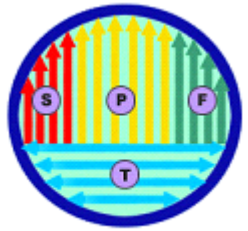


You Are What You Know

WIPRO

Applying Thought

Testing Concepts

Testing Concepts Ver2.0

Course Objectives

After completing this course, you should be able to:

- Ø Explain the need for Testing
- Ø Identify Testing Process
- Ø Identify the types of tests to be conducted on applications
- Ø Describe various types of Test and their objectives

Contents of this Course

Chapter-1

SDLC

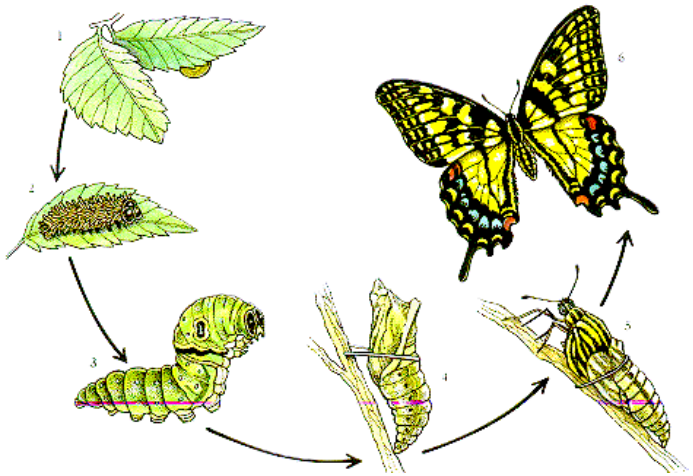
Software/Application??

1. Group of programs designed for end user using operating system and system utilities.

2. A self contained program that performs well- defined set of tasks under user control.

3. Programs, procedures, rules, and any associated documentation pertaining to the operation of a system.

Software Development life cycle



Phases or stages of a project from inception through completion and delivery of the final product…

… and maintenance too!

Software Development Stages

1. Requirement
2. Analysis
3. Design
4. Coding
5. Testing
6. Deployment
7. Maintenance

Requirement Phase

1. Business Requirement
    1. Represent high-level objectives of the organization or customer requesting the system or product

    2. Objectives captured in a project's vision and scope document

Ex.

The Flight Reservation System's login operations must provide basic security to access the system.

Requirement **Cont...**

2. User Requirements

1. Describe tasks the user must be able to accomplish with the product.

Ex.

1. Enter user id in the Agent Name field
2. Enter the password in the Password field
3. Click OK

Requirement **Cont...**

3. Functional Requirements
   1. Define the application's software functionality to enable users to accomplish their tasks, thereby satisfying the business requirements

   2. Documented in a *Software Requirements Specification* (SRS).

      Ex.

      1. Allow access to authorized user ids and passwords
      2. Deny access to invalid user ids and passwords

Analysis Phase

The process of identifying requirements, current problems, constraints, Opportunities for improvement , timelines and Resources costs .

Analysis Phase

1. …used for resolving any problems that need to be addressed.

2. determine what attributes and behavior each object should have, and how the objects should interact or interrelate with each other

3. “Figuring out what your program should do in the best way possible”

4. Feasibility study (social, economical etc)

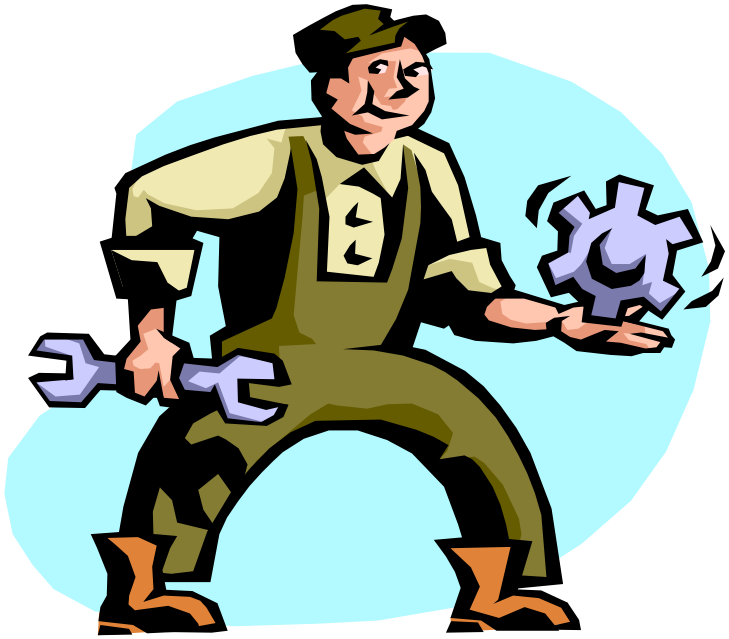
5. investigate the need for possible software automation in the given system.

Requirement Analysis

Difficulties in conducting requirement analysis

1. Analyst not prepared
2. Customer has no time/interest
3. Incorrect customer personnel involved
4. Insufficient time allotted in project schedule

Design Phase



The business of finding a way to meet
the functional requirements within the specified constraints using the available technology

Design Phase

1. Determine the set of instructions that will lead to a solution that meets the customer's requirements.

2. This set of instructions is called an algorithm.

3. Algorithms are usually represented pictorially (by flowcharts or **UML** state diagrams)

4. Architecture, including hardware and software, communication, software design (UML is produced here) are all part of the deliverables of a design phase

Design Phase

1. HLDS (High Level Design Specifications) From User Point of view
    1. List of modules and a brief description of each
    2. Brief functionality of each module
    3. Interface relationship among modules
    4. Dependencies between modules
    5. Database tables identified with key elements
    6. Overall architecture diagrams along

Design Phase

2. LLDS (Low Level Design Specifications) / Detailed Design

    1. Detailed functional logic of the module, in pseudo code
    2. Database tables, with all elements, including their type and size
    3. All interface details
    4. All dependency issues
    5. Error MSG listing
    6. Complete input and output format of a module

Coding Phase

1. The specifications from the detailed design phase are converted into programs.

2. Code is produced from the deliverables of the design phase during implementation, and this is the longest phase of the software development life cycle.

3. Design must be translated into a machine-readable form taking input as SRS.
    1. Done by Team of developers.
    2. Reviews after every 500 lines of code
        1. Code Inspection
        2. Code Walkthrough

Testing Phase

1. Finding the hidden defects in developed Product.

2. During testing, the implementation is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase.

3. Unit, Integration, system and acceptance tests are done during this phase. Unit tests act on a specific component of the system, while system tests act on the system as a whole.

Deployment and Close Phase

1. Deployment

    1. Deploying the product at client side
    2. Installation and setup of software application at the customer site

Maintenance Phase

1. Maintaining the software application
    1. Fixing bugs
    2. Integrate new features requested by customers
    3. Provide support
2. On-going process

Chapter-2

Fundamentals of Testing

Fundamentals of Testing

Topics Covered in this lesson are:

1. Fundamentals of Software Testing
2. Fundamentals of Software Quality
3. People Challenges in Software Testing

Part 1: Fundamentals of Software Testing

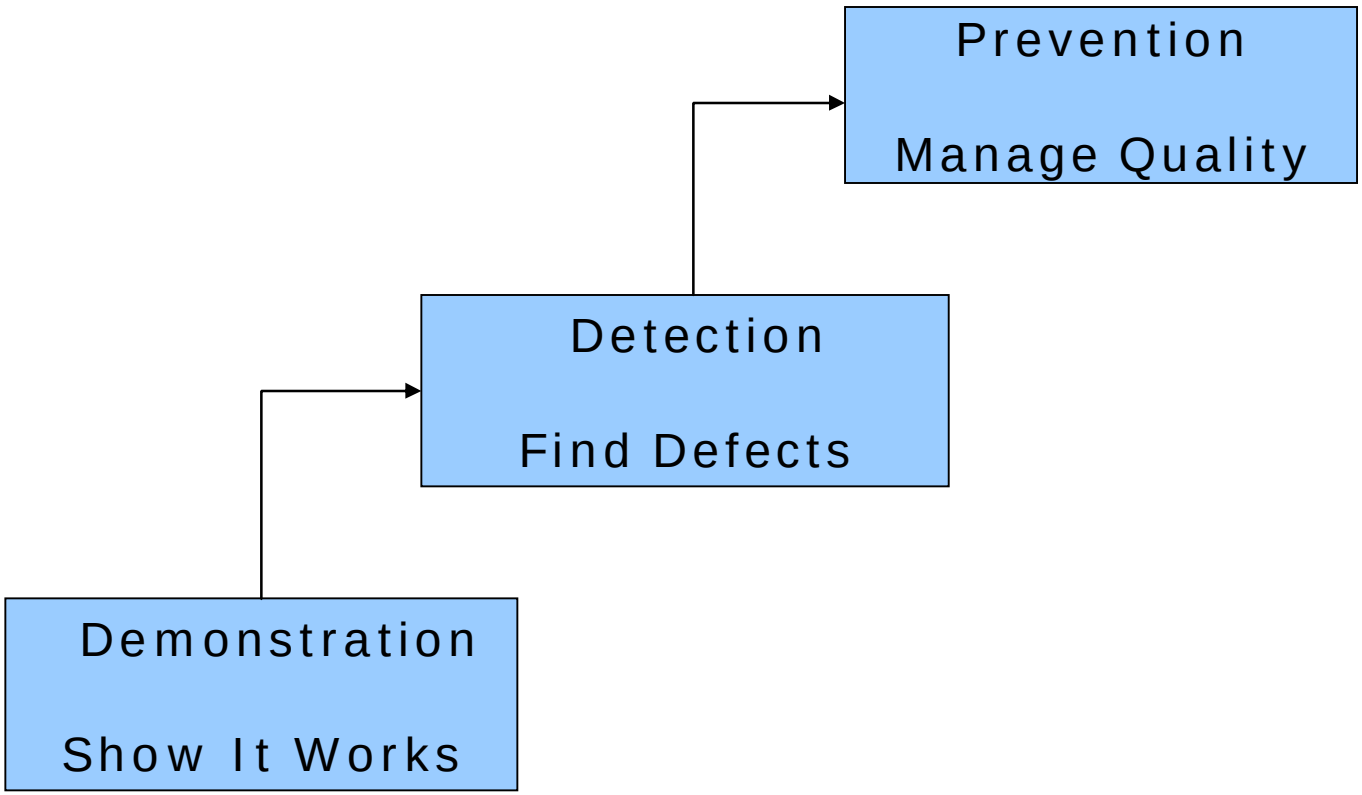
1. Testing Objectives
2. Testing Definitions
3. Principles And Best Practices
4. Challenges and Paradox of Testing
5. Testing Process
6. Testing Methods
    1. Black Box
    2. White Box

Software Testing

1. It is not sufficient to demonstrate that the software is doing what it is supposed to do.

2. It is more important to demonstrate that the software is not doing what it is not supposed to do.

Testing Objectives



Definitions of Testing

1. Software testing is a **process** of analyzing or operating software for the purpose of finding **bugs**.

2. Testing is the process of identifying **defects**, where a defect is any variance between **actual** and **expected** results.

3. Software testing is the process used to help identify the **correctness**, **completeness**, **security**, and **quality** of developed computer software.

Definitions of Testing

4. Testing is generally described as a group of procedures carried out to **evaluate** some aspect of a piece of software.

5. Testing can be described as a process used for revealing defects in software, and for establishing that the software has attained a specified degree of **quality** with respect to selected attributes.

Benefits of testing?

1. The prime benefit of testing is that it result in improved quality, bugs get fixed.

2. We take a destructive attitude towards the program when we test, but in larger extent our work is constructive.

3. We are beating up the program in the service of making it stronger.

4. Demonstrate that software functions appear to be working according to specification

5. The performance requirement appears to have been met.

Principles of Testing

1. Testing Is Risk-based
2. Analysis, Planning, And Design Are Important
3. Time And Resources Are Important
4. Timing Of Test Preparation Matters A Lot
5. Measuring And Tracking Coverage Is Essential

Best Practices of Testing

1. Establish an early, integrated master testing and evaluation plan
2. Make preventive testing part of all specification work
3. Inventory test objectives and design for testability
4. Test early, Test often
5. Measure test costs, coverage, results, and effectiveness

Issues in Testing

1. Testing considered late in the project
2. Requirements not testable
3. Integrate after all components have been developed
4. One step forward, two steps backward
5. Test progress hard to measure

Testing Paradoxes

1. The best way to test and verify requirements is to figure out how to test and verify requirements.

2. The best way to build confidence is to try to destroy it.

3. Testing is looking for what you don't want to find. A successful test is one that fails, and a failure is one that succeeds.

Testing Paradoxes

4. Managers who fail to spend appropriately on testing end up with failed projects because they can't meet their deadlines if they do spend appropriately on testing.

5. Quality is free if you pay for it.

Why does software have bugs?

1. Miscommunication or no communication
2. Software complexity
3. Programming errors
4. Changing requirements
5. Time pressures
6. Poorly documented code
7. software development tools

Essentials of Software Testing

1. TESTING IS A DESTRUCTIVE PROCESS : A CREATIVE DESTRUCTION

2. TESTING NEEDS A SADISTIC APPROACH

3. Test that detects a defect is valuable investment:
   it has helped in improving the product.

4. If sole purpose of a test is to validate
   specifications were implemented, then:
   1. Testing is an unnecessary and unproductive activity

Essentials of Software Testing

1. It is risky to develop software and not test it.

2. High pressures to deliver software as quickly as possible: test process must provide maximum value in shortest time.

3. Testing is no longer:
   1. After-programming evaluation
   2. Adjunct to the SDLC

Essentials of Software Testing

4. It is:
    1. Key integral part of EACH phase of SDLC

5. Highest payback comes from detecting defects early or preventing defects:
    1. Avoid incorrect design and coding so as to Avoid correcting defects later

Essentials of Software Testing

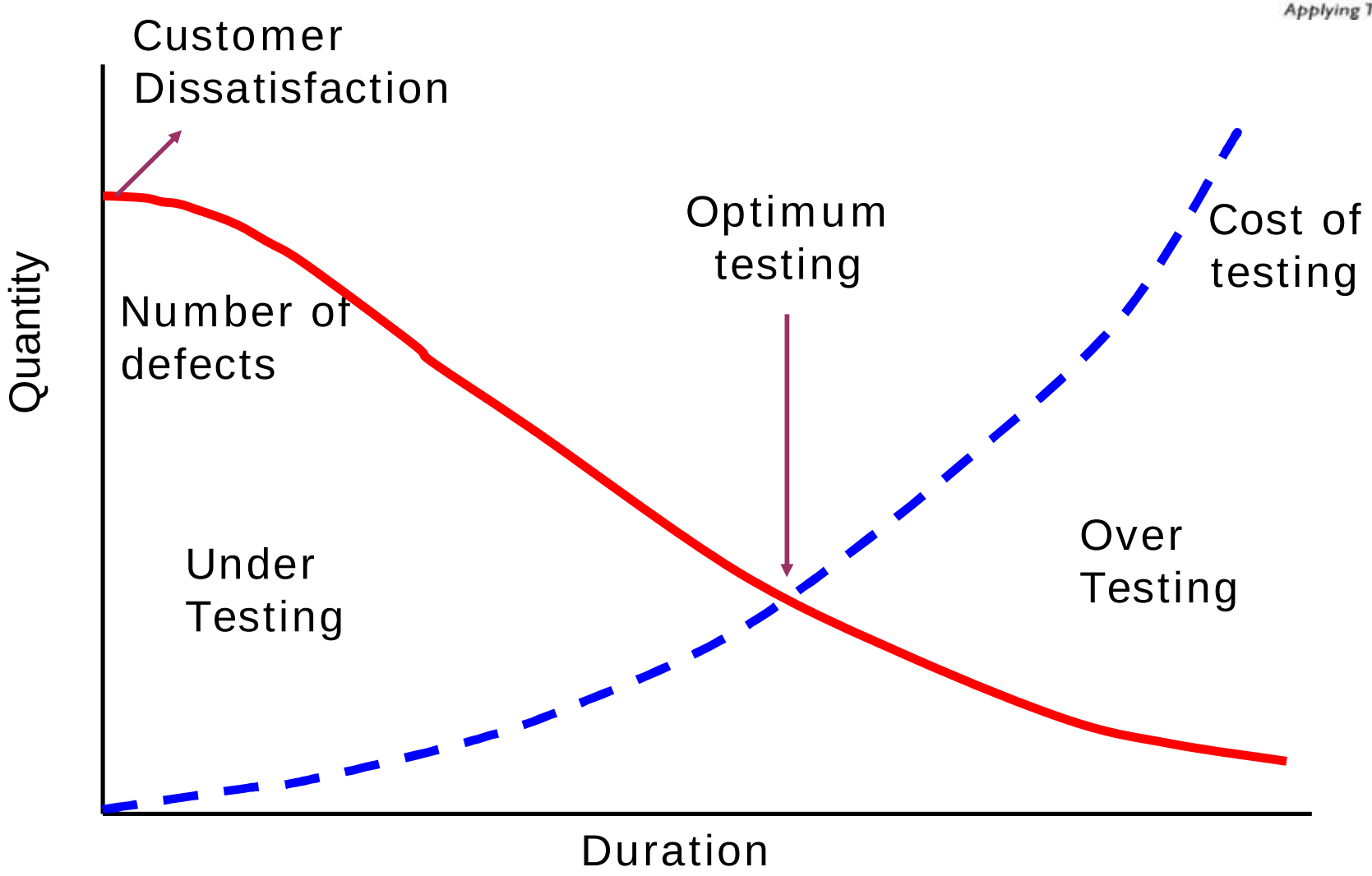
1. Aim is to reduce the time ‘wasted’ in defect removal: concentrate on defect prevention.

2. Misconceptions:
   1. Anyone can test software: no particular skill is required!

   2. Testers can test for quality at the end of a project!

   3. Defects found mean blaming the developers.

Economics of Testing



Testing Process – How ?

1. Create Test Plan
2. Design test cases
3. Write test cases
4. Review test cases
5. Execute the tests
6. Examine test results
7. Perform postmortem reviews

Participants in Testing – Who ?

1. Customers
2. Users
3. Developers

   (Includes individuals / groups which gather requirements, design, build, change and maintain software)
4. Testers
5. Senior Management
6. Auditors

Who Performs the Tests?

1. Unit testing and Basic level of integration testing are usually done by development teams.

2. Module testing, system testing, adhoc testing etc. is done by the testing team.

3. Customers/Users perform user acceptance tests (UAT)

Start Testing - When ?

1. Timing: As soon as we have a Software Requirements (baseline).

2. Objective –
    1. To trap requirements-related defects as early as they can be identified.

3. Approach –
    1. Test Requirements

Stop Testing – When?

Following factors are considered,

1. Deadlines, e.g. release deadlines, testing deadlines;
2. Test cases completed with certain percentage passed;
3. Coverage of code, functionality, or requirements reaches a specified point;
4. Bug rate falls below a certain level; or
5. Beta or alpha testing period ends.

What is defect?

1. The software doesn’t do something that the product specification says it should do.

2. The software does something that the product specifications says it shouldn’t do.

3. The software does something that the product specification doesn’t mention.

4. The software doesn’t do something that the product specification doesn’t mention but should do.

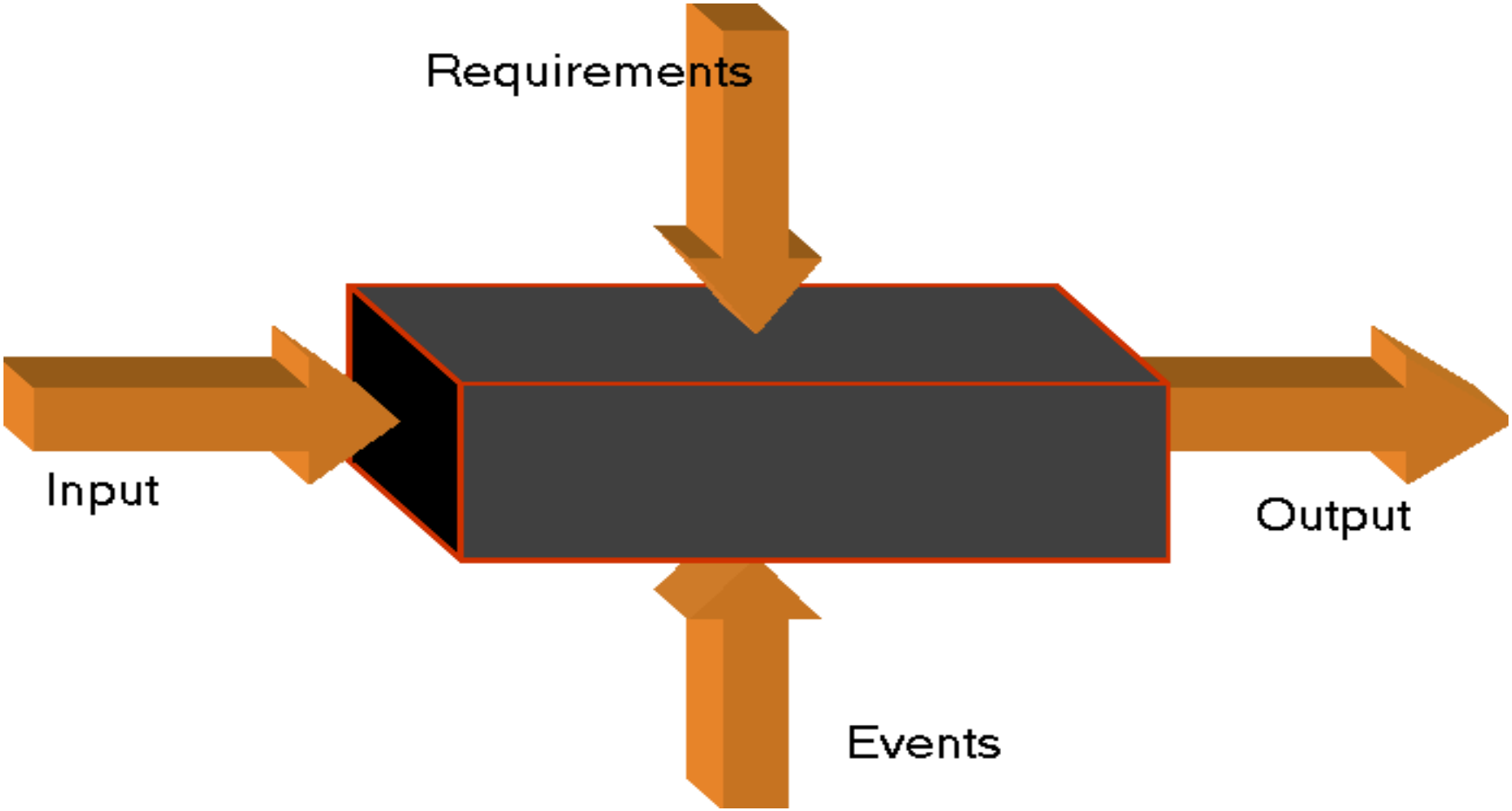
5. Difficult to understand, hard to use, slow etc.

