Software Testing

Unit 3

Black Box Testing & Grey Box Testing

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

Black Box Testing

List of Contents

Department of Computer Science and Engineering

What is Black Box Testing?

- Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding.
- The primary source of black box testing is a specification of requirements that is stated by the customer.
- Black box testing is done
  - With only the functional knowledge of the system
  - Without the knowledge of the internals of the system under test.

Black Box Testing

Input → Output
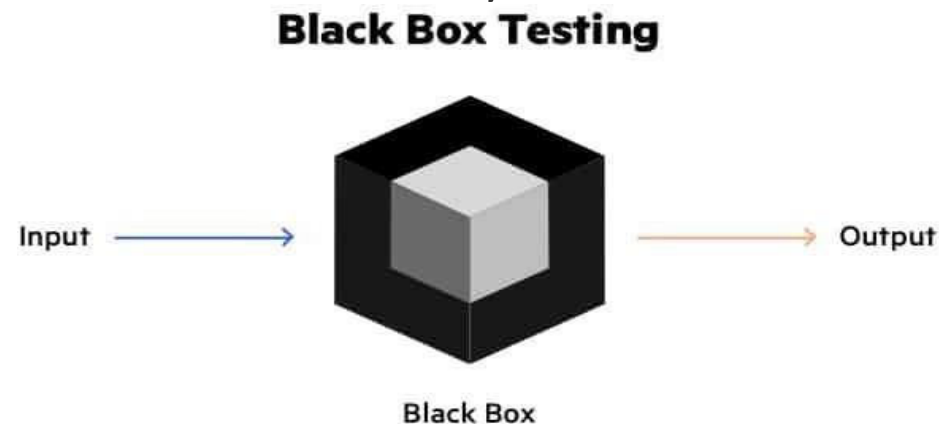
Black Box

Generic Steps Of Black Box Testing

1. The black box test is based on the specification of requirements, so it is examined in the beginning.
2. In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
3. In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
4. The fourth phase includes the execution of all test cases.
5. In the fifth step, the tester compares the expected output against the actual output.
6. In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

What is a Test Case

- Test cases are created considering the specification of the requirements.
- These test cases are generally created from working descriptions of the software including requirements, design parameters, and other specifications.
- For the testing, the test designer selects both positive test scenario by taking valid input values and adverse test scenario by taking invalid input values to determine the correct output.

Types of Black Box Testing

There are many types of Black Box Testing but the following are the prominent ones –

- **Functional testing** – This black box testing type is related to the functional requirements of a system; it is done by software testers.
- **Non-functional testing** – This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
- **Regression testing** – Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Reference for Regressing Testing

Tools used for Black Box Testing:

- Tools used for Black box testing largely depends on the type of black box testing you are doing.
- For Functional/ Regression Tests you can use – QTP, Selenium
- For Non-Functional Tests, you can use – LoadRunner, Jmeter

Selenium Tutorial

JMeter Tutorial

Differences b/w Black Box and White Box Testing

Black Box Testing	White Box Testing
It is a testing method without having knowledge about the actual code or internal structure of the application.	It is a testing method having knowledge about the actual code and internal structure of the application.
This is a higher level testing such as functional testing.	This type of testing is performed at a lower level of testing such as Unit Testing, Integration Testing.
It concentrates on the functionality of the system under test.	It concentrates on the actual code – program and its syntax's.
Black box testing requires Requirement specification to test.	White Box testing requires Design documents with data flow diagrams, flowcharts etc.
Black box testing is done by the testers.	White box testing is done by Developers or testers with programming knowledge.

When do we do Black Box Testing?

- Unlike traditional white box testing, black box testing is beneficial for testing software usability.
- The overall functionality of the system under test
- Black box testing gives you a broader picture of the software.
- This testing approach sees an application from a user's perspective.
- To test the software as a whole system rather than different modules.

Advantages of Black Box Testing

- Well suited and efficient for large code segments.
- Code access is not required.
- Clearly separates user's perspective from the developer's perspective through visibly defined roles.
- Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.

Techniques for Black Box Testing

- Requirements Based Testing

- Positive and Negative Testing

- Specification Based Testing

    - Equivalence partitioning

    - Boundary Value analysis

    - Decision Tables

    - State Based Testing

Requirements-Based Testing

- Done to ensure that all requirements in SRS are tested
- Difference between implicit and explicit requirements
- Review requirements first to ensure they are consistent, correct, complete and testable
- Review enables translation of (some of) the implied requirements to stated requirements
- A reviewed SRS tabulates requirements, along with a requirements id and a priority
- This is the genesis of a **Requirements Traceability Matrix**

Positive and Negative Testing
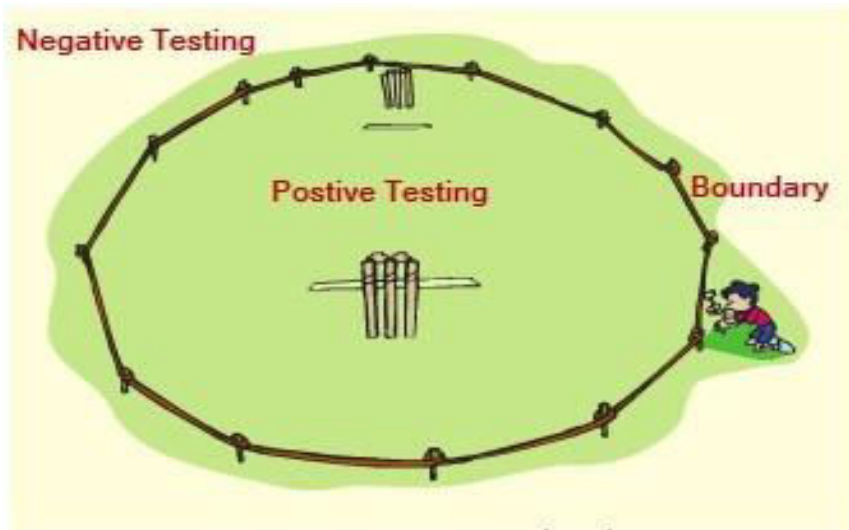
Positive Testing

- Type of testing which is performed on a software application by providing the valid data sets as an input.
- It checks whether the software application behaves as expected with positive inputs or not.
- Positive testing is performed in order to check whether the software application does exactly what it is expected to do.

