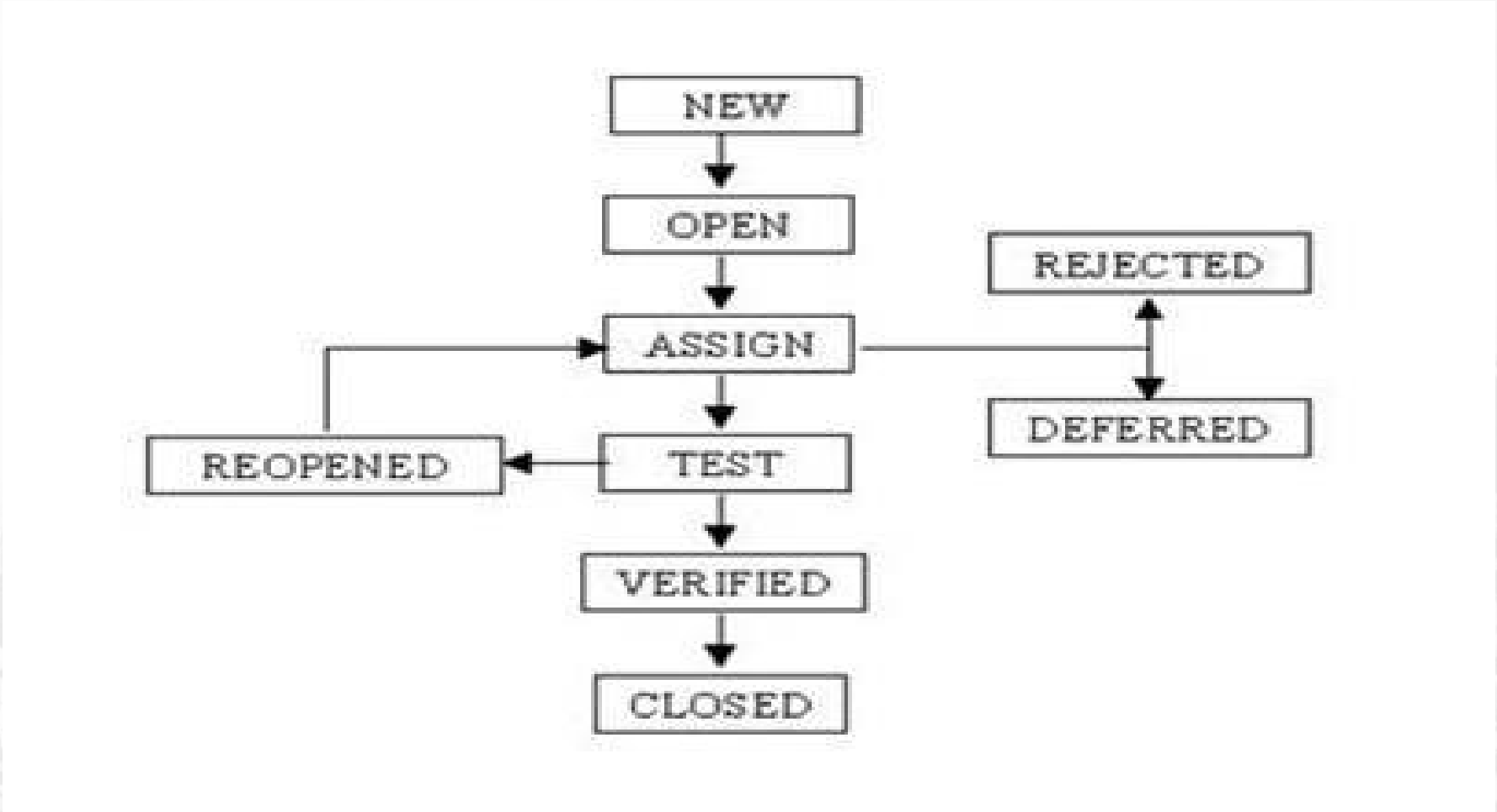
Reasons for Faults and Failures



Defect Management System

- A defect is a failure to meet expectations
- A defect management system is a repository of all types of errors, faults and failures of the system.
- It allows the testers and developers to:
  - Create new defects
  - Add steps to reproduce/explain the error condition
  - Assign defects to appropriate team member
  - Mark the defects as fixed
  - Add or assign test cases to verify the fix
  - Execute the test cases and mark the defect as fixed
  - Close the defect

Defect Lifecycle

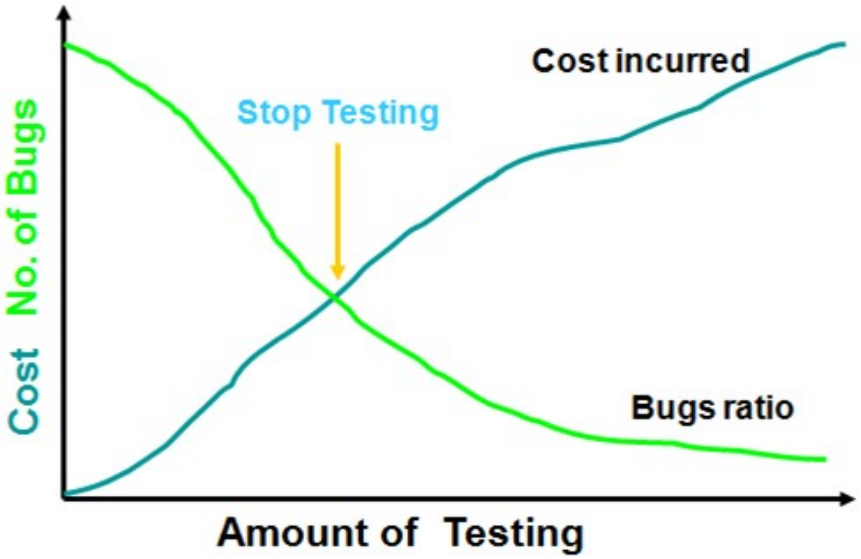


Defect Priority and Severity

- Defect Priority provides a perspective for the order of the defect fixes. Priority tells us how soon the defect must be fixed. Priority can be divided into P1, P2, P3,P4 and so on.
  - P1 – Fix the defect on highest priority, fix it before the next build
  - P2 – Fix the defect on high priority before the next test cycle
  - P3 – Fix the defect on moderate priority when time permits , before the release
  - P4- Postpone the defect for the next release or live with this defect.

- Defect Severity provides the perspective of the impact of that defect in product functionality.
  - Defect Severity levels can be:
  - S1,S2,S3, S4 or Extreme, Critical, important, Minor, Cosmetic
  - Extreme – Product Crashes or is unusable
  - Critical - Basic functionality of the product is not working.
  - Important- Extended functionality of the product not working
  - Minor – Product behaves differently
  - Cosmetic – GUI related

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