Test Methods

1. Black box testing (also called functional testing):
    1. Test the specs (Do not look inside the code)

2. White box testing (also called structural / glass box testing):
    1. Test the code

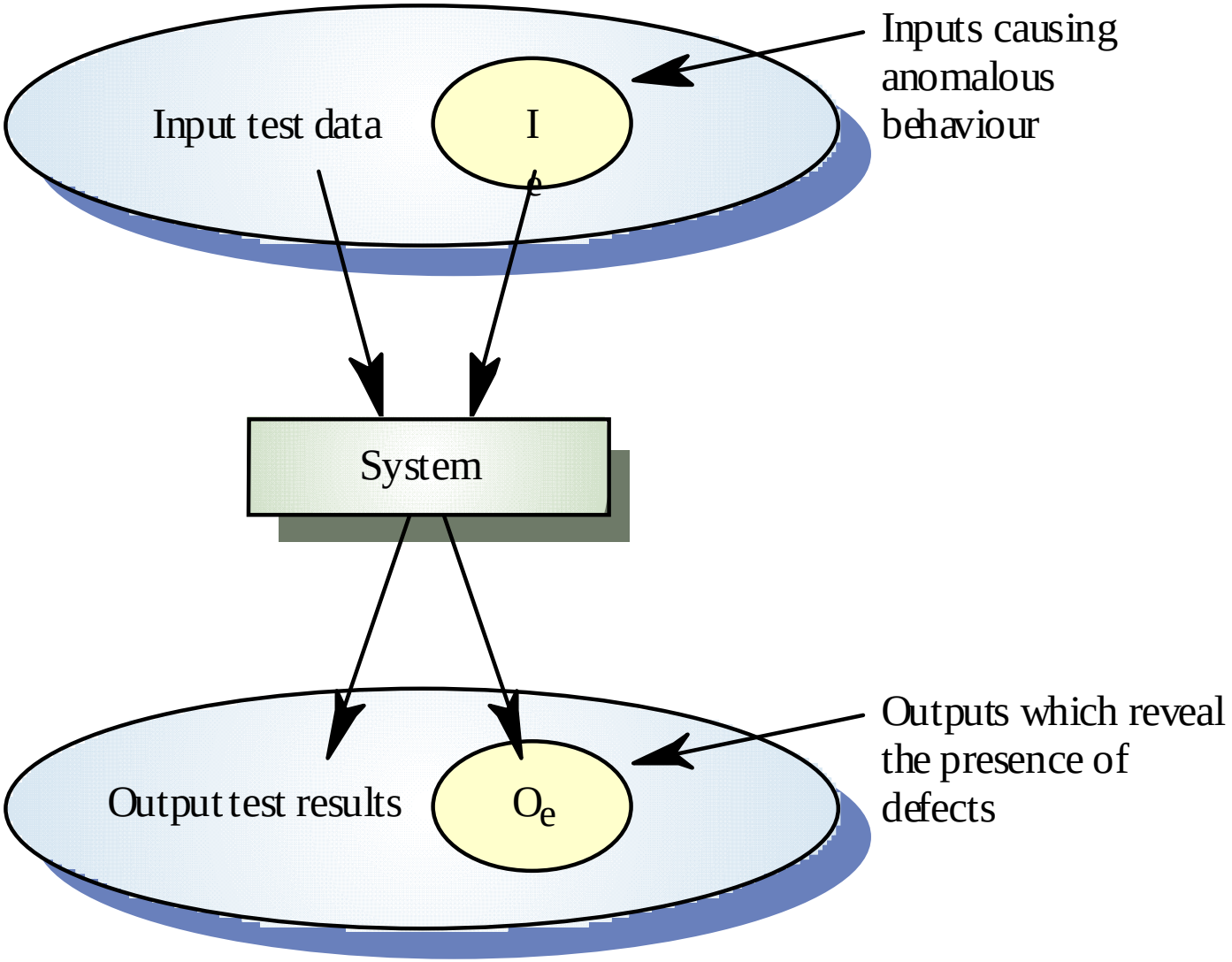
Black Box Testing



Black Box Testing

1. Approach to testing where the program is considered as a “black-box”.
2. Testing based exclusively on analysis of requirements (specification, user documentation, etc.).
3. Also called *functional* testing or “data-driven” or “I/O-driven” testing.
4. The test cases are based on specifications.
5. Black-box testing techniques apply to *all* levels of testing (e.g., unit/ component and system).
6. Test planning can begin early in the software process.
7. Black box testing is practiced widely.

Black Box Testing



Black Box Testing

1. Conducted for Integration testing, System Testing & Acceptance Testing

2. Test Case Design Methods:
    1. Commonly used methods
        1. Equivalence Partitioning Method
        2. Boundary value Analysis
        3. Error guessing
    2. Leaser used methods
        1. Cause Effect Graph
        2. State transition testing

White Box Testing

1. Testing that takes into account internal mechanism of a system or component, types include branch testing, path testing etc

2. Aspects missed out in black box testing are tested

3. Disadvantages:
    1. Does not ensure that user's requirements are met
    2. Does not establish if the decisions / conditions / paths / statements are insufficient

White Box Testing

1. White box testing assumes that the procedural design is known to us.

2. Procedural design can act as driver to test case design.

3. The tester is going to test the chronological order of program.

4. Test cases are designed which can test internal logic of the program.

White Box Testing

Derive test cases that:

1. Exercise all independent execution paths

2. Exercise all logical decisions on both true and false sides

3. Execute all loops at their boundaries and within operational bounds

White Box Testing

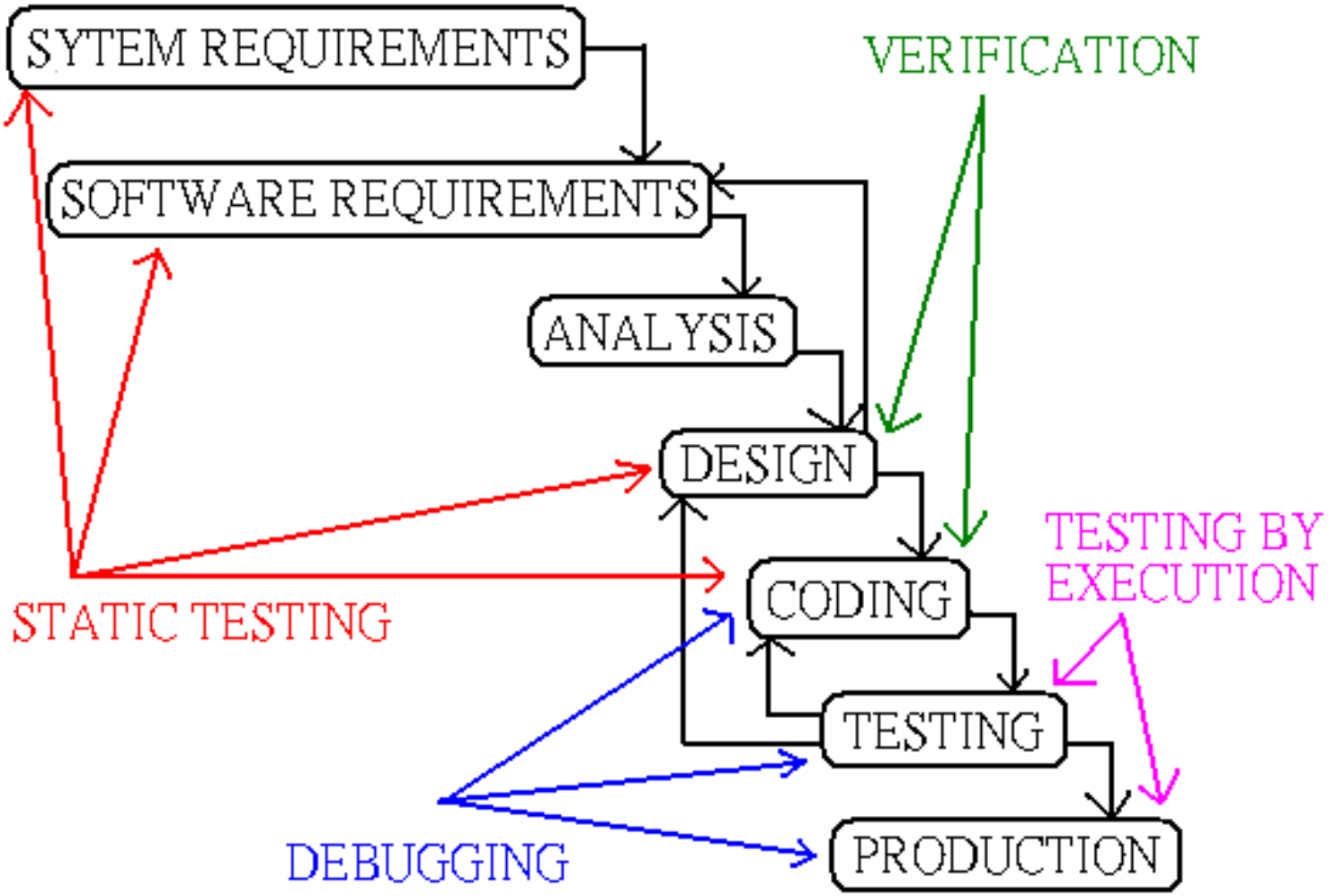
1. Control flow testing:
    1. Statement coverage
    2. Decision coverage
    3. Condition coverage
    4. Multiple condition coverage
    5. Path coverage

Static testing VS Dynamic testing

1. Static Testing
    1. A form of verification that most of the times does not requires execution of software.
    2. Ex. Inspecting documents/artifacts

2. Dynamic Testing
    1. Requires the execution of the software.
    2. Ex. Output Validation

Static and Dynamic Testing



Part 2: Fundamentals of Software Quality

1. What is Quality
2. Quality Culture
3. Process

What is Quality?

1. High levels of user satisfaction and low level of defects often associated with low complexity: **Tom McCabe.**

2. Quality is another name of consistently meeting customer's requirements which is fit for use to its intended functions with reasonable cost and within time.

3. Quality must be measurable and should predictable.

Quality Views

1. Customer's view:
    1. Delivering the right product
    2. Satisfying customer's needs
    3. Meeting the customer's expectations
    4. Treating every customer with integrity, courtesy, respect

Quality Views

2. Supplier's view:
    1. Doing the right thing
    2. Doing it the right way
    3. Doing it right the first time
    4. Doing it on time
3. Any difference between the two views can cause problems

Quality - Productivity

1. Increase in Quality can directly lead to increase in productivity.

2. Rework, Repairs and complaints are key factors affecting Quality and productivity

3. Direct relationship between Quality and cost of the process and product.

Employee Involvement in Quality Program

1. Employee involvement is the cornerstone of quality program
2. Proper communication between management and employees must exist
3. Employees participate and contribute to improvement of processes
4. Employees share the responsibility for innovation and quality improvement

Quality Culture

Quality Culture is	Quality Culture is not
Listening to customers and determining their requirements	Assuming you know the customer's requirements
Identifying cost of quality and focusing on prevention	Overlooking hidden costs of poor quality
Doing things right the first time	Doing things again to make them right
Continuous process improvement	One-time fix
Taking ownership at all levels	Assigning responsibility to one department
Demonstrating leadership and commitment	Assigning responsibility for quality to others

Shift in Focus

QUALITY CONTROL (QC)

QUALITY ASSURANCE (QA)

QUALITY MANAGEMENT SYSTEM (QMS)

Lessons Learned

1. Requirements not easily communicated
2. Requirements change very frequently
3. Software products are unique each time
4. Effect of bad quality is not known immediately
5. Quality objectives vary from product to product

Software Quality: Simple Tips

1. Aim at ‘customer delight’
2. Have measurable objectives
3. Understand requirements accurately; have a thorough traceability of ALL requirements
4. Implement the Plan-Do-Check-Act Cycle in each phase
5. Detect and remove defects as early as possible:
   Prevention is better than cure
6. Follow easy to use standards / conventions for naming, commenting, coding and documentation
7. Start with compiling and analyzing simple metrics

Part 3: People Challenges in Software Testing

1. People Challenges in Testing
2. Skills required in Tester

People Challenges in SW Testing

1. A real tester must take responsibility for improving the testing process
2. Testing is a Professional discipline requiring trained, skilled people
3. Cultivate a positive team attitude of creative destruction
4. Testing is a highly creative and challenging task
5. Programmers and testers are not adversaries: they join hands in a team effort to add value to a program
6. Testers pursue defects, not the people behind the defects

People Challenges in SW Testing

1. Testing tries person's qualities like patience, fairness, ambition, credibility, capability
2. Testing can affect a person's mental and emotional health in view of office politics and inter-personal conflicts

Skills Required in Tester

- Negative Thinking

  Negative thinking helps testers derive the negative user scenarios.

- Reading Skills

  Testers routinely encounter large quantities of data to read and comprehend. At the requirements review stage, testers have to review hundreds of pages of requirements.

Skills Required in Tester

- Communication and Interpersonal Skills

  For a tester, both verbal and written communication are crucial e.g. communication of defects to developers, quality level of product

- Time Management and Effort Prioritization

  Testers have to juggle a lot of tasks like creating test cases, execution, documenting test results, creating test metrics etc. Tester may be involved in testing more than one module or project at the same time.

Chapter-3

Verification and Validation

Verification And Validation

Topics Covered in this lesson are:

- Verification
- Validation
- Techniques of Verification
- V Model

Verification

1. Disciplined approach to evaluate whether a software product fulfils the requirements or conditions imposed on them (Are we doing the job right?)
2. Method: walkthrough, formal inspection and review of each software product
3. Also called static testing
4. Done by systematically reading the contents of a software product with the intention of detecting defects

Verification

5. Helps in identifying not only presence of defects but also their location
6. A ‘filter’ applied at various points during the SDLC to ‘purify’ the product as it progresses through various phases

Verification - Examples

1. Requirement Reviews - Developers, Testers, Users, Managers
2. Design Reviews - Developers, Testers
3. Code Walkthroughs - Developers
4. Code Inspections - Developers

Validation

1. Disciplined approach to evaluate whether the final, as-built software product fulfils its specific intended use (Are We Doing The Right Job?)

2. Method: testing each software product at each phase of life cycle using test plan, test cases. (Unit Testing, Integration Testing, System Testing, Acceptance testing)

3. Also called as dynamic testing

Validation

4. Done by systematically testing a software product with the intention of finding defects

5. Helps in identifying the presence of defects, not their location

6. Necessary to demonstrate not just that the software is doing what it is supposed to do, but also is not doing what it is not supposed to do

Validation Example

1. Unit Testing - Developers
2. Integrated Testing - Developers, Testers
3. System Testing - Testers, Users
4. User Acceptance Testing - Users

Techniques for SW Verification

1. Walkthroughs
2. Inspections
3. Reviews

Walkthrough

1. An informal process, initiated by the author of a software product to a colleague for assistance in locating defects and for suggesting improvements

    1. Normally not planned
    2. Author explains the product
    3. Colleague comes out with observations
    4. Author provides clarification if required
    5. Author notes down relevant points and takes corrective actions

Case Study - Walkthrough

```
#include <stdio.h>
int
main(void)
{
int lower, upper, step;
float fahr, celsius;
lower = 0;
upper = 300;
step = 20;
fahr = lower;
while (fahr <= upper) {
celsius = (5.0/9.0) * (fahr - 32.0);
printf(celsius);
fahr = fahr + step;
}
}
```

Inspection

2. A thorough, word-by-word checking of a software product (or part of a product) with the intention of:

    1. Locating defects

    2. Confirming traceability of relevant requirements

    3. Checking for conformance to relevant standards and conventions

3. The fundamental goal of the inspection process is to eliminate defects from a given, well defined work product.

Inspection Team

1. The inspection team is a group of individuals that work together to analyze each work product of a development activity in order to detect and remove defects.

2. Inspections accomplish this by assigning five different procedural roles to the individuals that make up the team

3. The five procedural roles are: Author, Moderator, Reader, Recorder, and Inspector.

Author…

1. The person who originally constructed the work product.

2. Will address specific questions that arise concerning the content of the work product .

3. Will be ultimately responsible for updating the work product after the inspection.

Moderator…

Responsible for :

- Ensuring that the inspection procedures are performed through out the entire inspection process.
- Verifying the work products readiness for inspection
- Verifying that the entry criteria is met
- Assembling an effective inspection team
- Keeping the inspection meeting on track
- Verifying that the exit criteria is met

Reader & Recorder…

Reader…

- The reader is responsible for leading the Inspection Team through the inspection meeting by reading aloud small logical units, paraphrasing where appropriate.

Recorder…

- will document all defects that arise from the inspection meeting. This documentation will include where the defect was found.

Inspectors…

1. All of the Inspection Team individuals are also considered to play the Inspector role, independent of other roles assigned.

2. The Inspector role is responsible for analyzing and detecting defects within the work product.

Inspection Reports…

1. Four standard reports that should be used anytime an inspection meeting is scheduled, defects are identified, or management reports are prepared.

    1. Inspection Meeting Notice
    2. Inspection Defect List
    3. Inspection Defect Summary
    4. Inspection Management Report

Case Study Inspection

1. Requirement

1. The program shall convert centigrade value to Fahrenheit and vice versa starting from 0 to 300 with interval of 20.

2. The program shall print all centigrade and Fahrenheit values.

Case Study - Inspection

```
#include <stdio.h>
int main()
{
int lower, upper, STEP;
float fahr, celsius;
lower = 0;
upper = 300;
STEP = 20;
fahr = lower;
while (fahr <= upper) {
celsius = (5.0/9.0) * (fahr - 32.0);
printf(celsius);
fahr = fahr + STEP;
}
}
```

Guidelines for Inspections

1. Plan the inspection
2. Allocate trained resources
3. Moderator / inspection leader circulates the product to be inspected and checklist to inspection team in advance
4. Product is clear-compiled, checked for spellings, grammar, formatting, etc.
5. Inspect the product; not the person behind it
6. Manager not to be involved

Guidelines for Inspections

1. Reader reads the product, line by line

2. Concentrate on locating defects; avoid discussions on possible solutions

3. Classify and record the defects or comments

4. Products that can be inspected: SRS, Design, Code,
   User Manual, Installation Manual, Help Screen, Various plans, etc.

Review

1. Follow-up: a subsequent examination of a Product for the purpose of monitoring earlier Changes.

2. Formal or official examination.

3. A process in which one or more persons checks changed documents or data to determine if the changes are correct.

4. An analysis undertaken at a fixed point in time to determine the degree to which stated objectives have been reached. This is generally used as a basis for decision making, including updating plans.

Types Of Reviews

1. In-Process Reviews

2. Milestone Reviews (also called)/ Decision-point/Phase-end Reviews

    1. Test Readiness Review
    2. Test Completion Review

3. Post Implementation Reviews (also called) Post Mortem Reviews

In Process Reviews

1. Review progress towards requirements
2. During a specific period of the development cycle – like design period
3. Limited to a segment of the product
4. Used to find defects in the work product and the work process
5. Catches defects early – where they are less costly to correct.