Negative Testing

- Testing method performed on the software application by providing invalid or improper data sets as input.
- The purpose of negative testing is to ensure that the software application does not crash and remains stable with invalid data inputs.

Positive and Negative Testing

Illustration

Example for Positive and Negative Testing

Example 2 – Positive and Negative Testing

Positive Testing

Test Case ID	Req ID	Input 1	Input 2	Current State	Expected State
Lock_1	BR-01	- Use key 1234	- Turn clockwise	Unlocked	Locked
Lock_2	BR-02	- Use key 1234	- Turn anti-clockwise	Locked	Unlocked

Negative Testing

Test Case ID	Req ID	Input 1	Input 2	Current State	Expected State
Lock_5	BR-04	- Use key 4567	- Turn clockwise	Unlocked	Unlocked
Lock_6	BR-05	- Use Hair-pin	- Turn anti-clockwise	Locked	Locked

Specification-Based Testing

- A specification can be anything like a written document, collection of use cases, a set of models or a prototype.

- Specification-based testing technique is also known as 'black-box' or input/output driven testing techniques because they view the software as a black-box with inputs and outputs.

- Specification-based techniques are appropriate at all levels of testing (component testing through to acceptance testing) where a specification exists.
  - For example, when performing system or acceptance testing, the requirements specification or functional specification may form the basis of the tests.

Types of Specification Based Testing Techniques

1. **Equivalence Partitioning**: Software Testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived.

2. **Boundary Value Analysis**: Software Testing technique in which tests are designed to include representatives of boundary values in a range.

3. **Decision Tables:** Software Testing technique in which tests are more focused on business logic or business rules. A decision table is a good way to deal with combinations of inputs. *

4. **State Transitioning**: Software Testing technique which is used when the system is defined in terms of a finite number of states and the transitions between the states is governed by the rules of the system.
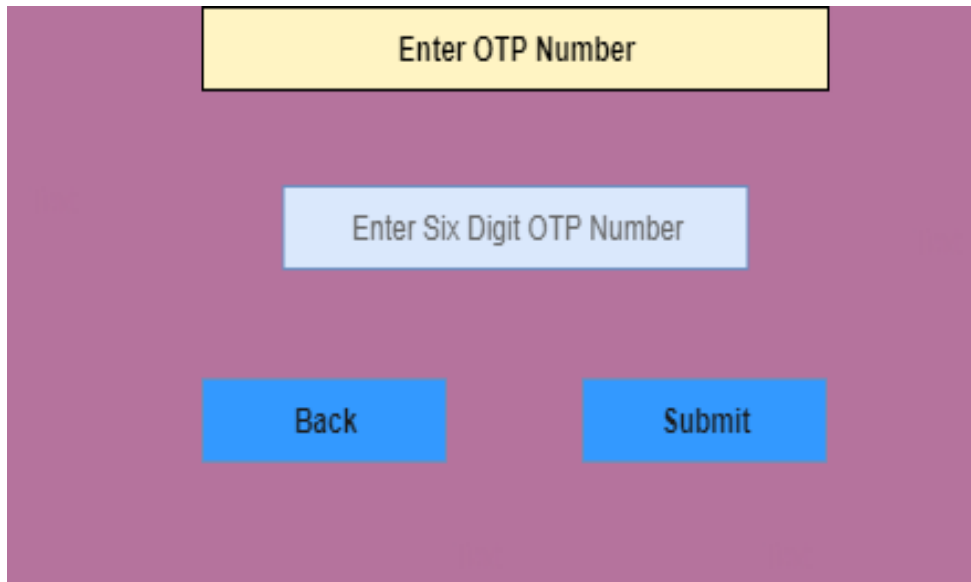
Equivalence Partitioning

- Equivalence partitioning is a technique of software testing in which input data is divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.

- If a condition of one partition is true, then the condition of another equal partition must also be true, and if a condition of one partition is false, then the condition of another equal partition must also be false.

- The principle of equivalence partitioning is, test cases should be designed to cover each partition at least once. Each value of every equal partition must exhibit the same behavior as other.

Equivalence Partitioning - Example 1

- Assume that there is a function of a software application that accepts a particular number of digits, not greater and less than that particular number.
- For example, an OTP number which contains only six digits, less or more than six digits will not be accepted, and the application will redirect the user to the error page.

INVALID	INVALID	VALID	VALID
1 Test case	2 Test case	3 Test case	
DIGITS >=7	DIGITS<=5	DIGITS = 6	DIGITS = 6
93847262	9845	456234	451483

Equivalence Partitioning - Example 2

- As present in the image, the "AGE" text field accepts only numbers from 18 to 60. There will be three sets of classes or groups.

- Two invalid classes will be:
  a) Less than or equal to 17.
  b) Greater than or equal to 61.