Decision Point & Phase End Reviews

1. Normally done at the end of a phase of SDLC, when the author/s feel/s that the product is error-free and can go to the next phase

    1. To determine the defects early in the lifecycle
    2. Usually conducted by a Manager
    3. Main purpose is to decide if the product can go the next phase
    4. Includes to check if suitable inspections have been done
    5. After review, the product is ‘base-lined’

Decision Point & Phase End Reviews
Conti…

1. Completion of phase-end review signals beginning of next phase

2. In iterative development methodologies each analysis/design "package" or segment of the application may be in different phases simultaneously

3. Must ensure iterations are sequenced appropriately to minimize defects and re-working previous iterations

Post Implementation Reviews

1. Also known as “Postmortems”
2. Review/ evaluation of the product that includes planned vs. actual development results and compliance with requirements
3. Used for process improvement of software development
4. Can be held up to three to six months after implementation
5. Conducted in a formal format

Classes Of Reviews

1. Informal Review
2. Semiformal Review
3. Formal Review

Informal Reviews

1. Also called peer-reviews
2. Generally one-on-one meeting between author of a work product and a peer
3. Initiated as a request for input
4. No agenda
5. Results are not formally reported
6. Occur as needed through out each phase

Semi-formal Reviews

1. Facilitated by the author
2. Presentation is made with comment at the end or comment made throughout
3. Issues raised are captured and published in a report distributed to participants
4. Possible solutions for defects not discussed
5. Occur one or more times during a phase

Formal Reviews

1. Facilitated by a moderator (not author)
2. Moderator is assisted by a recorder
3. Defects are recorded and assigned
4. Meeting is planned
5. Materials are distributed beforehand
6. Participants are prepared- their preparedness dictates the effectiveness of the review

Formal Reviews

1. Full participation by all members of the reviewing team is required

2. A formal report captures issues raised and is distributed to participants and management

3. Defects found are tracked through the defect tracking system and followed through to resolution

4. Formal reviews may be held at any time

Benefits of Reviews

1. Identification of the anomalies at the earlier stage of the life cycle
2. Identifying needed improvements
3. Certifying correctness
4. Encouraging uniformity
5. Enforcing subjective rules

Management Of V & V

1. Prepare the plans for execution of the process
2. Initiate the implementation of the plan
3. Monitor the execution of the plan
4. Analyze problems discovered during the execution of the plan
5. Report progress of the processes
6. Ensure products satisfy requirements

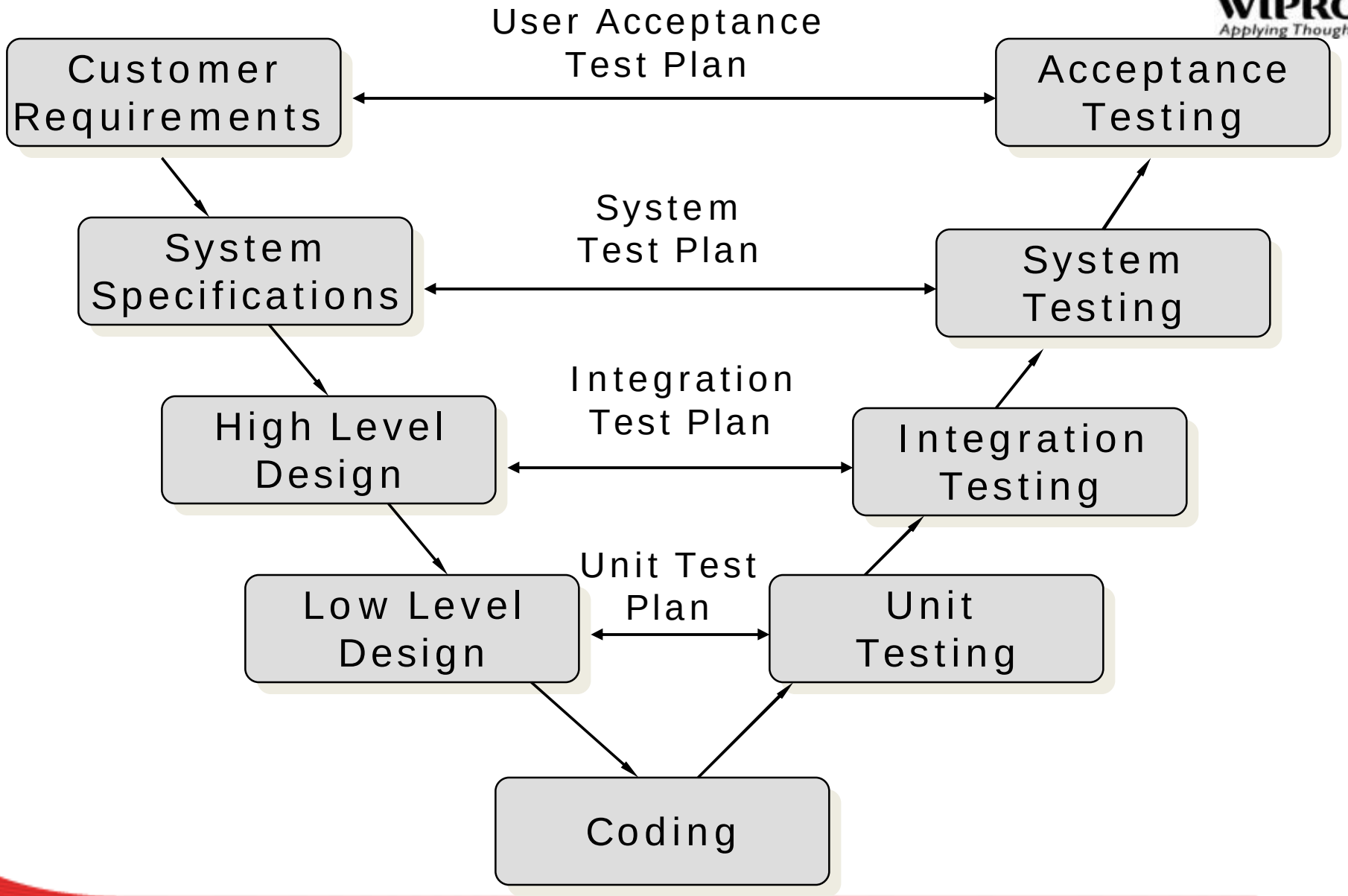
“V” Testing Concept

1. Life Cycle testing: CONTINUOUS testing throughout the SDLC

2. Goes hand-in-hand with formalized system development process

3. Need to plan the testing activities parallel with the SDLC phases

The ‘V’ Model



Chapter-4

Levels of Testing

Levels of Testing

Topics Covered in this lesson are:

1. Levels of Testing
2. Testing Process
    1. Unit
    2. Integration
    3. System
    4. UAT

Levels of Testing

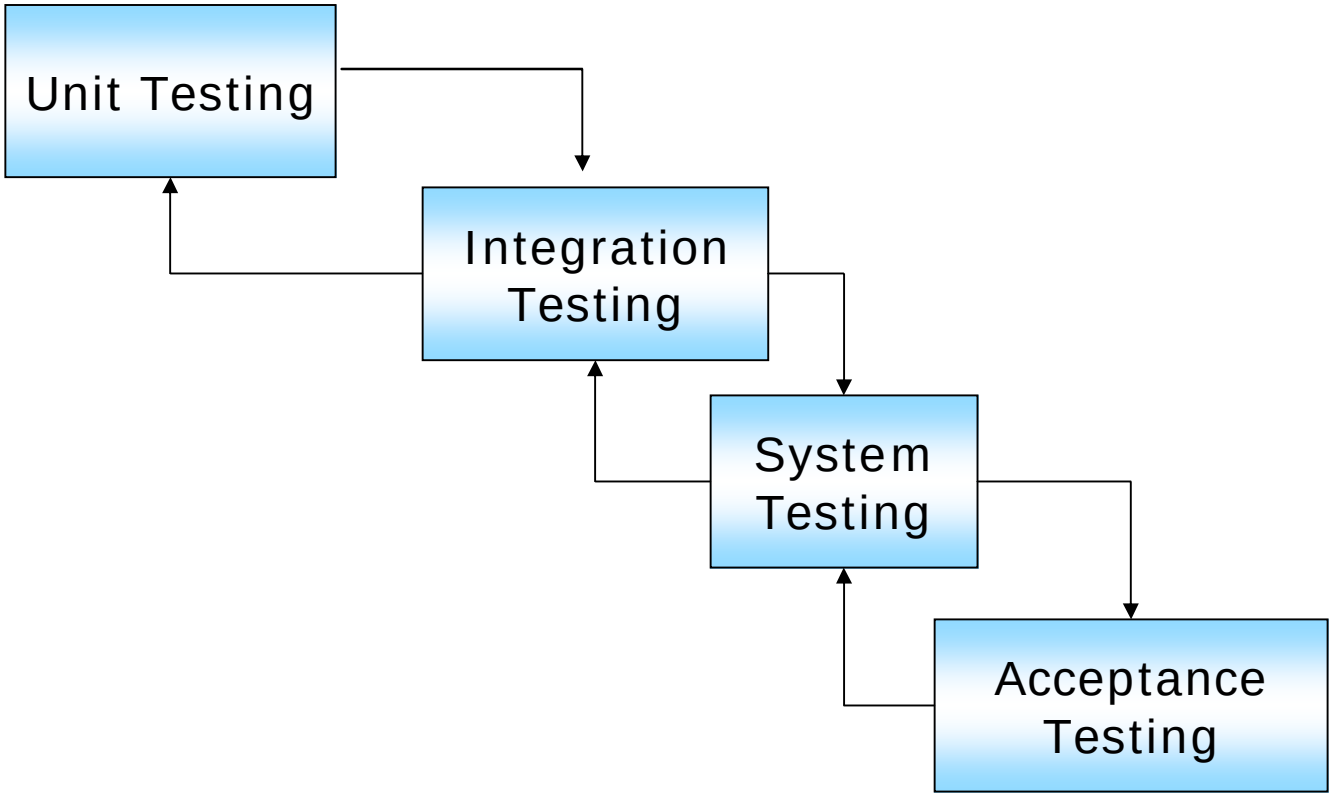
1. Unit Testing - Done by the developer at unit level.

2. Integration Testing - Conducted by the project team integrating the modules

3. System Testing - Conducted by project team or by separate testing team if any

4. Acceptance Testing - Conducted by client either in developer's site or at his site

The Testing Process



Unit Testing

1. The smallest piece of software that can be tested in isolation

2. It is procedure used to validate that individual unit of source code is working properly.

3. Approaches

    1. Black Box
    2. White Box

4. Benefit –
    1. Ensures code meets the requirements
    2. Simplifies integration

Why Unit Testing?

1. Test early for each component and prevent the defect from being carried forward to next stage.
2. To ensure that the design specifications have been correctly implemented.
3. It is inefficient and ineffective to test the system solely as a ‘Big Black Box’
4. The viable approach is to perform a hierarchy of tests
5. Ensure Reasonable and Consistent behaviour at the Lowest level of the Product
6. Experience has shown that Unit Testing is Cost Effective

Unit testing to uncover errors like

1. Comparison of different data types
2. Incorrect logical operators or precedence
3. Expectation of equality when precision errors makes equality unlikely.
4. Incorrect comparison of variables
5. Improper or nonexistent loop termination
6. Failure to exit when divergent iteration is encountered.
7. Improperly modified loop variables, etc.

Potential errors while error handling is evaluated

1. Error description is unintelligible
2. Error notes does not correspond to error encountered
3. Error condition causes system intervention prior to error handling
4. Exception- condition processing is incorrect
5. Error description does not provide enough information to assist in the location of the cause of error.

Unit Testing Procedure

1. Unit testing is normally considered as an adjunct to the coding step.

2. Unit test case design begins ,once the component level design has been developed, reviewed and verified.

3. A review of design information provides guidance for establishing test cases that are likely to uncover errors.

4. Each test case should be coupled with a set of expected results.

Unit Test Steps

1. The Unit test criteria, the Unit test plan, and the test case specifications are defined.

2. A code walkthrough for all new or changed programs or modules is conducted.

3. Unit Test data is created, program or module testing is performed, and a Unit test report is written.

4. Sign-offs to integration testing must be obtained, Sign-off can be provided by the lead programmer, project coordinator, or project administrator.

Unit Testing Activities - 1

1. Field Level Checks

    1. Null / Not Null Checks
    2. Uniqueness Checks
    3. Length Checks
    4. Date field Checks
    5. Numeric Checks
    6. Alphanumeric Checks
    7. Negative Checks
    8. Default Display

Contd ...

Unit Testing Activities - 2

2. Field Level Validation

1. To Test all Validations for an input field
2. Date Range Checks
3. Date Check validation with the system date

3. Functionality Checks

1. Screen Functionality
2. Referential Integrity Checks
3. Field Dependencies

Contd ...

Unit Testing Activities - 3

4. User Interface Checks
   1. Readability of the controls
   2. Tool Tips Validations
   3. Ease of usage of interface across
   4. Consistency with the user interface across the product
   5. User Interface dialogs
   6. Tab related checks for screen controls.

Integration Testing

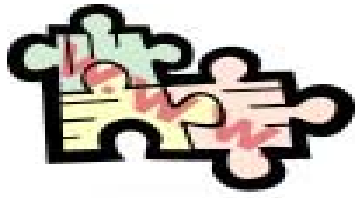
1. Starts at module level when various modules are integrated with each other to form a sub-system or system

2. More stress is given on interfaces between the modules

3. Focuses on design and construction of the software architecture.

4. Four Basic Approaches To Testing While Integrating Modules
    1. Bottom Up
    2. Top Down
    3. Critical Part First
    4. Big Bang

Why integration Testing ?



1. Data can be lost across an interface.
2. One module can have an inadvertent, adverse effect on another.
3. Sub-functions, when combined, may not produce the desired major function.
4. Individually acceptable imprecision may be magnified to unacceptable levels.
5. Global data structures can create problems, and so on…

Integration testing

1. Bottom-up testing strategy

1. The subsystem in the lowest layer of the call hierarchy are tested individually

2. Then the next subsystems are tested that call the previously tested subsystems

3. This is done repeatedly until all subsystems are included in the testing

4. Special program needed to do the testing, Test Driver:
    1. A routine that calls a subsystem and passes a test case to it

Integration testing

2. Bottom-up testing strategy

5. The program is combined and tested from the bottom of the tree to the top

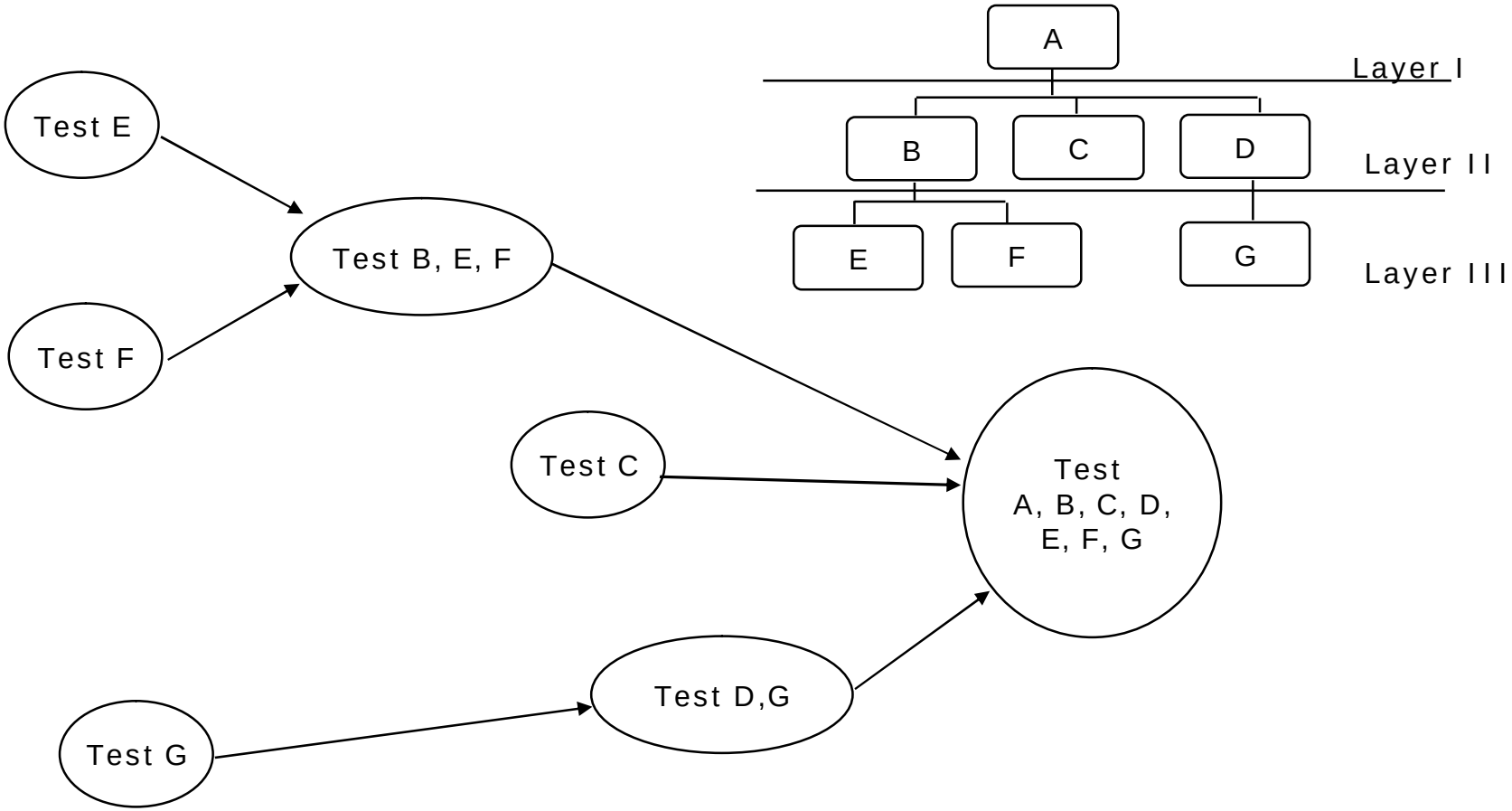
6. Requires a module driver for each module. Put together sub trees and test until whole tree.

7. Very common and effective approach, especially when combined with the Object Oriented Design

8. Each component at the lowest level of the system hierarchy is tested individually first, then next components to be tested.

Integration testing

Bottom-up Integration



Component Driver

1. A special code to aid the integration.

2. A routine that calls a particular component and passes a test case to it.

3. Take care of driver's interface with the test component

Bottom-up Integration Pros and Cons

Pros

1. Advantageous if major flaws occur toward the bottom of the program

2. Test conditions are easier to create

3. Many Programming and testing operations can be carried out simultaneously, yielding apparent improvement in Software Development effectiveness

4. Unit Testing of each module can be done very thoroughly

5. Errors in critical modules are found early

Bottom-up Integration Pros and Cons

Cons

1. Driver modules must be produced

2. The program as an entity does not exist until the last module is added

3. Major control and decision problems will be identified later in the testing process.

4. We cannot test the program in the actual environment in which it will be run.

5. Many modules must be integrated before a working program is available

Integration testing

2. Top-down testing strategy

1. Test the top layer or the controlling subsystem first
2. Then combine all the subsystems that are called by the tested subsystems and test the resulting collection of subsystems
3. Do this until all subsystems are incorporated into the test
4. Special program is needed to do the testing, test stub
    1. A program or a method that simulates the activity of a missing subsystem by answering to the calling sequence of the calling subsystem and returning back fake data.

Integration testing

2. Top-down testing strategy

1. Design, implement and test the top modules using stubs (or dummy modules).

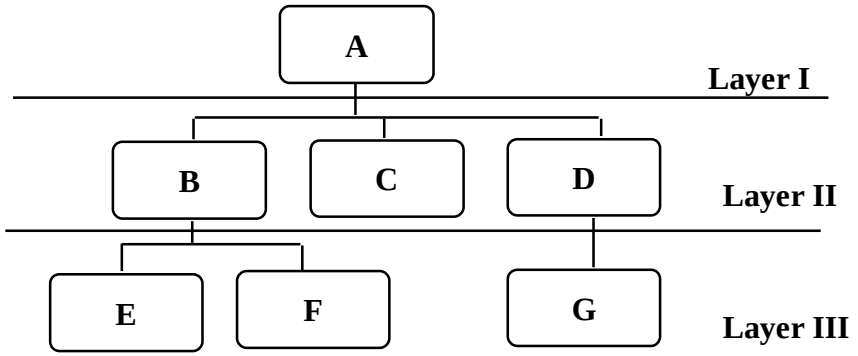
2. Stubs do not perform any real computations or manipulate any real data.

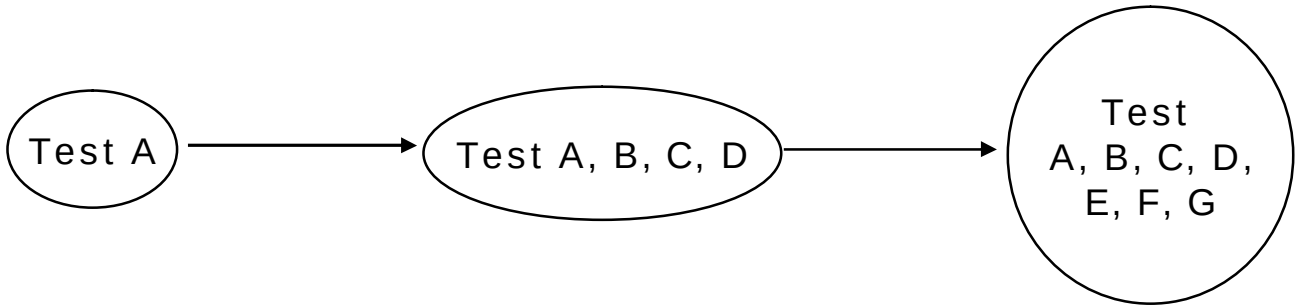
3. Tests are conducted as each component is integrated.

4. Stubs are removed and integration moves downward in the program structure.

Integration testing

Top-down integration testing





Layer I

Layer I + II

All Layers

Stub

1. Problem: A component being tested may call another that is not yet tested!

2. Solution: Write a special-purpose program to simulate the activity of the missing component.

3. The special-purpose program is called a stub.
    1. If the lowest level of components performs the input and output operations, stubs for them may be almost identical to the actual components they replace.

Top-down Testing Pros and Cons

Pros

1. Any design faults or major questions about functional feasibility can be addressed at the beginning of testing instead of the end.
2. Early skeletal program allows demonstrations and boosts morale
3. Integrated Testing is done in an environment that closely resembles that of the reality, so the tested product is more reliable
4. Stubs are functionally simpler than drivers, and therefore they can be written with less time
5. Interface errors are discovered early

Top-down Testing Pros and Cons

Cons

1. Writing stubs can be difficult, because they must allow all possible conditions to be tested.
2. Stub may itself needed to be tested to insure it is correct
3. Very large number of stubs may be required.
4. Unit Testing of lower modules can be complicated by the complexity of upper modules
5. In the initial phase of Testing, it is difficult to do coding and testing simultaneously
6. Errors in critical modules at low levels are found late

Integration testing

3. Sandwich Testing Strategy

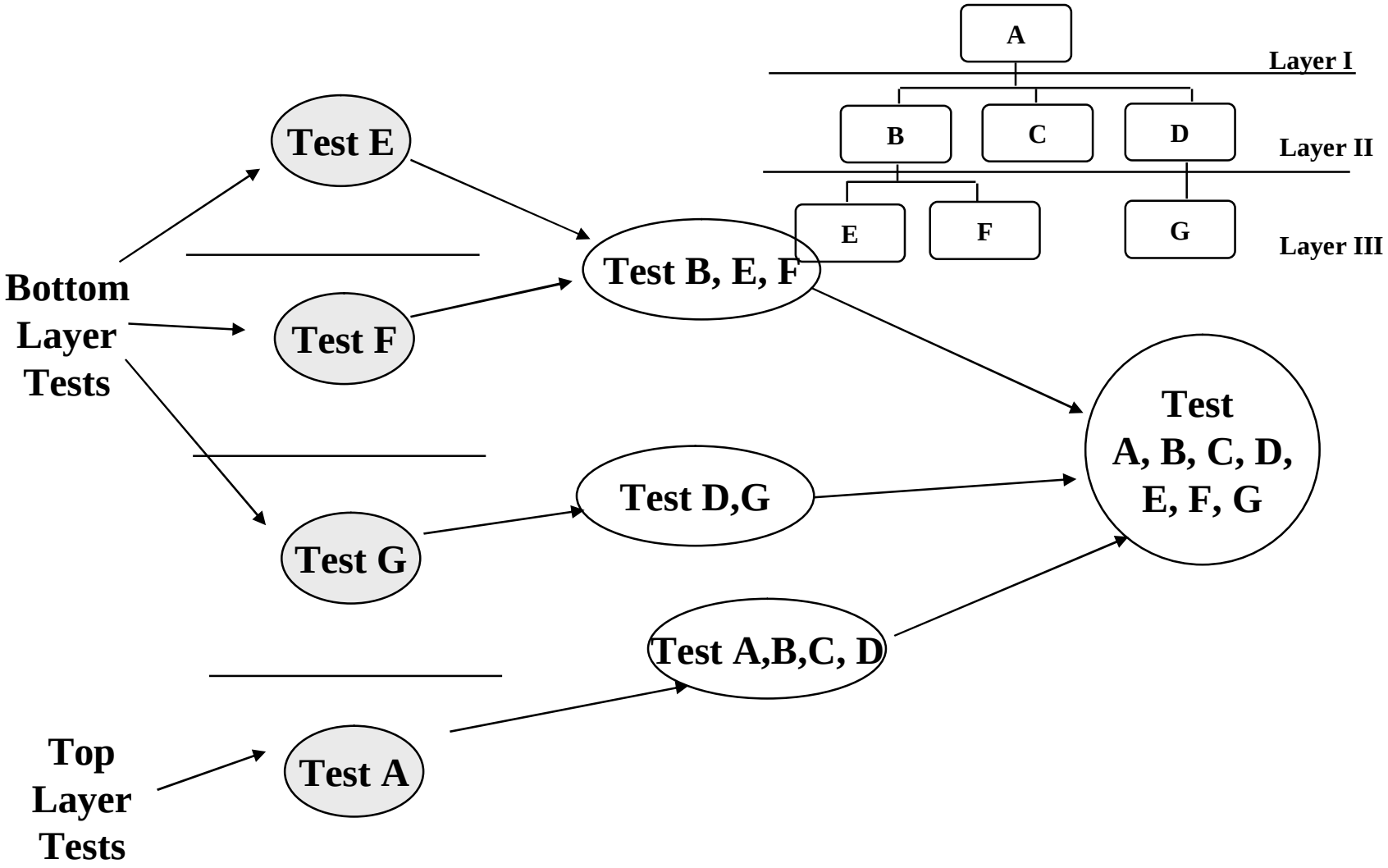
1. Combines top-down strategy with bottom-up strategy

2. The system is view as having three layers
   1. A target layer in the middle
   2. A layer above the target
   3. A layer below the target
   4. Testing converges at the target layer

3. How do you select the target layer if there are more than 3 layers?
   1. Heuristic: Try to minimize the number of stubs and drivers

Integration testing

Sandwich testing strategy



Integration Testing

4. Critical Part First

1. Design, implement and test the critical part of the system first

2. Important for time critical systems, where the performance of critical part makes up for performance of the whole system

Integration Testing

5. Big Bang Testing

1. A common approach in non-process oriented organizations

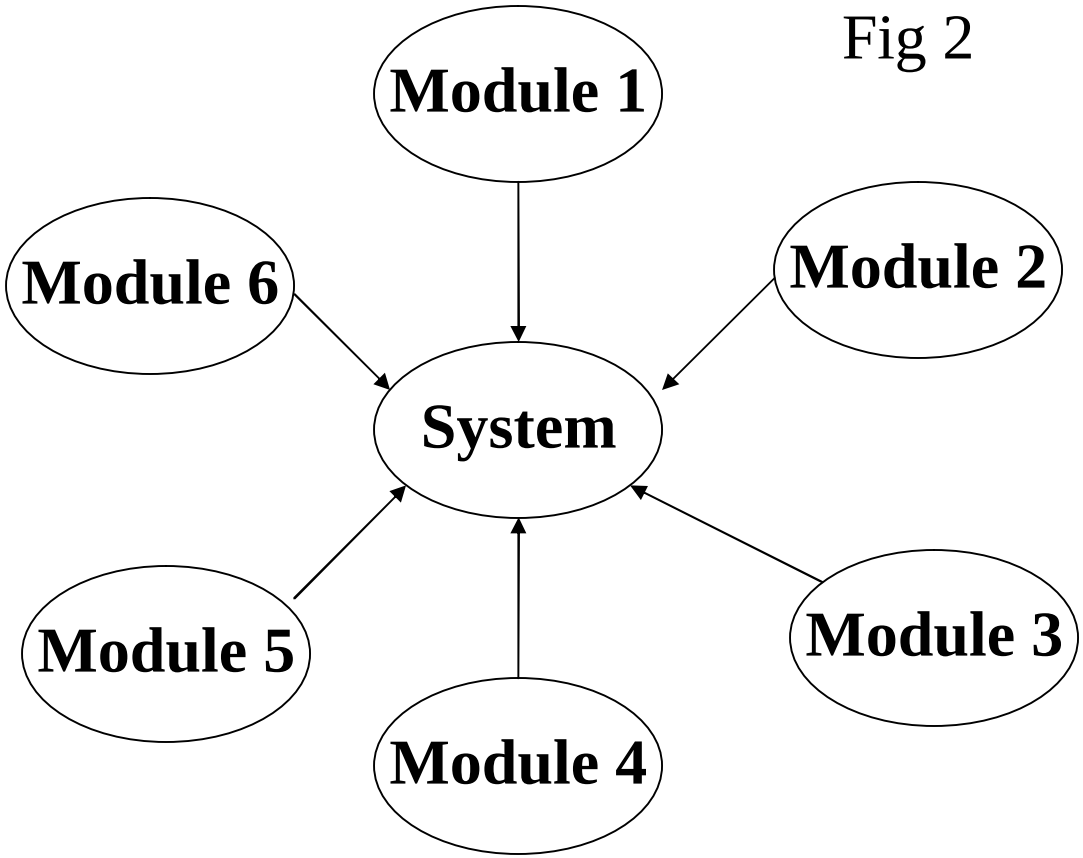
2. All modules are integrated at once

3. Many disadvantages and few advantages !!
    1. Hard to debug, hard to check interfaces
    2. Location of the defect is difficult to find
    3. No stubs and drivers are required

Integration Testing

Big Bang Integration Testing

Fig 2



System Testing

1. System testing is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

2. Validates that the system meets its functional and non-functional requirements

3. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specifications.

4. Final phase of testing before delivery

System Testing – Special Test

Special Test are grouped on the basis of applications under test and the type of tests performed –

1. Requirements Testing
2. Usability testing
3. Load testing
4. Stress testing
5. Performance testing
6. Configuration testing
7. Compatibility testing
8. Security testing
9. Recovery testing
10. Installation testing

System Testing - Other Specialized Tests

1. Regression testing
2. Error-handling testing
3. Manual-support testing
4. Inter-systems testing
5. Smoke testing
6. Sanity testing
7. Parallel testing

System Testing - Specialized Tests

Specialized systems and applications

1. Client-server systems
2. Web based systems
3. Commercial Off-the-shelf software

Requirements Testing

1. Every system is requirement tested
2. Process begins from requirement phase and continues until operations and maintenance phase
3. Objectives include –
    1. User requirements are implemented
    2. Correctness is maintained
    3. Processing complies with organization's policies and procedures.
4. Methods used include –
    1. Creation of a matrix to determine whether all requirements are implemented
    2. Use of checklist to verify whether system meets organizational policies and governmental regulations

Usability Testing

1. Performed to check ease of use of an application
2. To determine –
    1. How simple it is to understand application usage
    2. How easy it is to execute an application process
3. Direct observation of people using the system
4. Usability surveys
5. Beta tests

Usability Testing

1. Checks for human factor problems :
    1. Are outputs meaningful?
    2. Are error diagnostics straightforward?
    3. Does GUI have conformity of Syntax, conventions, format, style abbreviations?
    4. Is it easy to use?
    5. Is there an exit option in all choices?
2. It should not
    1. Annoy intended user in function or speed
    2. Take control from the user without indicating when it will be returned.
3. It should
    1. Provide on-line help or user manual
    2. Consistent in its function and overall design

User Interface Testing

1. Performed to check how user-friendly the application is.
2. To determine whether –
    1. Appropriate input help is displayed on screen
    2. Correct messages are displayed when an error is encountered
    3. Columns have meaningful names
    4. Navigation within the application is easy
3. Should be performed without assistance of system personnel.

Load Testing

1. Load testing is subjecting a system to a statistically representative load.

2. It is done to determine if the system performance is acceptable at pre-determined load level

Stress Testing

1. Whether system continues to function when subjected to large volumes (larger than expected)
2. Areas that may be subjected to test include –
    1. Input transactions
    2. Internal tables
    3. Disk space
    4. Communication channels etc.
3. System should run as it would in production environment.
4. Should be performed when there is uncertainty about volume of expected work that the system can handle.

Performance Testing

1. Is to determine whether the system meets its performance requirements

2. e.g. x transactions should be processed in y seconds or data should be retrieved in z seconds with say 100 concurrent users.

3. Can also be called as compliance testing with respect to performance

4. Design should address performance issues

5. Design verification can help in determining whether required measures have been taken to meet performance requirements

Performance Testing

Performance Testing Process

Step 1 Planning	**Step 2 Execution**	**Step 3 Reporting**
Understand requirements	Setup the environment	Generate custom reports based on: • Simulated Load • Round Time • Response time • CPU Utilization • Memory utilization • Disk utilization Etc
Test Planning	Generate Scripts	
Tool Identification	Conduct the Test	
	Problem Investigation	
Load, Stress and Scalability Testing		

Configuration Tests

1. Determine the effect of adding or modifying resources
    1. Memory
    2. Disk drives
    3. CPU
    4. Network Card
2. Test for compatibility issues
3. Determine minimal and optimal hardware and software configuration

Compatibility Testing

1. Similar to multi-platform testing

2. Performed to ensure that application functions properly on multiple system configurations

3. More significant in case of web-based applications where any browser can be used to access the application.

Security Testing

Planning the test - Can effectively be performed by using penetration point matrix

1. One dimension has potential perpetrators
2. Other dimension has potential points of penetration
3. Intersection has the probability of penetration
    1. Identify potential perpetrators
    2. Identify potential points of penetration
    3. Create a penetration matrix
    4. Identify high risk points of penetration

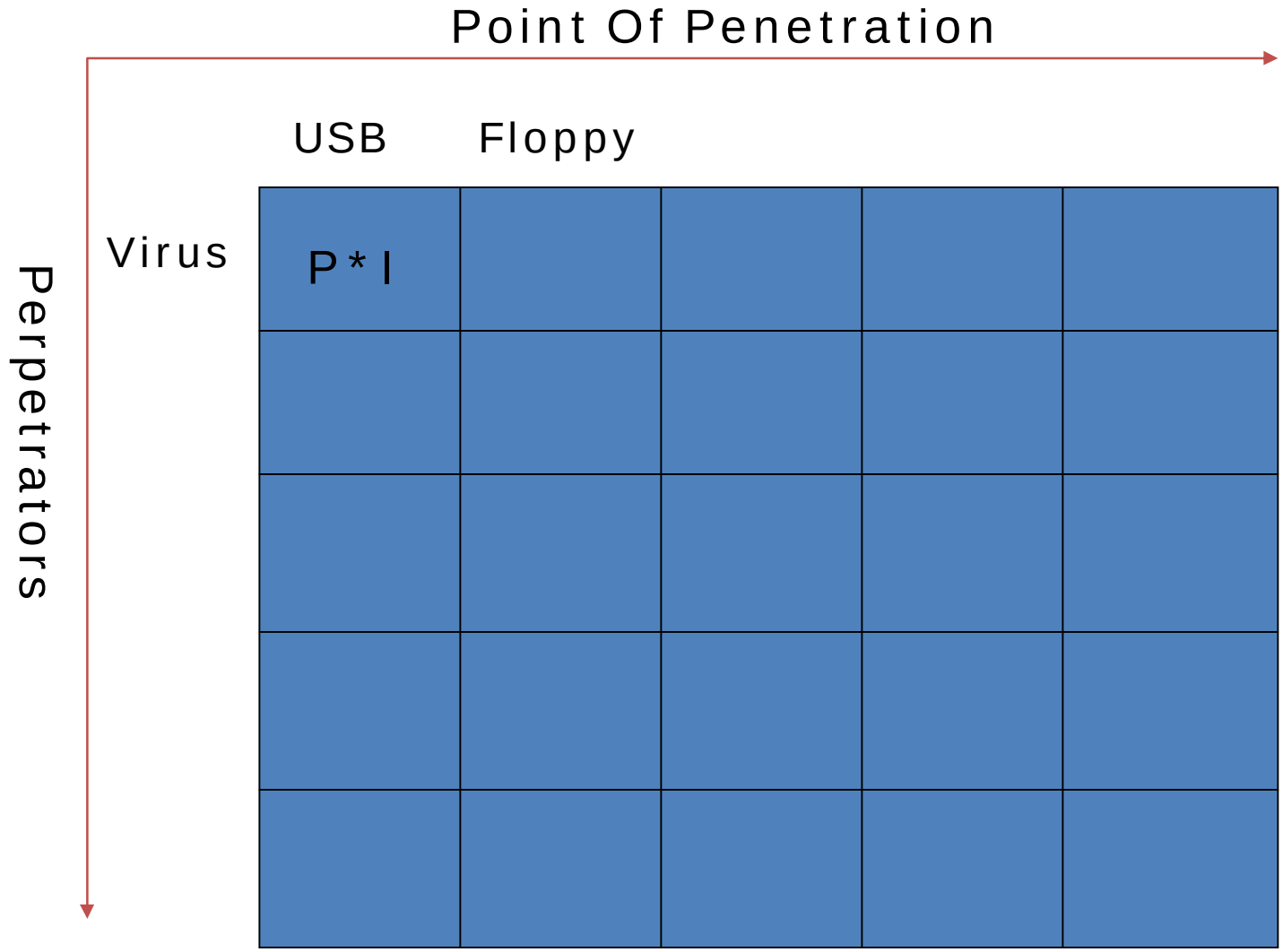
Execute test

Security Testing

1. Devise test cases that subvert the program's security checks:
    1. Obtain passwords
    2. Access idle terminals
    3. Imitate valid users
    4. Guess passwords
    5. Check permissions of different user groups/users
    6. Check database security

2. Best Timing for identifying security flaws
    1. Design Phase

Penetration Matrix



Recovery Testing

1. To determine whether operations can be continued after a disaster or after integrity of the system has been lost

2. Involves reverting to a point where the integrity of the system was known and then re-processing transactions up to the point of failure

3. Is used where continuity of operations is essential

Recovery Testing

1. Determines whether –
    1. Adequate backup data is preserved
    2. Backup data is stored in secured location and is easily retrievable
    3. Recovery process is documented
    4. Recovery personnel are trained
    5. Recovery tools are available

Installation Testing

1. To identify the ways in which installation procedures lead to incorrect results
2. Determines whether –
    1. Installation procedure is documented
    2. Personnel are trained in installation process
    3. Methodology for migration from old system to new system is documented

Other Specialized Tests

Regression testing

1. It is the re-execution of some or all the test cases to check
   1. Any addition to the software
   2. Any deletion to the software
   3. Any updation or fixing of defect

   has not introduce any defect in unchanged part of the software.
2. Can be at –
   1. Unit level
   2. Module level
   3. System level

Regression Testing

3. Is performed when there is a high risk that changes may affect unchanged components
4. Re-running previously conducted tests to ensure that unchanged components function correctly
5. Reviewing previously prepared documents to ensure that they remain correct after changes have been made
6. Purpose of Regression testing
    1. Locate errors
    2. Increase confidence in correctness
    3. Preserve quality
    4. Ensure continued operations
    5. Check correctness of new logic
    6. Ensure continuous working of unmodified portions

Regression testing types

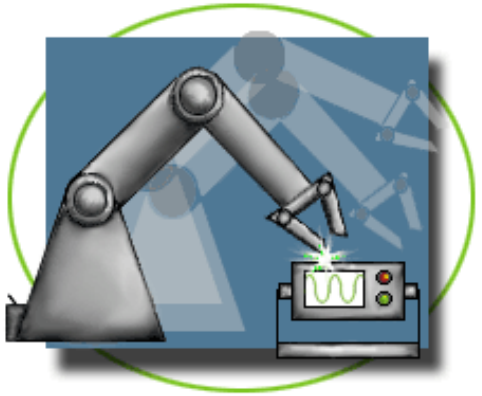
Corrective Maintenance | **Adaptive Maintenance** | **Perfective Maintenance** | **Preventive Maintenance**

1. **Corrective** - fixing bugs, Design Errors, coding errors
2. **Adaptive** - no change to functionality, but now works under new conditions, i.e., modifications in the environment.
3. **Perfective** - adds something new; makes the system "better" ,

   E.g.: adding new modules.
4. **Preventive** – prevent malfunctions or improve maintainability of the software.

   E.g.: code restructuring, optimization, document updating etc

Regression testing methods



1. Regression testing can be done either manually or by automated testing tools.
    1. Manual testing: Can be done for small systems, where investing in automated tools might not be feasible enough.
    2. Automated testing: One class of these tools is called as Capture-playback tool. This is very helpful in situations where the system undergoes lots of version changes.