Equivalence Class Partitioning (ECP)

AGE   [Enter Age]   * Accepts value from 18 to 60

Equivalence Class Partitioning		
Invalid	Valid	Invalid
<=17	18-60	>=61

- A valid class will be anything between 18 and 60.
- We have thus reduced the test cases to only 3 test cases based on the formed classes thereby covering all the possibilities. So, testing with any one value from each set of the class is sufficient to test the above scenario.

Boundary Value Analysis

- Boundary value analysis is one of the widely used case design techniques for black box testing.
- It is used to test boundary values because the input values near the boundary have higher chances of error.
- Boundary values are those that contain the upper and lower limit of a variable.
- Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.
- The basic assumption of boundary value analysis is, the test cases that are created using boundary values are most likely to cause an error.

Boundary Value Analysis - Example 1

Imagine, there is an Age function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29.

The invalid partition consists of the numbers which are less than 18 such as 12, 14, and more than 30 such as 31, 32, 34, 36 and 40.

Tester develops test cases for both valid and invalid partitions to capture the behavior of the system on different input conditions. Test cases are developed for each and every value of the range.

| Age | Between 18 to 30 |

```
12  14   15   16   17  18  20 22 24 25 26 28  30  31 32 34 36  38 40
-------------------------------|-------------------------------|---------------------------
Invalid Partition              Valid Partition                 Invalid Partition
```

Invalid test cases	Valid test cases	Invalid test cases
11, 13, 14, 15, 16, 17	18, 19, 24, 27, 28, 30	31, 32, 36, 37, 38, 39

Decision Tables

This technique is related to the correct combination of inputs and determines the result of various combinations of input.

To design the test cases by decision table technique, we need to consider conditions as input and actions as output.

Example 1

Most of us use an email account, and when you want to use an email account, for this you need to enter the email and its associated password.

If both email and password are correctly matched, the user will be directed to the email account's homepage; otherwise, it will come back to the login page with an error message specified with "Incorrect Email" or "Incorrect Password."

Decision Tables - Example 1 contd.

Let's see how a decision table is created for the login function in which we can log in by using email and password.

Both the email and the password are the conditions, and the expected result is action.

Email (condition1)	T	T	F	F
Password (condition2)	T	F	T	F
Expected Result (Action)	Account Page	Incorrect password	Incorrect email	Incorrect email

Structural Testing - Example 1 contd.

There are four conditions or test cases to test the login function. In the first condition if both email and password are correct, then the user should be directed to account's Homepage.

In the second condition if the email is correct, but the password is incorrect then the function should display Incorrect Password.

In the third condition if the email is incorrect, but the password is correct, then it should display Incorrect Email.

Now, in fourth and last condition both email and password are incorrect then the function should display Incorrect Email.

Tester uses 2^n formula where n denotes the number of inputs; in the example there is the number of inputs is 2 (one is true and second is false).

Number of possible conditions = 2^ Number of Values of the second condition
Number of possible conditions =2^2 = 4

Structural Testing - Example 2

Take an example of XYZ bank that provides interest rate for the Male senior citizen as 10% and for the rest of the people 9%.

Condition C1 has two values as true and false,

Condition C2 also has two values as true and false.

The number of total possible combinations would then be four.

This way we can derive test cases using a decision table.

Decision Table / Cause-Effect				
Decision Table	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Conditions				
C1 - Male	F	F	T	T
C2 - Senior Citizen	F	T	F	T
Actions				
A1 - Interest Rate 10%				X
A2 - Interest Rate 9%	X	X	X	

State Transition Testing

This technique is based on the idea that a system can be in one of a number of states, and that when an event occurs, the system transitions from one state to another. The events that can cause a state transition are known as **triggers** and the states that can be reached from a given state are known as **targets**.

- To carry out state transition testing, a tester first needs to identify all of the possible states that a system can be in, and all of the possible triggers that can cause a state transition.
- They then need to create a test case for each state transition that they want to test.
- When carrying out the test, the tester will start in the initial state, and then trigger the event that they want to test.
- They will then observe the system to see if it transitions to the expected state.

Practice Question

The basic membership fee to a club only for adults above 18 years is $200, however, this fee can increase or decrease depending on three factors: their age, income, and their marital status. Customers that are above the age of 65 have their fee decreased by $50. Customers that have income above $10000 per month have their fee increased by $50. Customers that are married get a fee increase of 10%.

Derive test cases and test data for the following Black Box Software testing methods:

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table

Grey Box Testing

- Grey box testing is a software testing method to test the software application with partial knowledge of the internal working structure.

- It is a combination of black box and white box testing because it involves access to internal coding to design test cases as white box testing and testing practices are done at functionality level as black box testing.