Error - Handling Testing

1. To determine the ability of the system to properly process erroneous transactions
    1. All reasonably expected errors are recognizable by the application system
    2. Procedures provide high probability that the errors will be corrected properly
    3. Reasonable control over errors during correction
2. Should happen throughout development life cycle

Manual - Support Testing

1. Testing of interface between people and application system
2. To determine whether –
   1. Manual support procedures are documented and complete
   2. Manual support people are adequately trained
   3. Manual support and automated segment are properly interfaced
3. Should occur without assistance of the system personnel
4. Provide input and have manual-support group enter it into the system
5. Prepare output reports and ask people to take necessary action based on those reports

Inter - Systems Testing

1. Testing of interface between two or more application systems
2. To determine whether –
    1. Parameters and data are correctly passed between applications
    2. Documentation for the involved systems is accurate and complete
3. Should be conducted whenever there is a change in parameters between applications
4. Representative set of test transactions is prepared in one system and passed to another system for processing and results are verified for correctness

Smoke Testing

1. Checking the testability of the software is termed as Smoke testing.
2. Also known as a build verification testing / Basic functional testing / Link testing
3. The Smoke test scenarios emphasize breadth more than depth.
4. Smoke tests can either be performed manually or using an automated tool.

Sanity Testing

1. Checking behavior of the system is termed as Sanity testing.
2. Also called as narrow regression testing.
3. Considered as subset of regression testing.
4. Sanity testing is a focused narrow and deep, but cursory
5. Sanity tests are useful both for initial environment validation and future interactive increments

Parallel Testing

1. Comparison of results from two different systems (old v/s new or manual v/s automated)
2. To determine whether –
    1. New version of application or new system performs correctly with reference to existing system that is working correct
    2. There is consistency or inconsistency between two systems
3. Still used while accepting a new system

Parallel Testing

4. Same input data should be used in both systems
5. Input data may be modified as per requirement of new system
6. The new system is used in parallel with the existing system for certain period
7. Thorough cross-checking of the outputs and comparison with outputs from existing system

Compliance Testing

1. With Respect to Processes:-
2. Performed to check whether system was developed in accordance with standards, procedures and guidelines
3. To determine whether –
    1. Development and maintenance methodologies are followed
    2. Completeness of system documentation
4. Depends upon the management's desire to have the standards enforced
5. Use of checklists
6. Peer reviews
7. SQA reviews
8. Internal audits

Specialized Systems and Applications

Client - Server Systems

1. Key components of client-server technology

    1. Client installation
    2. Security
    3. Data
    4. Standards

2. Testing should be adjusted to address these four key components

Web - Based Systems

1. Key factors
    1. Security
    2. Performance
    3. Correctness
    4. Compatibility
    5. Reliability
    6. Data integrity
    7. Usability
    8. Recoverability

Web - Based Systems

1. Security
    1. External intrusions
    2. Secured transactions
    3. Viruses
    4. Access control
    5. Authorization

2. Performance (Load testing)
    1. Concurrency
    2. Stress
    3. Throughput

Web - Based Systems

3. Correctness
    1. Functionality
    2. Calculations
    3. Navigation
4. Compatibility
    1. Operating system
    2. Browser
5. Reliability
    1. Consistently correct results
    2. Server and system availability

Web - Based Systems

6. Data integrity
    1. Correct data is accepted
    2. Data stays in correct state
7. Usability
    1. Easy to use and understand
    2. Navigation is correct
8. Recoverability
    1. Lost connections
    2. Client system crashes
    3. Server system crashes

Commercial Off - the - Shelf Software

1. Define business needs in a manner that can be used to evaluate software capabilities
2. Match capabilities against business needs
3. Check critical success factors like ease of use, maintainability, cost effectiveness, transferability, reliability, security etc, are present in the software
4. Check operational fitness i.e. compatibility with hardware, OS and other packages
5. Check whether software can be used within organization's business work flow
6. Evaluate whether people do have necessary skills to effectively use the software

Acceptance Testing

1. Final stage of testing before the system is accepted for operational use
2. Done with data supplied by the client
3. Validates –
    1. User Needs (Functional)
    2. System Performance (Non-Functional)
4. Is the process of comparing a program to its requirements
5. Testing the system with the intent of confirming readiness of the product and customer acceptance.

Acceptance Test

1. The performance and reliability of the system will be tested and confirmed.

2. Formal testing conducted to enable a user, customer or other authorized entity to determine whether to accept a system or component.

3. An acceptance test is a test that the user defines, to tell whether the system as a whole works the way the user expects.

Acceptance Testing

1. Purpose
    1. The purpose of acceptance testing is to verify system from user perspective
2. Assumptions/Pre-Conditions
        1. Completed system and regression testing
        2. Configuration Manager
        3. Test data
        4. Final versions of all documents ready
        5. Overview to the testing procedures
        6. Exit decision
        7. Specific procedures
        8. Acceptance Criteria MUST be documented
            1. Acceptance Testing
            2. Project stakeholders

Acceptance Testing

1. Expectations
    1. Verification from the user's perspective
    2. Performance testing should be conducted again
    3. Extra time
    4. User manual to the testers
    5. Non-testable requirements
    6. Review with the Sponsor and User
    7. Plans for the Implementation

Acceptance Testing - Alpha Testing

1. Tested at developer site by customer

2. Developer "looks over shoulder" and records errors & usage problems

3. Tests conducted in a controlled environment

Acceptance Testing - Beta Testing

1. Beta testing conducted at one or more customer sites by end user of software

2. Live application environment cannot be controlled by developer

3. Customer records all problems encountered and reports to developer at regular intervals

Chapter-5

Test Planning

Test Planning

Topics Covered in this lesson are:

1. What is test plan
2. Template for test plan
3. Guidelines for creating the Test Plan

Test planning

1. Without plan we don’t know where we are going
2. How we are going to do it
3. How long will it take
4. Resources required
5. Cost involved

Why to plan tests?

1. Documented tests are repeatable.
2. Documented tests are controllable.
3. Coverage can be established.
4. Test plan is dynamic
5. Test plan defines overall objectives and approach

Why Plan Tests

1. If you fail to plan ! Plan To Fail !!

2. Test Plan:

   It talks about test scope, test objectives, test team, goals and responsibilities of Testing members, test case design techniques to be used, defect management tool To be used, risk assessment and mitigation, test strategies, And various criteria like test entry criteria, test suspension criteria, and test exit Criteria.

   It will also talks about the time lines of testing activities in a summarized format & Types of metrics to be collected.

Scope : Features to be tested.
Features not to be tested.

Test Plan Template

1. Test Plan Identifier
2. Introduction
3. Test Scope
4. Test Objectives
5. Assumptions
6. Risk Analysis
7. Test Design
8. Roles & Responsibilities
9. Test Schedule & Resources
10. Test Data Management
11. Test Environment
12. Communication Approach
13. Test Tools

Test Plan

1. Test Plan Identifier

   Its an Unique identifier for the test plan with version for which we are preparing the plan

2. Introduction
   1. Overview of System X
   2. Purpose of this Document
   3. Formal Reviewing

Test Scope

3. Test Scope
    1. Answers two important questions:
        1. What Will Be Covered In The Test
        2. What Will Not Be Covered In The Test
    2. Includes
        1. Functional or Structural Requirements
        2. System Interfaces
        3. Infrastructure Components
        4. Application Documentation

Test Objectives

4. Test Objective

1. A test objective is simply a testing “goal.”
2. It is a statement of what the tester is expected to accomplish or validate during a specific testing activity
3. Test objectives:
    1. Guide the development of test cases, procedures, and test data
    2. Enable the tester and project managers to gauge testing progress and success
    3. Enhance communication both within and outside of the project team by helping to define the scope of the testing effort

Test Objectives

4. Each objective should include a high-level description of the expected test results in measurable terms, and should be prioritized
5. In cases where test time is cut short,
   test cases supporting the highest priority objectives would be executed first.

Assumptions

5. Assumptions

1. Assumptions Document test prerequisites, which if not met could have negative impact on the test
2. Examples –
    1. Skill Level of resources
    2. Test Budget
    3. State of application at start of testing
    4. Tools available
    5. Availability of test equipment

Risk Analysis

6. Risk Analysis

1. It documents test risks and their possible impact on the test effort
2. Examples:
    1. New Technology
    2. New Test Automation Tool
    3. Sequence and increments of code delivery
    4. Availability of Application Test Resources

Risk Analysis

3. Testing is risk based activity
4. All type of test planning considers reducing the risks to acceptable level
5. Risks can not be eliminated, but the occurrence and /or the impact of loss can be reduced
6. Three methods of measuring magnitude of risk
    1. Intuition / Judgment – One or more individuals state that they believe the risk is greater than the cost of controls
    2. Consensus – A team or group of people agrees to the severity of magnitude of a risk.
    3. Risk Formula –>

        Risk = Probability or Frequency * Loss per occurrence

Test Design

7. The test design details the following:
    1. Policy
        1. The types of tests that must be conducted
        2. The stages of testing that are required
        3. E.g., Unit, Integration, System, Performance, and Usability
    2. Strategy
        1. Outlines the sequence and timing of tests.

Roles & Responsibilities

8. Roles & Responsibility

1. Define who is responsible for each stage of testing
2. Generate a responsibility matrix

Worker	Resources	Specific Responsibilities/Comments
Test Manager	Mini L.	Responsibilities: • Provide technical direction • Acquire appropriate resources • Management reporting
Tester	Yuvi B. Priya Sv.	Responsibilities: • Execute tests • Log results • Recover from errors • Document defects

Roles & Responsibilities

1. Test Manager
    1. Single point contact between Wipro onsite and offshore team
    2. Prepare the project plan
    3. Test Management
    4. Test Planning
    5. Interact with Wipro onsite lead, Client QA manager
    6. Team management
    7. Work allocation to the team
    8. Test coverage analysis

Roles & Responsibilities

2. Test Manager cont..
    1. Co-ordination with onsite for issue resolution.
    2. Monitoring the deliverables
    3. Verify readiness of the product for release through release review
    4. Obtain customer acceptance on the deliverables
    5. Performing risk analysis when required
    6. Reviews and status reporting
    7. Authorize intermediate deliverables and patch releases to customer.

Roles & Responsibilities

3. Test Lead
    1. Resolves technical issues for the product group
    2. Provides direction to the team members
    3. Performs activities for the respective product group
    4. Review and Approve of Test Plan / Test cases
    5. Review Test Script / Code
    6. Approve completion of Integration testing
    7. Conduct System / Regression tests
    8. Ensure tests are conducted as per plan
    9. Reports status to the Offshore Test Manager

Roles & Responsibilities

4. Test Engineer
    1. Development of Test cases and Scripts
    2. Test Execution
    3. Result capturing and analysing
    4. Defect Reporting and Status reporting

Test Schedule & Planned Resources

9. Test Schedule and Planned Resources
    1. Test Schedule Includes
        1. Major test activities
        2. Sequence of tests
        3. Dependence on other project activities
        4. Initial estimates for each activity
    2. Test Resource Planning Includes
        1. People
        2. Tools
        3. Facilities

Test Data Management

10. Test Data Management

1. Define the data required for testing as well as infrastructure required to manage test data
2. Includes –
    1. Methods for preparing test data
    2. Backup and Rollback Procedure
    3. Data Requirements
    4. Data Security Issues

Test Environment

11. Test Environment

1. Environment requirements for each stage and type of testing should be outlined in this section of the plan, for example:
    1. Unit testing may be conducted in the development environment, while separate environments may be needed for integration and system testing.
    2. Procedures for configuration management and release and version control should be outlined.
    3. Requirements for hardware and software configurations.
    4. The location of individual test events.
    5. The defect tracking mechanisms to be used.

Communication Approach

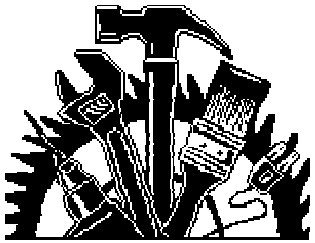
12. Communication Approach

1. In the complex, matrix environment required for testing in most companies, various communication mechanisms are required.
2. These avenues should include
    1. Formal and informal meetings.
    2. Working sessions.
    3. Processes, such as defect tracking.
    4. Tools, such as issue and defect tracking, electronic bulletin boards, notes databases, and Intranet sites.
    5. Techniques, such as escalation procedures or the use of white boards for posting current state of testing (e.g., test environment down).
    6. Miscellaneous items such as project contact lists, meeting audiences, and frequency of defect reporting.

Tools

13. Tools

1. All tools that are needed to support testing process
2. Tools used for:
    1. Test Management
    2. Configuration Management
    3. Test Script Development
    4. Automated Test Tools
    5. Stress / Load Testing tools
    6. Defect Tracking Tools



Guidelines for Developing Test Plan

1. Start Early
2. Keep The Test Plan Flexible
3. Frequently Review The Test Plan
4. Keep The Test Plan Concise & Readable
5. Calculate The Planning Effort
6. Spend The Time To Do A Complete Test Plan

Chapter-6

Test Design

Test Design

Topics Covered in this lesson are:

1. Importance
2. Test Design Essentials
3. Good Test Case
4. Test Case Mistakes
5. Test Case Template
6. Test Design Stages
7. White Box Testing / Structural testing
8. Equivalence Partitioning and Boundary Value Analysis
9. Test Data

Importance of Test Design

1. Foundation to design and develop test script
2. Greater confidence in Quality
3. Completeness of test
4. Estimation of Test Effort

Test Design Essentials

1. Test cases covers all features
2. There is a balance between normal, abnormal, boundary and environmental test cases
3. There is a balance between black box and white box testing
4. There is a balance between functional tests and non- functional tests
5. Finally, documenting test cases

Test Cases and Test Data

1. Test Cases and Test Data

    1. Test data are inputs that have been devised to test the system

    2. Test Cases are inputs and outputs specification plus a statement of the function under test.

    3. Test data can be generated automatically (simulated) or real (live).

What Is A Good Test Case

1. **Accurate** - tests what it's designed to test
2. **Economical** - no unnecessary steps
3. **Repeatable, reusable** - keeps on going
4. **Traceable** - to a requirement
5. **Appropriate** - for test environment, testers
6. **Self standing** - independent of the writer
7. **Self cleaning -** picks up after itself

How To Write A Good Test Cases

1. Testability – Easy to test
2. Use active case, do this, do that , System displays this, does that
3. Simple conversational language
4. Exact, consistent names of fields, not generic
5. Don’t explain windows basics
6. Order of cases follows business scenarios

What a Good Test Case Does

1. An excellent test case satisfies the following criteria:
    1. Reasonable probability of catching an error
    2. Exercises an area of interest
    3. Doesn’t do unnecessary things
    4. Neither too simple nor too complex
    5. Not redundant with other tests
    6. Makes failures obvious
    7. Allows isolation and identification of errors

Test Cases Mistakes

1. Seven most common mistakes –
    1. Making cases too long
    2. Incomplete, incorrect, or incoherent setup
    3. Leaving out a step
    4. Naming fields that changed or no longer exist
    5. Unclear whether tester or system does action
    6. Unclear what is pass or fail result
    7. Failure to clean up

Test Case Design Stages

1. Identify test resources.
2. Identify conditions to be tested.
3. Rank test conditions.
4. Select conditions for testing.
5. Determine correct results of processing.
6. Create test cases.
7. Document test conditions.

Template for Test Case

In order to select the fields that we will use in our template, let us first identify all possible field choices for the template –

1. Project Name, Test Suite ID and Name
2. Version date, Version Number, Version Author
3. Approval and distribution date
4. Revision History with reasons for update
5. Environment pre-requisites (Installation & Network)

Template for Test Case

6. Test Pre-conditions (Data created before testing)
7. Test Case Name , Test Case Number
8. Type Of Testing (I.e. Functional, Load etc.), Objectives
9. Valid or invalid conditions
10. Input Data (ID type, values)
11. Test Steps
12. Expected Results

Case Study -

1. Improving quality and testability of test cases
2. Case Before –
    1. Testers had poor reputation
    2. Testers were frustrated, managers saw no value
    3. Tests rambled, purpose unclear, order random
    4. Setups were wrong
    5. Tests were not maintained to match software
    6. Couldn’t tell pass or fail

Case Study -

1. Improving quality and testability of test cases
2. Case Study After –
    1. Tests were better written
    2. Fewer, shorter steps written in active case
    3. Descriptions, Setup information gave insight into how software worked
    4. Case order matched business scenarios
    5. Tests well maintained
    6. Was easy to decide pass or fail

Building Test Data

1. Data
    1. Synthetic
    2. Real

Test Case Design

1. Use White Box techniques
    1. Statement coverage
    2. Edge coverage
    3. Condition coverage
    4. Path coverage

2. Use Black Box techniques
    1. Equivalence Class / Partitioning
    2. Boundary Value Analysis
    3. Error guessing
    4. State Transition Diagram
    5. Cause Effect Graph

White Box Test Case Design

1. Statement Coverage
2. Edge Coverage
3. Condition Coverage
4. Path Coverage
5. Cyclomatic Complexity

Purpose

1. Understand the Objective
2. Effective conversion of specifications
3. Checking Programming Style with coding standards
4. Check Logic Errors
5. Incorrect Assumptions
6. Typographical Errors

Code Based Testing - White Box Testing

1. Coding Standards
2. Logic Programming Style
3. Complexity of Code
4. Structural Testing
5. Ensure Reduced Rework
6. Quicker Stability
7. Smooth Acceptance
8. Valuable Source
9. Selecting test cases

Code Based Testing or White Box Testing

1. Testing control structures of a procedural design.

2. Can derive test cases to ensure:

    1. All independent paths are exercised at least once.

    2. All logical decisions are exercised for both true and false paths.

    3. All loops are executed at their boundaries and within operational bounds.

    4. All internal data structures are exercised to ensure validity.

Contd..2

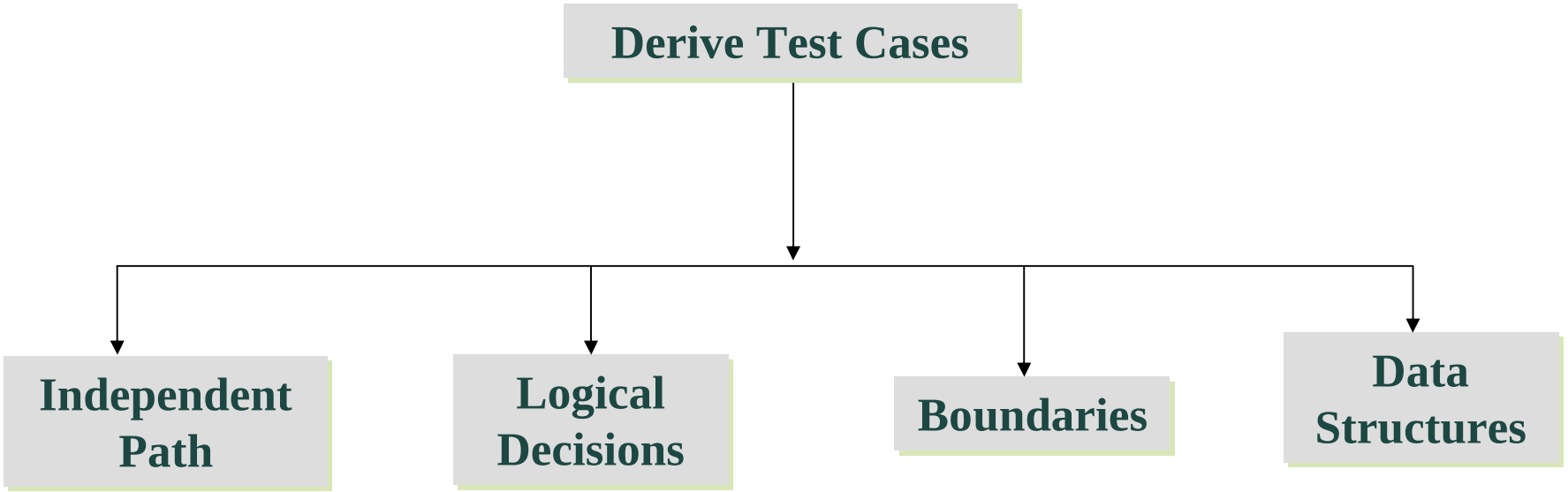
Code Based Testing or White Box Testing

1. Why do white box testing when black box testing is used to test conformance to requirements?
    1. Logic errors and incorrect assumptions most likely to be made when coding for "special cases". Need to ensure these execution paths are tested.
2. May find assumptions about execution paths incorrect and so make design errors. White box testing can find these errors.
3. Typographical errors are random. Just as likely to be on an obscure logical path as on a mainstream path.
    1. "Bugs lurk in corners and congregate at boundaries"

Types of Code Based Testing & Adequacy Criteria

1. Involve Control Flow Testing
    1. Statement Coverage
        1. Is every statement executed at least once?
    2. Edge Coverage
        1. Is every edge in the control flow graph executed?
    3. Condition Coverage
        1. Is edge + every Boolean (sub) expression in the control flow graph executed?
    4. Path Coverage
        1. Is every path in the control flow graph executed?
    5. Cyclomatic Complexity
        1. Is the logical structure of the program appropriate?

Test Cases



Statement Coverage

1. Control Flow elements to be exercised in statements.

2. Statements coverage criterion requires elementary statement, where program is executed at least once.

$$\text{Statement coverage (C)} = \frac{\text{Number of Executed Statements (P)}}{\text{Total Number of Statements (T)}}$$

Edge Coverage (Branch Coverage)

1. Focus is on identifying test cases executing each branch at least once.

$$\text{Edge Covers (C)} = \frac{\text{Number of Executed Branches (P)}}{\text{Total Number of Branches (T)}}$$

Conditions Coverage

1. Combination of Edge Coverage and more detailed conditions.

2. Examples: True & False, Elementary Conditions, Comparisons, Boolean Expressions.

$$\text{Basic Conditions Coverage (C)} = \frac{\text{Number of Executed Conditions (P)}}{\text{Total Number of Conditions (T)}}$$

Conditions Coverage (Contd.)

3. Errors in expressions can be due to:
    1. Boolean operator error
    2. Boolean variable error
    3. Boolean parenthesis error
    4. Relational operator error
    5. Arithmetic expression error

4. Condition testing methods focus on testing each condition in the program.

Conditions Coverage (Contd.)

5. Strategies proposed include:

    1. Branch testing - execute every branch at least once.

    2. Domain Testing - uses three or four tests for every relational operator.

    3. Branch and relational operator testing - uses condition constraints.

Conditions Coverage (Contd.)

Example 1: C1 = B1 & B2

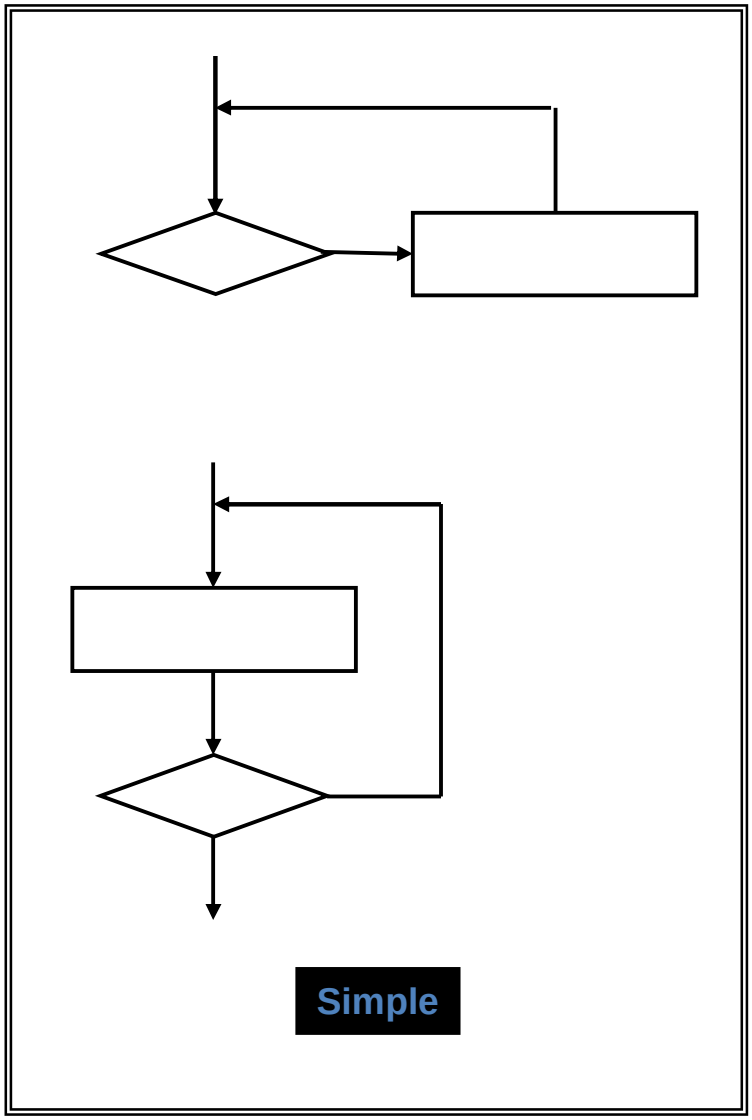
- where B1, B2 are boolean conditions.
- Condition constraint of form (D1,D2) where D1 and D2 can be true (t) or false(f).
- The branch and relational operator test requires the constraint set {(t,t),(f,t),(t,f)} to be covered by the execution of C1.

Coverage of the constraint set guarantees detection of relational operator errors

Path Coverage : Data Flow Testing

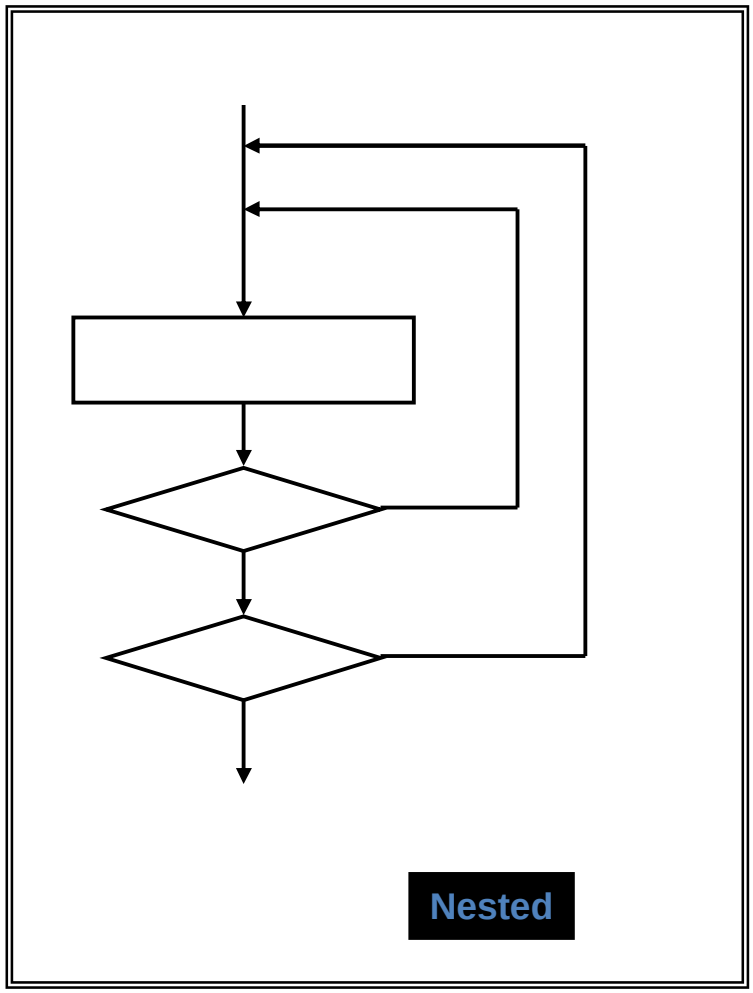
1. Path Coverage executed at least once.
    1. Selects test paths according to the location of definitions and use of variables.
2. Test for Loops (iterations)
    1. Loop Testing.
    2. Loops fundamental to many algorithms.
    3. Can define loops as simple, concatenated, nested and unstructured.

Path Coverage: Simple Loops



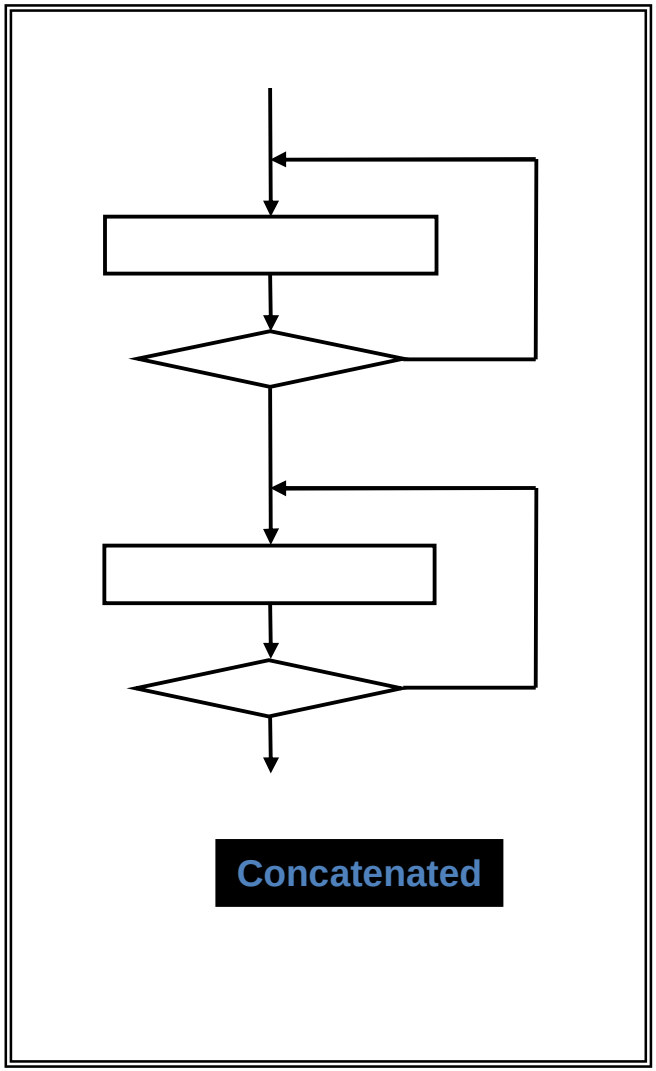
1. Simple Loops of size n:
2. Only one pass through loop
3. Two passes through loop
4. m passes through loop
   where, m<n.
   1. (n-1), n and (n+1) passes through the loop.

Path Coverage: Nested Testing



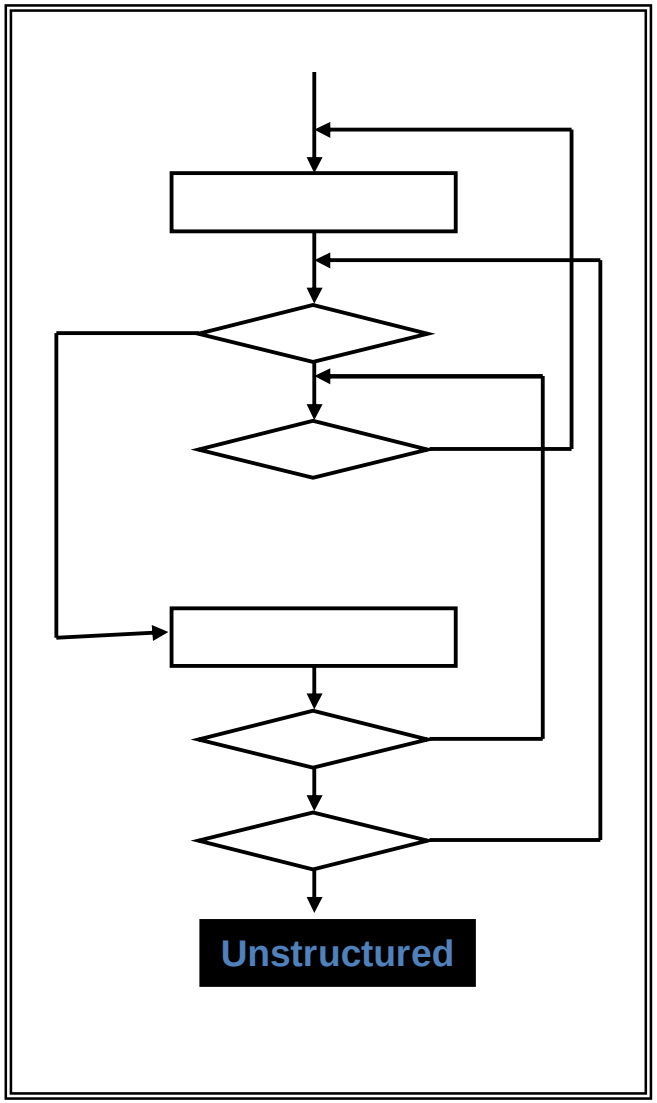
1. Nested Loops
    1. Start with inner loop. Set all other loops to minimum values.
    2. Conduct simple loop testing on inner loop.
    3. Work outwards.
    4. Continue until all loops are tested.

Path Coverage: Concatenated Loop



1. Concatenated Loops test
    1. If independent loops, use simple loop testing.
    2. If dependent, treat as nested loops.

Path Coverage: Unstructured Loops



1. Unstructured loops
    1. Don't test - redesign.

Cyclomatic complexity

1. Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program.

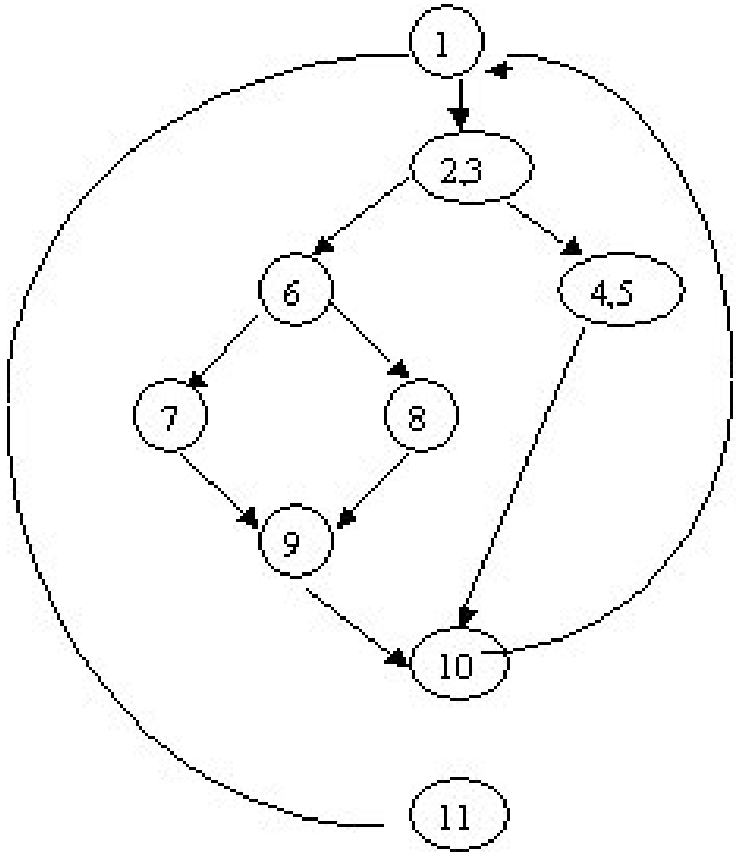
2. When used in the context of basis path testing method, the value computed for Cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed once.

Cyclomatic complexity

3. An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.

4. When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.

Cyclomatic complexity



Cyclomatic complexity

1. For example, a set of independent paths for the flow graph given below
    1. Path 1: 1-11
    2. Path 2: 1-2-3-4-5-10-1-11
    3. Path 3: 1-2-3-6-8-9-10-1-11
    4. Path 4: 1-2-3-6-7-9-10-1-11

Cyclomatic complexity

1. Cyclomatic complexity has a foundation in graph theory and provides us with an extremely useful software metric. Complexity is computed in one of the three ways.

1. The number of regions of the flow graph corresponds to the Cyclomatic complexity.

2. Cyclomatic complexity, V(G), for a flow graph,
   G is defined as
   $V(G) = E - N + 2$

3. Cyclomatic complexity, V(G) for a flow graph G is also defined as
   $V(G) = P + 1$

Where P is the number of predicate nodes contained in the flow graph G.

Cyclomatic complexity

1. The flow graph has four regions.

2. V(G) = 11 edges – 9 nodes + 2 = 4

3. V(G) = 3 predicate nodes + 1 = 4

Relationship with Programming Complexity

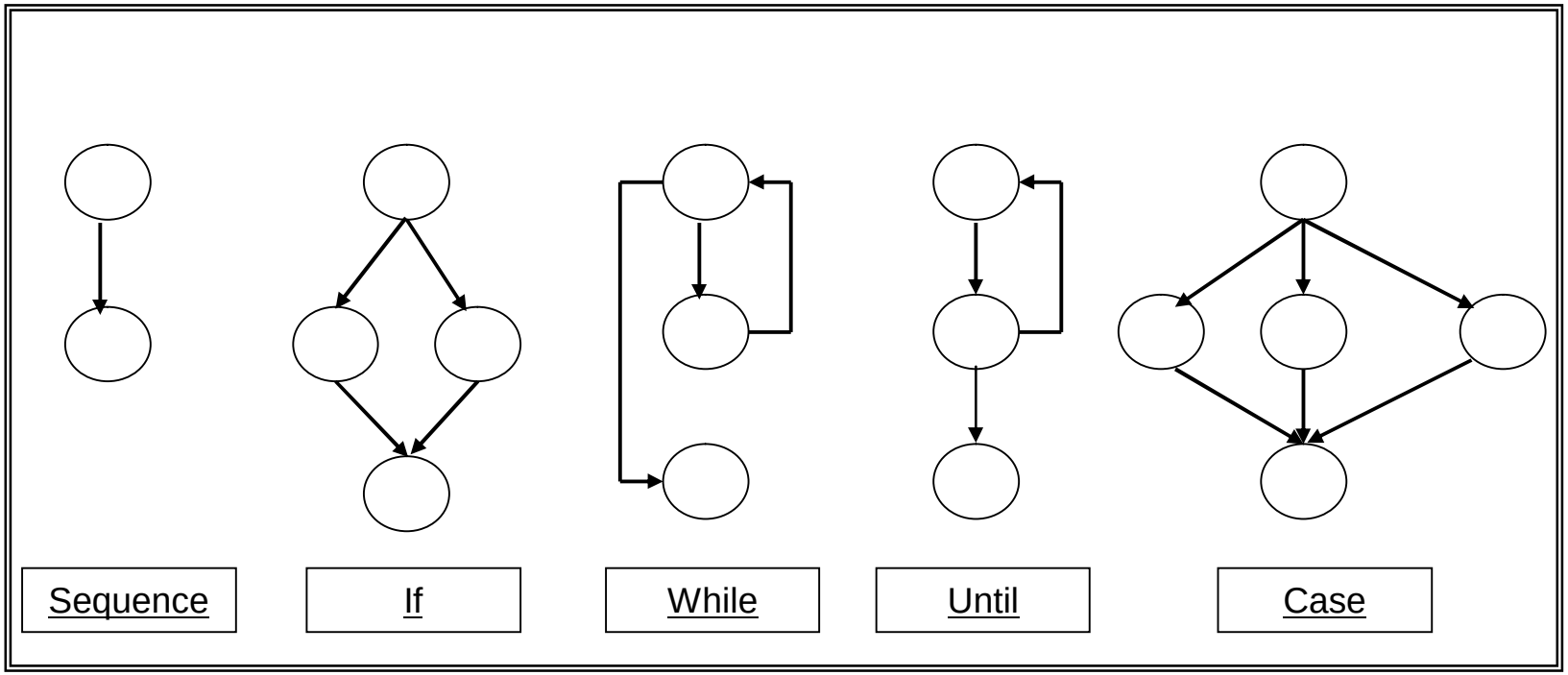
1. Cyclomatic Complexity calculations help the developer/tester to decide whether the module under test is overly complex or well written.
2. Recommended limit value of Cyclomatic Complexity is 10.
    1. >10

       Structure of the module is overly complex.
    2. >5 and <10

       Structure of the module is complex indicating that the logic is difficult to test.
    3. <5

       structure of the module is simple and logic is easy to test.

Flow Graphic Notation



Flow Graphic Notation

1. On a flow graph:
    1. Arrows called edges represent flow of control.
    2. Circles called nodes represent one or more actions.
    3. Areas bounded by edges and nodes are called regions.
    4. A predicate node is a node containing a condition.
    5. Any procedural design can be translated into a flow graph.
    6. Note that compound Boolean expressions at tests generate at least two predicate node and additional arcs.

Contd..2

Deriving Cyclomatic Complexity

1. Cyclomatic Complexity equals number of independent paths through standard control flow graph model.

2. Steps to arrive at Cyclomatic Complexity
    1. Draw a corresponding flow graph.
    2. Determine Cyclomatic Complexity.
    3. Determine independent paths.
    4. Prepare tests cases.

Graph Matrices

1. Can automate derivation of flow graph and determination of a set of basis paths.
2. Software tools to do this can use a graph matrix.
3. Graph matrix:
    1. Is square with # of sides equal to # of nodes.
    2. Rows and columns correspond to the nodes.
    3. Entries correspond to the edges.

Contd..2

Graph Matrices

1. Can associate a number with each edge entry.
2. Use a value of 1 to calculate the Cyclomatic complexity
    1. For each row, sum column values and subtract 1.
    2. Sum these totals and add 1.