BlackBox Testing   +   WhiteBox Testing   =   GreyBox Testing
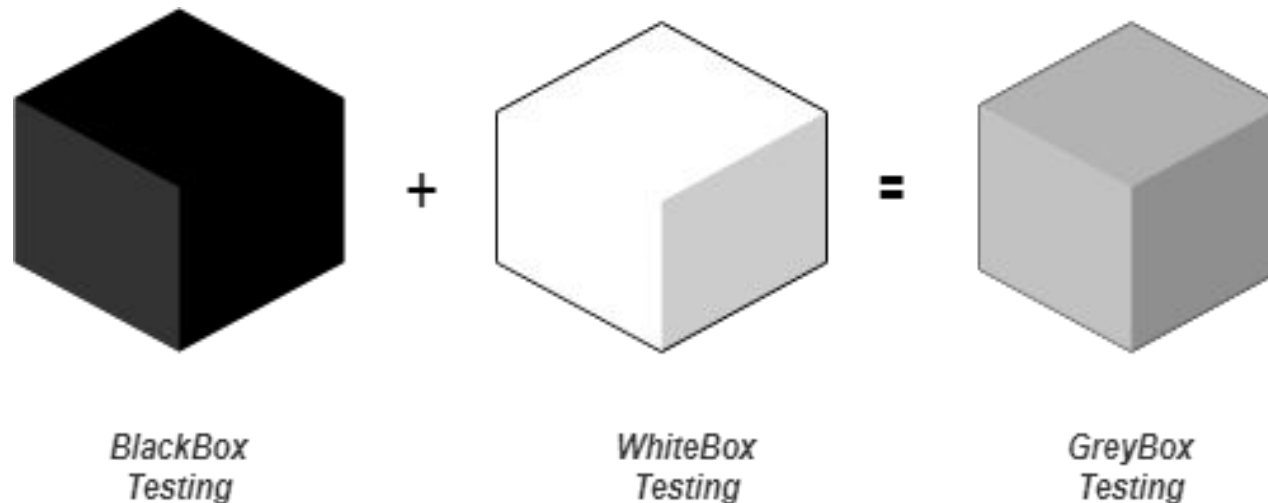
Grey Box Testing Strategy

- Grey box testing does not make necessary that the tester must design test cases from source code.
- To perform this testing test cases can be designed on the base of, knowledge of architectures, algorithm, internal states or other high -level descriptions of the program behavior.
- It uses all the straightforward techniques of black box testing for function testing.

Example of Gray Box Testing : While testing websites feature like links or orphan links, if tester encounters any problem with these links, then he can make the changes straightaway in HTML code and can check in real time.

Generic Steps to Perform Grey Box Testing

1. Select and identify inputs from BlackBox and WhiteBox testing inputs.
2. Identify expected outputs from these selected inputs.
3. Identify all the major paths to traverse through during the testing period.
4. Identify sub-functions which are the part of main functions to perform deep level testing.
5. Identify inputs for subfunctions.
6. Identify expected outputs for subfunctions.
7. Executing a test case for Subfunctions.
8. Verification of the correctness of result.

Advantages of Grey Box Testing

- It provides combined benefits of both black box testing and white box testing both
- It combines the input of developers as well as testers and improves overall product quality
- It reduces the overhead of long process of testing functional and non-functional types
- It gives enough free time for a developer to fix defects
- Testing is done from the user point of view rather than a designer point of view

Techniques used for Grey Box Testing

1. **Matrix Testing:** This testing technique involves defining all the variables that exist in their programs.

2. **Regression Testing:** To check whether the change in the previous version has regressed other aspects of the program in the new version. It will be done by testing strategies like retest all, retest risky use cases, retest within a firewall.

3. **OAT or Orthogonal Array Testing:** It provides maximum code coverage with minimum test cases.

4. **Pattern Testing:** This testing is performed on the historical data of the previous system defects. Unlike black box testing, gray box testing digs within the code and determines why the failure happened

THANK YOU

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

Software Testing

Unit 3
Test Management

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

List of Contents

- **Testing Phase as a project**
- **Testing Strategy**
- **Facets of Test Planning**
- **Test Management**
- **Test Process**
- **Test Reporting**
- **Best Practices**

Department of Computer Science and Engineering

Testing Phase as a project

Defining a Project

- A project, as per the Project Management Institute (PMI-2004), is *"a temporary endeavor to create a unique product or service."*
- It is aimed at creating a unique product or service, setting it apart from others in some distinctive way.

- Every project has a clear and predefined starting point and a specific end date.
- The end result of a project, be it a product or a service, possesses distinctive features that set it apart from others
- Notably, **even testing can qualify as a project on its own**. This implies that activities like testing need to be meticulously planned, executed, tracked, and reported, just like any other project.

Testing Strategy

1. High-Level Approach and Philosophy

Define the overarching principles that guide your testing approach.

1. Balanced Focus on Product and Process Aspects

Highlight the importance of both product functionality and testing process efficiency.

1. Break down the strategy into specific components:

   a. Automation: Explain the extent of test automation and its benefits.
   b. Test Data: Describe how test data is managed and secured.
   c. Team / Third Party: Clarify roles and responsibilities for in-house and external teams.
   d. Test Record Management: Emphasize the importance of organized documentation.
   e. Test Defect Management: Describe the defect tracking and resolution process.
   f. Controls for Testing: Discuss metrics for coverage, effort, and reporting.
   g. Interface with Other Teams / Customer: Outline collaboration methods.
   h. Test Metrics: Define key performance indicators (KPIs) for measuring test success.
   i. Alignment with Product Release / Lifecycle Phases: Explain how testing aligns with development milestones.