Contd..2

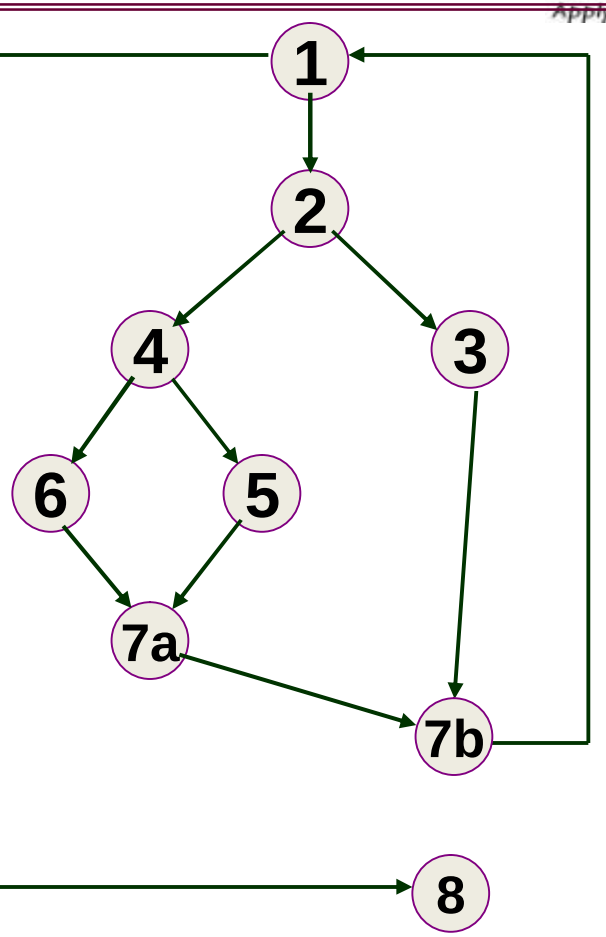
Some other interesting link weights

1. Probability that a link (edge) will be executed.
2. Processing time for traversal of a link.
3. Memory required during traversal of a link.
4. Resources required during traversal of a link.

Contd..2

Graph Matrices

	1	2	3	4	5	6	7a	7b	8
1		1							1
2			1	1					
3								1	
4					1	1			
5							1		
6							1		
7a								1	
7b	1								
8									



Black Box Testing

1. Equivalence Class / Partitioning
2. Boundary Value Analysis
3. Error guessing
4. State Transition Diagram
5. Cause Effect Graph

Equivalence Partitioning

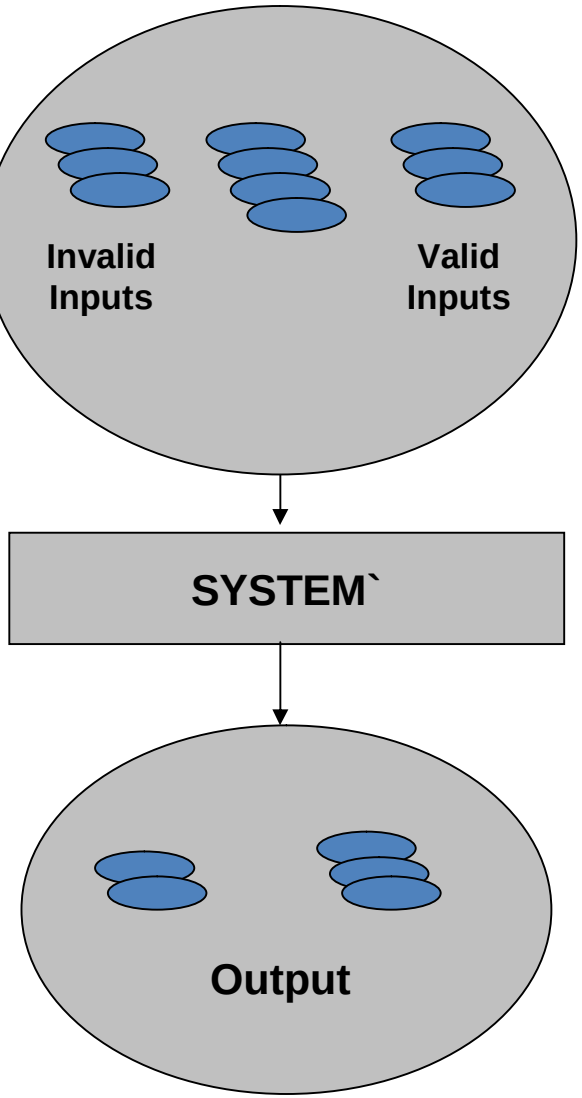
1. It is an organized way of designing and defining test cases.

2. Equivalence Partitioning is the process of dividing the input domain in to the different classes (Valid and Invalid), And for a valid input class make the equal partitions so that it will reduce the no of test cases.

3. Idea is to partition the input space into a small number of equivalence classes such that, according to the specification, every element of a given class is ‘‘handled’’ (i.e., mapped to an output)‘‘ in the same manner.’’

Equivalence Partitioning

4. Divides the input domain into classes of data for which test cases can be generated.
5. Attempts to uncover classes of errors.
6. Divides the input domain of a program into classes of data.
7. Derives test cases based on these partitions.
8. An equivalence class is a set of valid or invalid states of input.
9. Test case design is based on equivalence classes for an input domain.



Equivalence Partitioning

10. Assuming the program is implemented in such a way that being ‘‘handled in the same manner’’ means that either
    1. (a) every element of the class would be mapped to a correct output, (if a test case in an equivalence class does not detect an error, then any other test case in that class should also does not detect the error.)
       or
    2. (b) every element of the class would be mapped to an incorrect output (if a test case in an equivalence class detects an error, then any other test case in that class should also detect.)

Equivalence Partitioning

11. Two types of equivalence classes
    1. The set of valid inputs to the program is regarded as the “Valid equivalence class”
    2. All other inputs are included in the “Invalid equivalence class”

Example

An input condition specifies a range of values between 4 & 10

Valid Equivalence class: 4 – 10

Invalid Equivalence classes:

Less than 4

Greater than 10

Equivalence Partitioning

Example

An input condition specifies a set of values (A cloth can be Red, Green, Yellow, Violet)

Valid Equivalence class:

Red, Green Yellow, Violet (All valid values)

Invalid Equivalence classes:

Purple, Brown (Invalid values)

Equivalence Partitioning

Example

"The input condition is a "must be" situation. Like, the input string must be upper-case"

Valid Equivalence class:

Uppercase characters

Invalid Equivalence classes:

All other input except uppercase characters

Equivalence Partitioning

Example

The application is expected to display the class the student has obtained after taking the input of the no. of percentage scored.

The details of assigning the class specified below

0-34 – Fail

35 – 50 - Pass

51 – 59 - Second class

60 – 69 - First class

70 – 100 - Distinction

Equivalence Partitioning

Apply Equivalence Partitioning first to arrive at a broader set

0 – 34 : -1, 0, 17, 34, 35

35 – 50 : 34, 35, 40, 45 50, 51

51 – 59 : 50, 51, 55, 59, 60

60 – 69 : 59, 60, 63, 66, 69, 70

70 – 100 : 69, 70, 85, 100, 101

Totally, 27 test cases

Equivalence Partitioning

Eliminate redundant values obtained during EP

{ -1,0, 17, 33, 34, 35, 40, 45, 50, 51, 55, 59, 60, 63, 66, 69, 70, 85, 100, 101 }

Totally, 20 Test Cases

Equivalence Partitioning optimizes Test cases

Boundary Value Analysis

1. Complements to Equivalence partition
2. BVA leads to a selection of test cases that exercise bounding values
3. Design test cases test
    1. Min values of an input
    2. Max values of an input
    3. Just above and below input range
4. Helps to write test cases that exercise bounding values.
5. Complements Equivalence Partitioning.
6. Guidelines are similar to Equivalence Partitioning.
7. Two types of BVA:
    1. Range
        1. Above and below Range
    2. Value
        1. Above and below min and max number

Boundary Value Analysis

1. Many times validations fail for boundary values for a field
2. Programmers tend to overlook this aspect.
3. Developers have tendency to test only middle value and overlook boundaries.
4. Middle values are called comfortable values.
5. Example
    1. Month Range 1-12
        1. Boundary conditions will be 1 and 12

Boundary Value Analysis

1. When examining specifications to identify equivalence classes, attention should be given to values which lie on the boundary of specified ranges.

2. BOUNDARY VALUE ANALYSIS involves selecting test data from equivalence classes which satisfy boundary conditions.

3. In addition, test data should be selected to meet any boundary conditions which the specifications place on the output.

4. Boundary conditions are those situations directly on, above or beneath the edges of input equivalence classes and output equivalence classes.

Boundary Value Analysis

1. Requirement

   "The output of the component A is provided to component B for further processing. Component B accepts the output of component A only if the output is 5. Test component B for boundaries"

2. BVA for single value
   1. Applicable formula: (n-1), n, (n+1)
   2. Test Values: 4,5,6

Boundary Value Analysis

Partition system inputs and outputs into equivalence sets

If input is a 5-digit integer between 10,000 and 99,999, equivalence partitions are

<10,000,

10,000-99, 999 and

>=100000

Choose test cases at the boundary of these sets

9999, 10000, 99999, 100000

Boundary Value Analysis

Requirement

"The no. of tickets that can be booked at a time is 10. Test whether the program accepts proper input values"

Range: 1 to 10

Formula: (m-1), m, n, (n+1)

Test Values: 0,1,10,11

Case Study

Requirements :

An employee in an organization can enter transactions only in working hours. When an employee selects ‘transaction menu’ , the system checks the ‘system time’ and performs the following checks :

1. Transactions are not allowed on Saturday and Sunday.
2. There is a list of holidays. Transactions are not allowed on holidays.
3. For clerks, the ‘transaction hours’ are 9.00 to 13.00.
4. For officers, the ‘transaction hours’ are 9.00 to 15.00.
5. Anybody else is not allowed access to transaction menu.

Identify Equivalent Classes.

Identify boundary values for each class.

Error guessing

1. An *ad hoc* approach, based on intuition and experience

2. Identify tests that are likely to expose errors

3. Make a ***list*** of possible errors or error prone situations and then develop tests based on the list

State Transition Testing

1. Object = **state** + **behavior**

2. Behavior is the **sequence** of messages (or events) that an object accepts

3. State machines model **life cycle** of an object, i.e. its dynamic behavior

4. We also talk about **sequential** systems
   (how different from a decision table?)

Key Concepts

1. **State**: a condition in which a system is waiting for one or multiple events

2. **Transition**: represents change from one state to another caused by an event

3. **Event**: input that may cause a transition

4. **Action**: operation initiated because of a state change (occur on transitions)

Cause Effect Graph

1. Functional or specification-based technique.
2. Systematic approach to selecting a set of high-yield test cases that explore ***combinations*** of input conditions
3. Rigorous method for transforming a natural-language spec into a formal-language spec
4. Exposes incompleteness and ambiguities in the specification.
5. All possible *causes* are identified: inputs, stimuli, anything that will elicit a response from the system
6. All possible *effects* are identified: outputs, changes in the system state

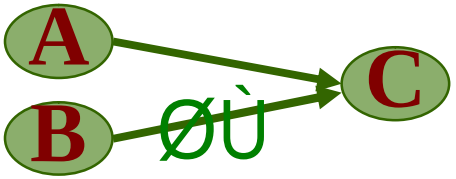
Cause Effect Graph

1. graphs are combined using operators:
    1. *not*
    2. *and*
    3. *or*
    4. *nor*

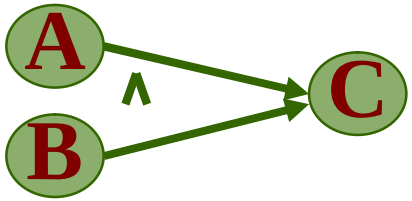
Type of processing node	**description**
AND	Effect occurs if all input are true (1)
OR	Effect occurs if both or one input is true
XOR	Effect occurs if one input is true
Negation (Not)	Effect occurs if input are false (0)

Drawing Cause-Effect Graphs

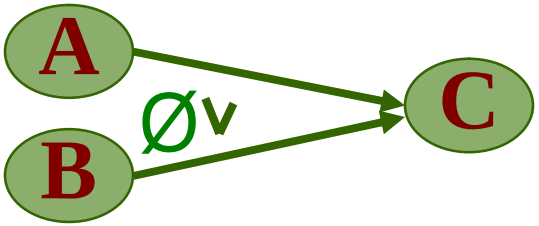
If A then B



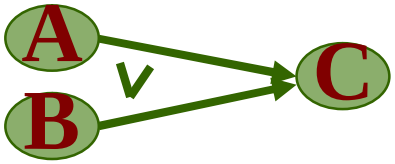
If (not(A and B)) then C



If (A and B)then C



If (not (A or B))then C



If (A or B) then C

If (not A) then B

Case Study

Requirements :

An employee in an organization can enter transactions only in working hours. When an employee selects 'transaction menu' , the system checks the 'system time' and performs the following checks :

1. Transactions are not allowed on Saturday and Sunday.
2. There is a list of holidays. Transactions are not allowed on holidays.
3. For clerks, the 'transaction hours' are 9.00 to 13.00.
4. For officers, the 'transaction hours' are 9.00 to 15.00.
5. Anybody else is not allowed access to transaction menu.

Identify Equivalent Classes.

Identify boundary values for each class.

Consider different values for employee code. Find out the designation from employee master.

Case Study

1. The video store offers discount schemes in order to attract the customers

2. The customer receives 10% discount on any CD for the first year.

3. There after if the customer decides to go for 2 year subscription, discount will be 13%, for 3 years, it will be 15%, 5 years it will be 20%

4. If the CD is of type “Cartoon” the customer will get 5% discount, if it of type “Horror” then the discount will be 2%

Building Test Data

1. Unit Testing – Generated Data
2. Integration Testing – Generated / Client Supplied Data
3. System Testing - Generated / Client Supplied Data
4. Acceptance Testing (Alpha & Beta Testing) – Client Supplied Data
5. Structural Testing –
    1. Test data derived from the knowledge of the structure of the system
    2. Example –
        1. Amount of load
        2. Number of concurrent user creation
        3. Number of records to be inserted at a time

Building Test Data

1. Functional Testing –
    1. Data derived by considering the requirements / specifications
    2. Examples –
        1. Basic pay, overtime pay, bonus of an employee in payroll system
        2. Date of reservation, train no., starting point and destination in case of Railway reservation system
2. Usability Testing –
    1. Checks the 'Fit For Use' features of the system.
    2. Example –
        1. Number of fields, color of the screen, navigation, alignment of the field values, font used, consistency in design (comparing the dimensions of the window etc. with the defined standard etc.)

Building Test Data

1. Tools used to build test data –
    1. Data Dictionary
    2. Test Data Generator

Chapter-7

Test Execution

Test Execution

Topics Covered in this lesson are:

1. Objectives
2. Execution Considerations
3. Test Execution Activities
4. Executing Tests
5. Concerns

Test Execution Objectives

1. Definition

   The processing of a test case by the software under test, producing an outcome.

2. Purpose
   1. To execute the appropriate collections of tests required to evaluate product quality

   2. To capture test results that facilitate ongoing assessment of the product

Execution Considerations

1. Execute tests which are deemed highest risks first.
2. Tests on which there are many dependent tests will be executed first
3. Test cases central to the architecture
4. Test execution on number of Operating system, browsers, servers etc.
5. Execution Manual or Automated?

Execution Activities

1. Test platforms
2. Test cycle strategy
3. Executing the Unit Test
4. Executing the Integration Test
5. Executing the System Test

Execution Activities

1. Set up test environment

    1. Hardware
    2. Software
    3. Tools
    4. Data

2. Test Cycle Strategy
    1. Number of test cycles

Execute Unit Tests

1. Execute using tools or manually
    1. Execute tests individually
    2. Execute a suite

2. Executed by Developer

3. Defects are not reported in Defect Tracking tool

4. Entrance Criteria
    1. The unit is compiled successfully
    2. Code has gone through Code inspection process
    3. Unit Test Environment is ready

Execute Integration Tests

1. The most effective method for validating successful integration is to:

    1. Test the client components
    2. Test the server components
    3. Test the network
    4. Integrate the client, server, and network

2. Entrance Criteria

    1. Unit testing of the components is complete
    2. No open defect exists in the unit

Execute System Tests

1. Set up system test environment, mirroring the planned production environment as closely as possible.

2. Establish the test bed.

3. Identify test cycles needed to replicate production where batch processing is involved.

4. Assign test cases to test cycles

5. Execute the tests

Review - When to Stop Testing

1. Test Manager will consider the following factors
2. Deadlines, e.g. release deadlines, testing deadlines;
3. Test cases completed with certain percentage passed;
4. Test budget has been depleted;
5. Coverage of code, functionality, or requirements reaches a specified point;
6. Bug rate falls below a certain level; or
7. Beta or alpha testing period ends.

Concerns in Test Execution

1. Software is not in a testable mode for this test level.
2. There is inadequate time and resources.
3. Significant problems will not be uncovered during testing.

Chapter-8

Defect Management

Defect Management

Topics Covered in this lesson are:

1. What is a defect
2. Defect Life Cycle
3. Defect Management Process

Review - What is defect?

1. The software doesn't do something that the product specification says it should do.

2. The software does something that the product specifications says it shouldn't do.

3. The software does something that the product specification doesn't mention.

4. The software doesn't do something that the product specification doesn't mention but should do.

5. Difficult to understand, hard to use, slow etc.

Examples of Defects

1. User gives wrong / incomplete requirements
2. Analyst interprets requirement incorrectly
3. Requirements are not recorded correctly
4. Incorrect design specs
5. Incorrect program specs
6. Errors in coding
7. Data entry errors
8. Errors in testing: falsely detect an error / fail to detect existing errors
9. Mistakes in error correction

Software Bug

1. A fault in a program which causes the program to perform in an un-intended manner.

2. Software bug occurs when

    1. The software does not do something that specification says it should do.

    2. The software does something that specification says not to do

Error

1. The stage of system from where further action of system will lead it to failure.

2. A human action that produces incorrect step, process, result or data definition.

3. Mistake made by developer.

    1. Typographical error.

    2. Misleading of specs.

    3. A misunderstanding of what module should do.

Error

4. It may be

    1. Actual bugs in the code.
    2. Calculation errors.
    3. Errors in handling/interpreting data.
    4. User does not observe operation described in manuals.
    5. Testing errors.

Fault…

1. A fault occurs when a human error results in a mistake in some software product.

2. Difference between incorrect program and the correct versions.

3. A single error can lead to one or more faults.

4. One fault can result in changes to one product or multiple changes to multiple products

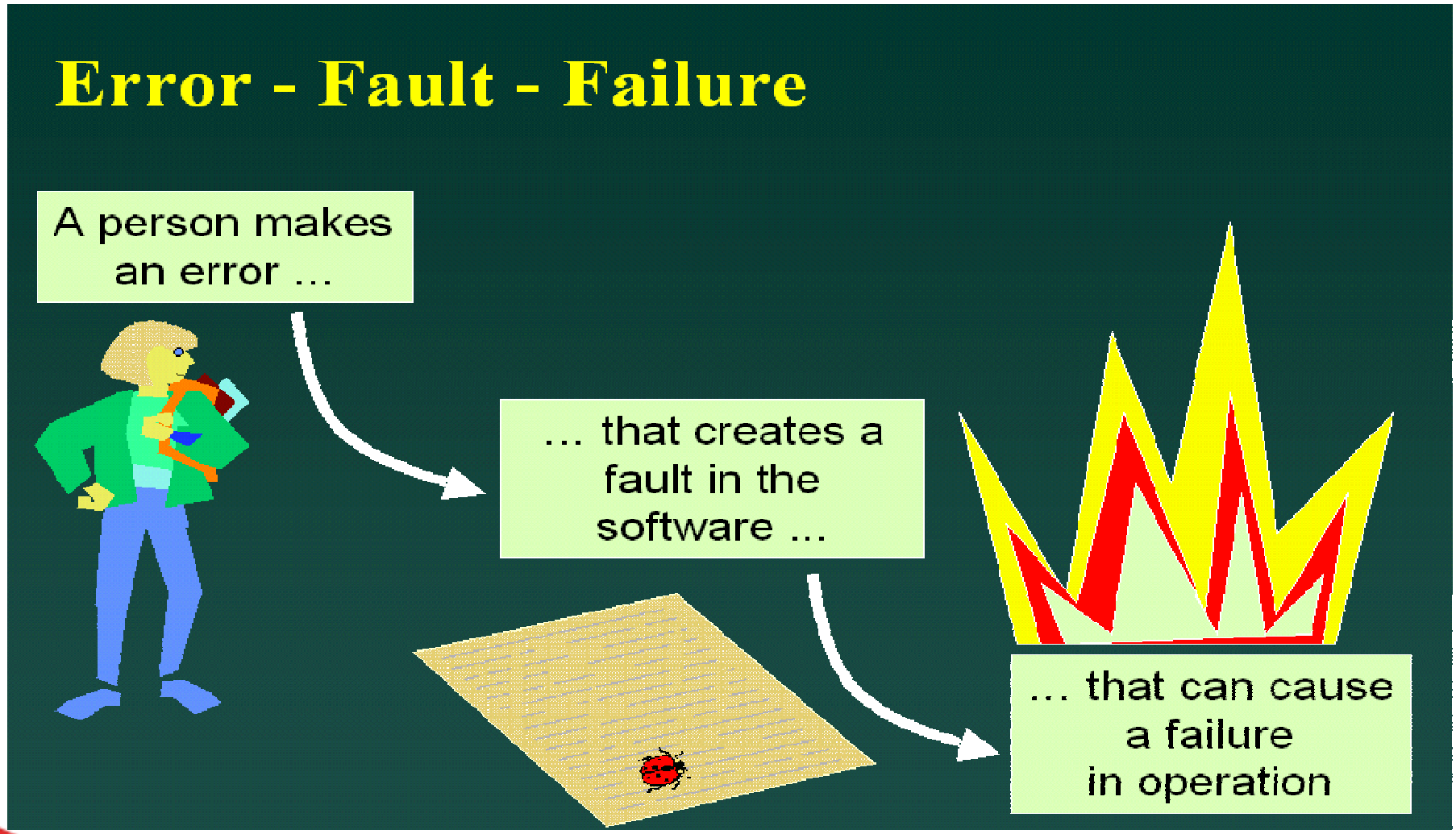
Failure…

1. A fault can go undetected until a failure occurs, which is when a user or tester perceives that the system is not delivering the expected service.