Facets of Test Planning
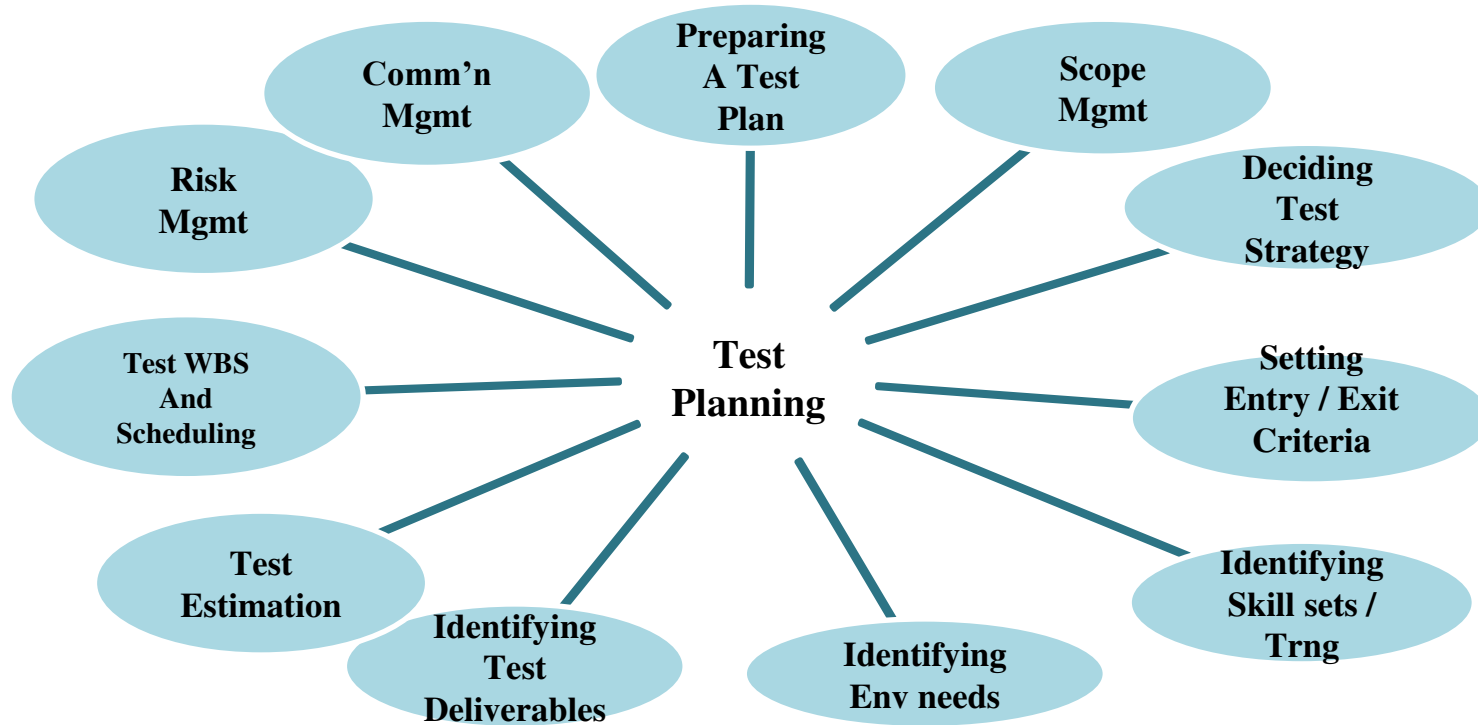
Scope Management

- Understanding what constitutes a release of a product
- Breaking down the release into features
- Prioritizing the features for testing. This includes
  - Features that are new and critical for the product release
  - Features whose failures will be catastrophic
  - Features that are re-extensions of features that have a problem-prone track record
  - Consideration on environmental and other combinatorial factors
- Deciding which features will be tested and which will not be
- Gathering details to prepare for estimation

Deciding the test approach / strategy for the chosen features

- What type of testing would you use for testing the functionality of each feature?
- What are the configurations or scenarios for testing the features?
- What integration testing would you do to ensure these features work together?
- What localization validations would be needed?
- What "non-functional" tests would you need to do?

Setting Up Criteria for Testing

Entry criteria

Defining when to start testing for each type of testing.

Exit criteria

Considering completeness, risk of release – how much to test and when to stop.

Suspension criteria : Defining things such as -

Show stopper bugs, Crossing a threshold number of bugs

Developers producing a new version, making the old one redundant

Resumption criteria

The above mentioned hurdles being cleared

Identifying Staff Skill Sets and Training

Organizational Structure Establishment

Defining the framework of your organization for efficient operations.

Roles are Defined to:

- Have clear accountability for a given task, so that each person knows what they have to do
- Clearly list the responsibilities for various people involved so that everyone knows how his or her work fits into the entire project
- Complement each other, ensuring no one steps on others' toes
- Supplement each other, so that no task is left unassigned
- Establish management and reporting responsibilities
- Match job requirements with people's skills and aspirations as best as possible
- Identify and provide necessary training (*Why?*) (Ans : Identify skill and knowledge gaps, Provide essential tools for excellence, Improve employee performance and satisfaction)

Identifying Environment Needs

Basically refers to identifying resource requirements such as:

- Hardware requirements, including RAM, processor, and disk space.
- Test tool prerequisites.
- Supporting tools like compilers, test data generators, and configuration management tools.
- Different software configurations (e.g., operating systems) needed.
- Special considerations for resource-intensive tests (e.g., load and performance tests).
- Ensure sufficient software licenses.
- Infrastructure and resources, including office space and support functions.
- Planning based on limitations and constraints.

Identifying Test Deliverables

- Test Plan: Including master test plan and project-specific test plans.
- Test Case Design Specs: Detailed specifications for test case creation.
- Test Cases: Inclusive of any specified automation.
- Post-Testing Use: Understanding the purpose of test cases after testing.
- Test Data/Test Bed: Data and environment setup for testing.
- Test Logs: Records generated during test execution.
- Test Summary Reports: Summarizing the overall test results.

Test Estimation

There are different phases in Test Estimation. We are supposed to estimate:

1. Size
2. Effort
3. Resources
4. Elapsed Time required

Test Estimation : Size

- Refers to actual amount of testing to be done

- Depends on
    - Size of product under test
    - Extent of automation required
    - Number of platforms and interoperability requirements

- Expressed as
- Number of test cases
- Number of test scenarios
- Number of configurations to be tested

Test Estimation : Effort

- Effort estimation impacts project cost directly.
- Factors influencing effort estimate:
  - Productivity metrics (e.g., test case creation, automation, execution, analysis per day).
  - Potential for reusing existing resources.
  - Process robustness and efficiency.