2. Deviation of the system from its expected behavior.

3. Inability of a system to perform its required functions within specified performance requirements.

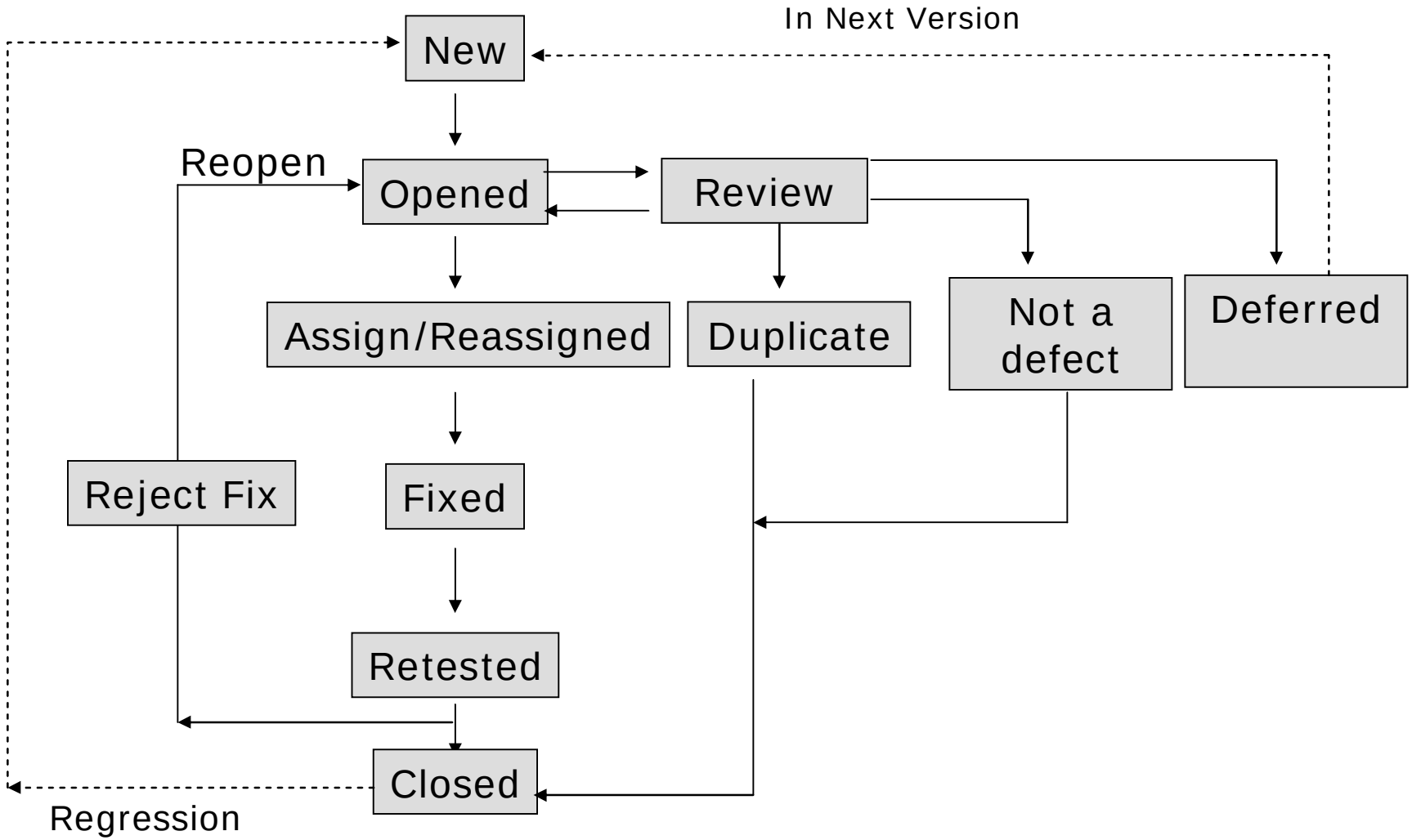
Error - Fault - Failure



Defect Management

1. Overview of defect management process
2. Primary goal is to prevent defects
3. Should be integral part of development process
4. As much as possible should be automated
5. Defect information should be used for process improvements
6. To prevent defects processes must be altered

Defect Life Cycle



How to decide severity?

Severity	Description	Criteria
1	Very High / Show Stopper	Core dumps, Inability to install/uninstall the product, Product will not start, Product hangs or Operating System freezes, No workaround is available, Data corruption, Product abnormally terminates
2	High	Workaround is available, Function is not working according to specifications, Severe performance degradation, Critical to a customer
3	Medium	Incorrect error messages, Incorrect data, Noticeable performance inefficiencies
4	Low	Misspellings, Grammatical errors, Enhancement requests. Cosmetic flaws

How to decide priority?

Priority	Description	Criteria
1	Very High	Immediate fix, block further testing, very visible
2	High	Must fix before the product is released
3	Medium	Should fix if time permits
4	Low	Would like fix but can be released as it is.

Defect Report Attributes

1. Defect Id
2. Project Name
3. Module Name
4. Sub-module Name
5. Phase
6. Type
7. Severity
8. Priority
9. Summary
10. Description
11. Status
12. Reported by/on
13. Assigned to
14. Cc to

Defect Reporting

1. To have a complete record of discrepancies that may be used in multiple ways

2. Forms basis for quality measurement

3. Major purpose
    1. Correct the defect
    2. Report status of application
    3. Gather statistics to predict defects in future applications
    4. Process improvement

Defect Tracking Tools

1. Test Director
2. Rational Clear Quest
3. Bugzilla
4. Excel Sheet

Case Study 1

The following are the defects

1. The month list does not allow store manager to select the month for producing top ten list report

2. The reason for blocking a customer is not mandatory

3. The overdue alert is missing

4. The report of rental pattern is not formatted properly in terms of alignment.

5. The overdue calculated is up to 3 decimal places

Case Study 1

1. After adding movies to stock beyond 20, the stock quantity remains unchanged.
2. System removes users form the users list when deleted, but form the database.
3. The report of rental patterns should be displayed in the form of line graph.
4. System crashes when order is placed on movie from other office.
5. Back up does not provides facility to choose another location, uses default.
6. **You have to generate a defect report as per the given format.**

Case Study 2

Causes and cost of defect

Requirements ambiguous, not defined	32	8000 $
Design incorrect	13	5000 $
Coding errors	45	4000 $
Miscellaneous	10	2500 $

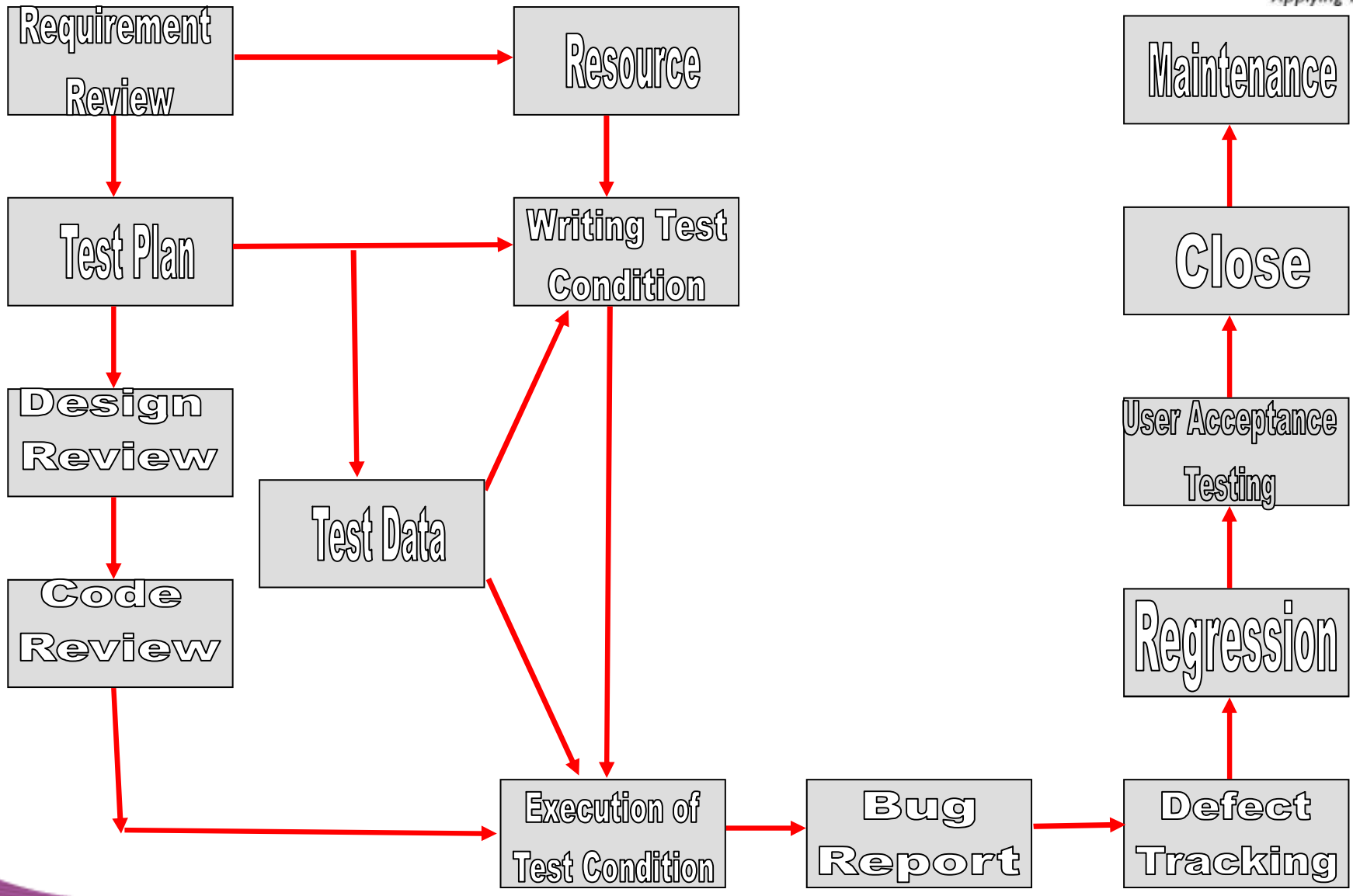
Case Study -2

1. Number of defects with following severity levels
    1. Critical – 12
    2. High – 25
    3. Medium – 22
    4. Low - 41
2. The Number of defects identified at each stage
    1. Verification - 10
    2. Validation – 100
    3. Customer site – 15
3. The development team resolved all critical and high severity defects before delivery.
4. **You are expected to analyze and report findings**

Chapter-9

Software Testing Life Cycle

Software Testing Life Cycle



Software Testing Life Cycle- Phases

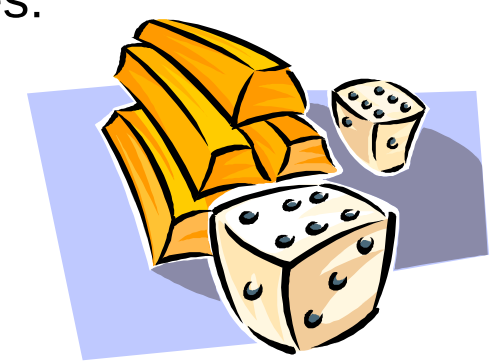
1. Requirement Analysis
2. Prepare Test Plan
3. Test Case Designing
4. Design Review
5. Code Review
6. Test Case Execution
7. Bug Reporting, Analysis and Regression testing
8. Inspection and release
9. Client acceptance and support during acceptance
10. Test Summary analysis

Requirement Analysis

1. Objective

   The objective of Requirement Analysis is to ensure software quality by eradicating errors as earlier as possible in the developement process, as the errors noticed at the end of the software life cycle are more costly compared to that of early ones, and there by validating each of the Outputs.

2. The objective can be acheived by three basic issues:
   1. Correctness
   2. Completeness
   3. Consistency



Prepare Test Plan- Activities

1. Scope Analysis of project
2. Document product purpose/definition
3. Prepare product requirement document
4. Develop risk assessment criteria
5. Identify acceptance criteria
6. Document Testing Strategies.
7. Define problem - reporting procedures
8. Prepare Master Test Plan

Design-Activities

1. Setup test environment

2. Design Test Cases: Requirements-based and Code-based Test Cases

3. Analyze if automation of any test cases is needed

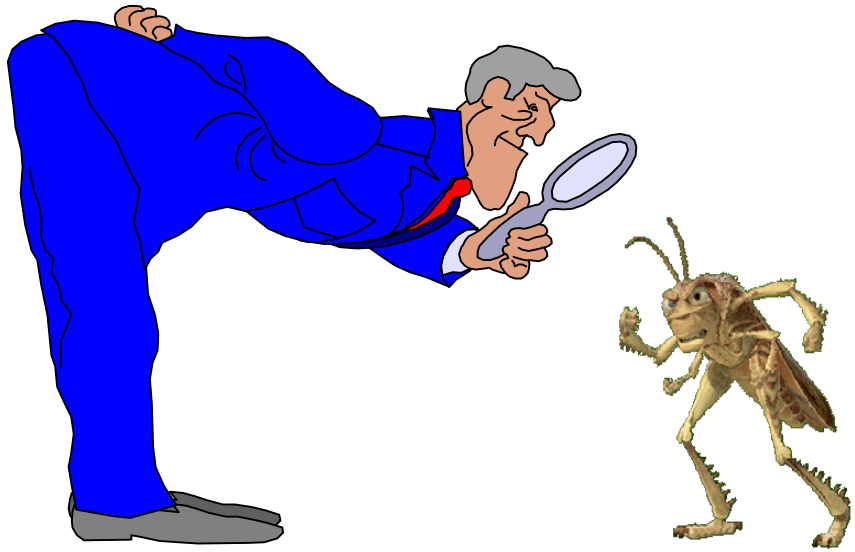
Execution- Activities

1. Initial Testing, Detect and log Bugs
2. Retesting after bug fixes
3. Final Testing
4. Implementation
5. Setup database to track components of the automated testing system, i.e. reusable modules

Bug Reporting, Analysis, and Regressing Testing

1. Activities

1. Detect Bugs by executing test cases
2. Bug Reporting
3. Analyze the Error/Defect/Bug
4. Debugging the system
5. Regression testing



Inspection and Release-Activities

1. Maintaining configuration of related work products
2. Final Review of Testing
3. Metrics to measure improvement
4. Replication of Product
5. Product Delivery Records
6. Evaluate Test Effectiveness

Client Acceptance

1. Software Installation
2. Provide Support during Acceptance Testing
3. Analyze and Address the Error/Defect/Bug
4. Track Changes and Maintenance
5. Final Testing and Implementation
6. Submission, client Sign-off
7. Update respective Process

Support during Acceptance-Activities

1. Pre-Acceptance Test Support
2. Installing the software on the client's environment
3. Providing training for using the software or maintaining the software
4. Providing hot-fixes as and when required to make testing activity to continue
5. Post Acceptance Test Support
6. Bug Fixing

Test Summary Analysis- Requirement

1. Quantitative measurement and Analysis of Test Summary
2. Evaluate Test Effectiveness
3. Test Reporting
    1. Report Faults – (off-site testing)
    2. Report Faults – (on-site/ field testing)

Chapter-10

GUI Testing

Windows Compliance Standards

1. Windows resize options

   Maximize, minimize and close options should be available.

2. Using TAB

   Should move the focus (cursor) from left to right and top to bottom in the window.

3. Using SHIFT+TAB

   Should move the focus (cursor) from right to left and bottom to top.

4. Text

   Should be left-justified.

Windows Compliance Standards (Contd..)

1. Edit Box
    1. U should be able to enter data.
    2. Try to overflow the text, text should be stopped after the specified length of characters.
    3. Try entering invalid characters - should not allow.

2. Radio Buttons
    1. Left and right arrows should move ‘ON’ selection. So should UP and DOWN.
    2. Select with the mouse by clicking.

3. Check Boxes
    1. Clicking with the mouse on the box or on the text should SET/UNSET the box.
    2. Space should do the same.

Windows Compliance Standards (Contd..)

4. Command Buttons
    1. Should have shortcut keys (except OK and Cancel buttons).
    2. Click each button with the mouse - should activate.
    3. TAB to each button & press Space/Enter - should activate.

5. Drop Down List
    1. Pressing the arrow should give list of options.
    2. Pressing a letter should bring you to the first item in the list with that start letter.
    3. Pressing Ctrl+F4 should open/drop down the list box.

Windows Compliance Standards (Contd..)

6. Combo Boxes
    1. Should allow text to be entered.
    2. Clicking the arrow should allow user to choose from the list

7. List Boxes
    1. Should allow a single selection to be chosen by clicking with the mouse or using the Up and Down arrows.
    2. Pressing a letter should bring you to the first item in the list with that start letter.

Screen Validation Standards

1. Aesthetic Conditions
    1. The general screen background should be of correct color (company standards,….).
    2. The field prompts and backgrounds should be of correct color.
    3. The text in all the fields should be of the same font.
    4. All the field prompts, group boxes and edit boxes should be aligned perfectly.
    5. Micro help should be available and spelt correctly.
    6. All dialog boxes and windows should have a consistent look and feel.

Screen Validation Standards (Contd..)

2. Validation Conditions
    1. Failure of validation on every field should cause a user error message.
    2. If any fields are having multiple validation rules, all should be applied.
    3. If the user enters an invalid value and clicks on the OK button, the invalid entry should be identified and highlighted.
    4. In the numeric fields, negative numbers should be allowed to enter.
    5. Should allow the minimum, maximum and mid range values in numeric fields.
    6. All mandatory fields should require user input.

Screen Validation Standards (Contd..)

3. Navigation Conditions
    1. The screen should be accessible correctly from the menu and toolbar.
    2. All screens accessible through buttons on this screen should be accessed correctly.
    3. The user should not be prevented from accessing other functions when this screen is active.
    4. Should not allow to open number of instances of the same screen at the same time.

Screen Validation Standards (Contd..)

4. Usability Conditions
    1. All the dropdowns should be sorted alphabetically (unless specified).
    2. All pushbuttons should have appropriate shortcut keys and should work properly.
    3. All read-only and disabled fields should be avoided in the TAB sequence.
    4. Should not allow to edit micro help text.
    5. The cursor should be positioned in the first input field or control when opened.
    6. When an error message occurs, the focus should return to the field in error after canceling it.
    7. Alt + Tab should not have any impact on the screen upon return.

Screen Validation Standards (Contd..)

5. Data Integrity Conditions
    1. The data should be saved when the window is closed by double clicking on the close box.
    2. There characters should not be truncated.
    3. Maximum and minimum field values for numeric fields should be verified.
    4. Negative values should be stored and accessed from the database correctly.

Screen Validation Standards (Contd..)

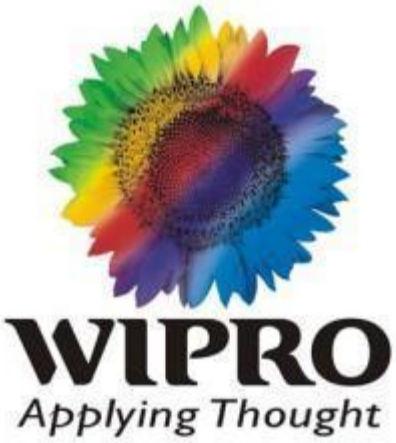
6. Modes (Editable, Read-only) conditions
    1. The screen and field colors should be adjusted correctly for read-only mode.
    2. Is the read only field necessary for this screen?
    3. All fields and controls should be disabled in read-only mode.
    4. No validation is performed in read-only mode.

Screen Validation Standards (Contd..)

7. General Conditions
    1. “Help” menu should exist.
    2. All buttons on all tool bars should have corresponding key commands.
    3. Abbreviations should not be used in drop down lists.
    4. Duplicate hot keys/shortcut keys should not exist.
    5. Escape key and cancel button should cancel (close) the application.
    6. OK and Cancel buttons should be grouped separately.
    7. Command button names should not be abbreviations.

Screen Validation Standards (Contd..)

7. General Conditions (Contd..)
    8. Field labels/names should not be technical labels, they should be meaningful to system users.
    9. All command buttons should be of similar size, shape, font and font size.
    10. Option boxes, option buttons and command buttons should be logically grouped.
    11. Mouse action should be consistent through out the screen.
    12. Red color should not be used to highlight active objects (many individuals are red-green color blind).
    13. Screen/Window should not have cluttered appearance.
    14. Alt+F4 should close the window/application.



Thank You

Yuvraj Boldhan

Senior System Engineer

yuvraj.boldhan@wipro.com