Test WBS and Scheduling

Activity Breakdown and Scheduling

- Dependency & Time Overview:
    - Highlight key dependencies.
    - Emphasize elapsed time considerations.

- Comprehensive Activity Plan:
    - Provide a detailed breakdown of activities.
    - Ensure synchronization for smooth workflow.

Test WBS and Scheduling (contd.)

Scheduling of Activities

- Identifying external and internal dependencies among the activities
- Sequencing the activities based on the expected duration as well as on the dependencies
- Identifying the time required for each of the WBS activities, taking into account the above two factors
- Monitoring the progress in terms of time and effort
- Rebalancing the schedules and resources as necessary

Dependencies

External Dependencies

- Product developer availability for support.
- Access to required documentation.
- Hiring resources and considerations.
- Availability of training resources.
- Procurement of necessary hardware/software.
- Availability of translated message files for testing.

Internal Dependencies

- Finalize test specifications.
- Code/script the tests.
- Execute the tests.
- Resolve conflicts if environment sharing is necessary.
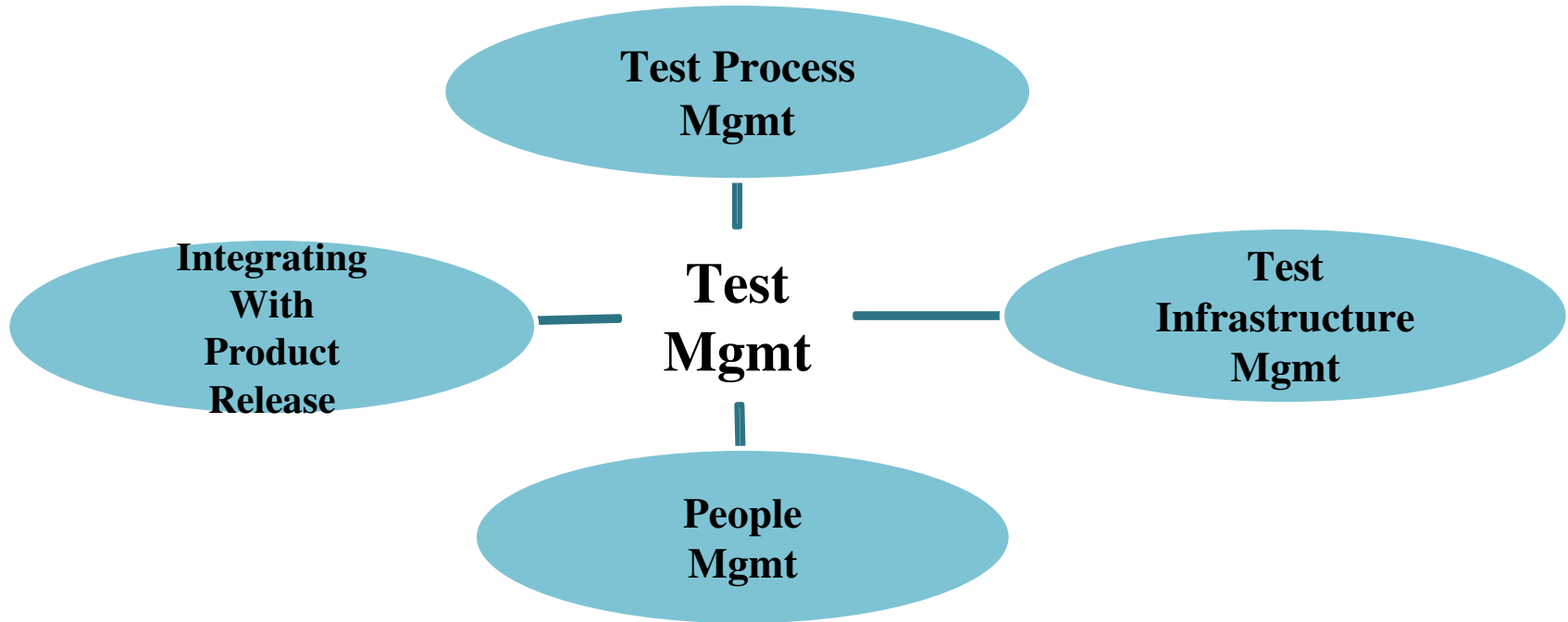
Risk Management

Common Risks and Mitigation in Testing Projects

- Unclear Requirements:
    - Engage the testing team from the start for clarity.
- Schedule Dependency and Downstream Positioning:
    - Implement backup tasks, multiplexing, and parallel automation.
- Insufficient Testing Time and Excessive Caution:
    - Utilize the V Model and establish clear entry/exit criteria.
- Critical Defects:
    - Identify and address "show stopper" defects promptly.
- Non Availability of Skilled Testers:
    - Showcase career paths to attract and retain skilled testers.
- Challenges in obtaining the required test automation tool.

Test Management

Test Process Management

- Test Artifacts Naming and Storage Guidelines
- Ensuring alignment between product features and corresponding test suites.
- Documentation and Test Coding Standards
- Separate Test Reporting Standards
- Test Configuration Management

Test Infrastructure Management

Essential Components of Test Infrastructure Management

- Test case repository (TCDB)
- Artefact configuration database (SCM)
- Defect repository
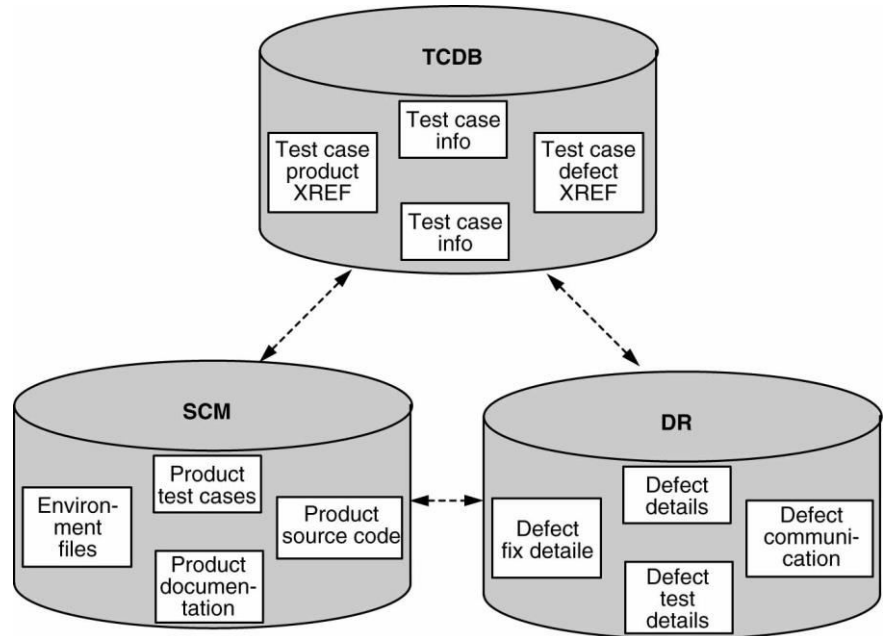
Integrating with Product Release

- Synchronize development and testing timelines for various testing phases (e.g., integration, system testing).
- Establish Service Level Agreements (SLAs) to define testing completion timeframes.
- Maintain uniform definitions of defect priorities and severities.
- Establish communication channels with documentation team for defect-related updates and workarounds.

Test Process

1. Putting together and baselining a test plan

2. Develop comprehensive test specifications, covering:

- Purpose of test
- Items being tested, with their version numbers
- Environmental setup needed
- Input data required
- Steps to be followed
- Expected results
- Any relationship to other tests

3. Update of RTM

4. Identifying possible automation candidates

5. Developing and "baselining" test cases

Test Process (contd.)

6. Executing tests and keeping RTM and DR current

7. Collecting and analyzing metrics (later class)

8. Preparation of test summary report

9. Recommending product release criteria

Test Reporting

- **Test Incidence Report**
  - Update to DR (Defect Repository)
  - Includes Test ID, Product & Component Info, Defect Description, and Fix Info
- **Test Cycle Report**
  - Provides a summary of cycle activities
  - Highlights uncovered defects, categorized by severity and impact
  - Tracks progress from the previous cycle
  - Lists outstanding defects for the current cycle
  - Notes any variations in effort or schedule
- **Test Summary Report**
  - Phase-wise summary and final test summary included

Test Reporting - Test Summary Report

- **Test Summary Report Overview** : Provides an overview of test cycle or phase activities.
- **Activities Variance** : Highlights variances from planned activities, including:
  - Tests that couldn't be executed (with explanations).
  - Test modifications compared to original specifications (update TCDB).
  - Additional tests conducted (not in the original plan).
  - Differences in effort and time compared to the plan.
  - Any other deviations from the plan.
- **Summary of Test Results** : Covers
  - Failed tests with root-cause descriptions (if available).
  - Severity of defects uncovered by the tests.
- **Comprehensive Assessment** : Evaluates the product's "fit-for-release."
- **Recommendation for Release** : Provides a release recommendation.

Test Reporting - Recommendation for Release

- Utilize Dijkstra's Doctrine.
- Testing Team's Role: Inform management about defects and potential risks.
- Quality Profile: Assess the overall quality of the product.
- Final Decision: Management decides whether to release as-is or allocate additional resources for defect resolution.

Best Practices

- **Usage of Process Models**: Adopt CMMI or TMM for efficient processes.
- **Foster Collaboration:** Cultivate strong cooperation between Development and Testing teams.
- **Leverage Technology**: Utilize an integrated SCM-DR-TCDB infrastructure and prioritize test automation.

THANK YOU

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

Software Testing

Unit 3
Defect Management

Prof Raghu B. A. Rao

Department of Computer Science and Engineering

List of Contents

- **Defect Management Process**
- **Defect in Software Testing**
- **The Objective of Defect Management Process (DMP)**
- **Various Stages of Defect Management Process**

Department of Computer Science and Engineering

Defect Management Process

A systematic approach to identify and resolve bugs.

Key Stages:

1. Discovery of Defects
2. Categorization of Defects
3. Defect Resolution by Developers
4. Verification by Testers
5. Defect Closure
6. Generation of Defect Reports (End of Project)

Defect in Software Testing

- A defect in software testing refers to a problem or error within the code.
- Essentially, it is when the application does not perform as intended or specified.
- It signifies a deviation from the expected behavior of the software.
- The defect is essentially the gap between what the software should do and what it actually does.
- Test engineers are responsible for identifying defects during the testing phase.
- Once identified, defects are typically fixed by developers as part of the software development life cycle.
- During testing, test engineers observe discrepancies between the expected and actual outcomes, which are labeled as defects.

The Objective of Defect Management Process (DMP)

- **Early Detection of Defects**: One key aim of the Defect Management Process (DMP) is to identify and address defects in software development at an early stage.
- **Process Enhancement**: Implementing the DMP contributes to refining the software development process and its implementation.
- **Mitigating Defect Impact**: The DMP is geared towards minimizing the negative effects and impact of defects on the software.
- **Defect Prevention:** DMP plays a role in preventing the occurrence of defects in software.
- **Defect Resolution**: The primary objective of the Defect Management Process is to efficiently resolve and fix identified defects.

Various Stages of Defect Management Process

The defect management process includes several stages, which are as follows:

1. Defect Prevention
2. Deliverable Baseline
3. Defect Discovery
4. Defect Resolution
5. Process Improvement
6. Management Reporting

Defect Management Process

Defect Prevention • Deliverable Baseline • Defect Discovery • Defect Resolution • Process Improvement

Management Reporting
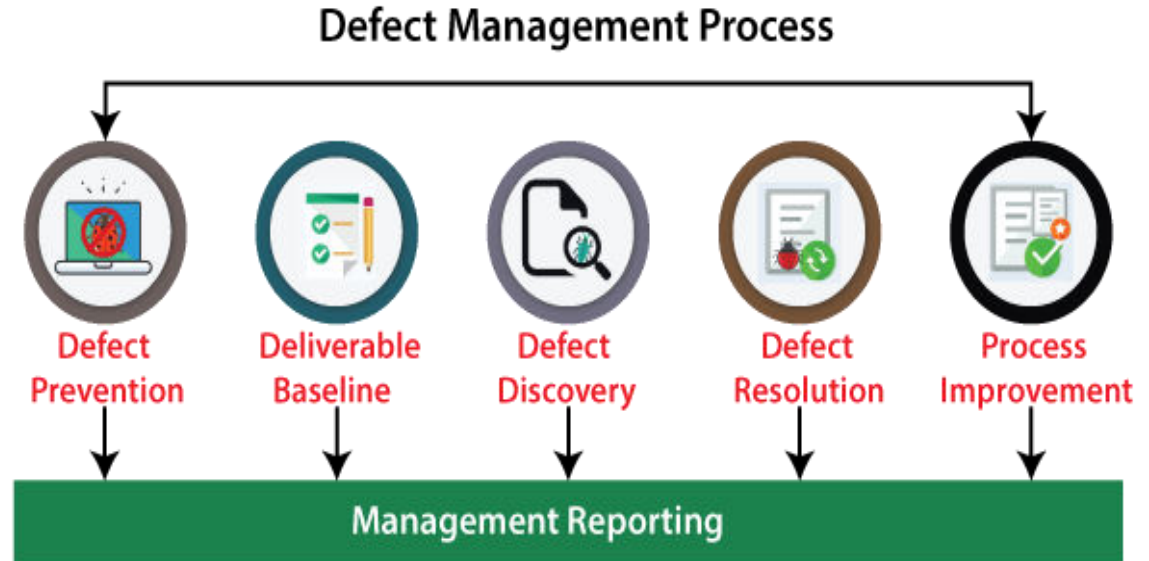
1. Defect Prevention

- The first stage of the defect management process is defect prevention. In this stage, the execution of procedures, methodology, and standard approaches decreases the risk of defects.
- Defect removal at the initial phase is the best approach in order to reduction its impact.
- The defect prevention stage includes the following significant steps:
    - Estimate Predictable Impact
    - Minimize expected impact
   - Identify Critical Risk

2. Deliverable Baseline

- The second stage of the defect management process is the Deliverable baseline. Here, the deliverable defines the system, documents, or product.
- We can say that the deliverable is a baseline as soon as a deliverable reaches its pre-defined milestone.

3. Defect Discovery

- The next stage of the defect management process is defect discovery. At the early stage of the defect management process, defect discovery is very significant. And later, it might cause greater damage.
- If developers have approved or documented the defect as a valid one, then only a defect is considered as **discovered**.
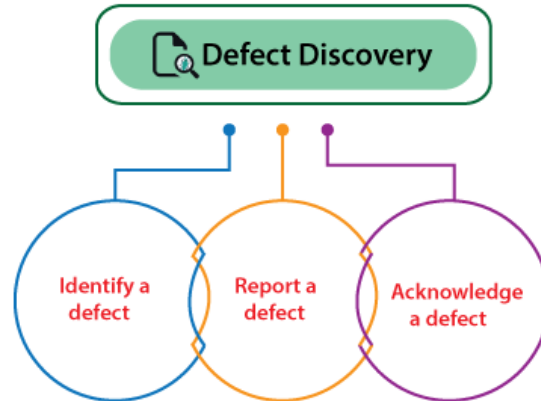
4. Defect Resolution

- The **Defect Resolution** is a step-by-step procedure of fixing the defects, or we can say that this process is beneficial in order to specified and track the defects.

- This process begins with handing over the defects to the development team. The developers need to proceed with the resolution of the defect and fixed them based on the **priority.**

- We need to follow the below steps in order to accomplish the defect resolution stage.
  - Prioritize the risk
  - Fix the defect
  - Report the Resolution

5. Process Improvement

- The **process improvement** phase, we will look into the lower priority defects because these defects are also essential as well as impact the system.

- All the acknowledged defects are equal to a critical defect from the process improvement phase perspective and need to be fixed.

- The people involved in this particular stage need to recall and check from where the defect was initiated.

- Depending on that, we can make modifications in the **validation process, baselining document, review process** that may find the flaws early in the process, and make the process less costly.

6. Management Reporting

- Management reporting is the last stage of the defect management process. It is a significant and essential part of the defect management process.
- The management reporting is required to make sure that the generated reports have an objective and increase the defect management process.
- In simple words, we can say that the evaluation and reporting of defect information support organization and risk management, process improvement, and project management.

THANK YOU

Prof Raghu B. A. Rao

Department of Computer Science and Engineering