UNIT - IV

Software Testing Methodology

Defects hard to find

Two reasons defects go undetected

1.Not looking:

- Tests often are not performed because a particular test condition was unknown.
- Some parts of a system go untested because developers assume that software changes don't affect them.

2. Looking, but not seeing:

- This is like losing your car keys ,only to discover they were in plain sight the entire time.
- Sometimes developers become so familiar with their system that they overlook details, which is why independent verification and validation is used to provide a fresh view point.

IT improperly interprets requirements

Information technology (IT) staff misinterpret what the user wants, but correctly implement what the IT people believe is wanted.

The users specify the wrong requirements

The specifications given to IT are erroneous.

- The requirements are incorrectly recorded.
- The design specifications are incorrect.
- The program specifications are incorrect.
- There are errors in the program coding
- There are data entry errors.
- There are mistakes in error corrections.
- The corrected condition causes another defect.

Verification

Verification :- A Tester uses verification methods to ensure the system (software , hardware, documentation, and personnel) complies with the organization's standards and processes relying on review or non-executable methods.

Computer System verification example

Requirements review:-

Performed by → Developers, Users

Explanation →The study and discussion of the computer system requirements to ensure they meet stated user needs and are feasible.

Deliverable → reviewed statement of requirements, ready to be translated into system design.

Design review

Performed by → Developers

Explanation → The study and discussion of the computer system design to ensure it will support the system requirements.

Deliverable → System design, ready to be translated into computer programs, hardware configurations documentation , and training

Code walkthrough

Performed by → Developers

Explanation → An informal analysis of the program source code to find defects and verify coding techniques.

Deliverable → Computer software ready for testing or more detailed inspections by the developer

Code Inspection

Performed by → Developers

Explanation → A formal analysis of the program source code to find defects as defined by meeting computer system design specifications. Usually performed by a team composed of developers and subject matter experts.

Deliverable → Computer software ready for testing by the developer

Validation

Validation:- Validation physically ensures that the system operates according to the plan by executing the system functions through a series of tests that can be observed and evaluated.

Computer System Validation example

Unit testing

Performed by → Developers

Explanation → The testing of a single program , module ,or unit of code. Usually performed by the developer of the unit. Validates that the S/W performs as designed.

Deliverable → S/W unit ready for testing with other system component, such as other S/W units, hardware, documentation, or users

Integrated Testing

Performed by → Developers

Explanation → The testing of related programs, modules ,or units of code. Validates that multiple parts of the system interact according to the system design.

Deliverable → Portions of the system ready for testing with other portions of the system

System Testing

Performed by → Developers, users

Explanation → The testing of an entire computer system. This kind of testing can include functional and structural testing, such as stress testing.
It Validates the system requirements.

Deliverable → A tested computer system , based on what was specified to be developed or purchased .

User acceptance testing

Performed by → Users

Explanation → The testing of a computer system or parts of a computer system to make sure it will work in the system regardless of what the system requirements indicated.

Deliverable → A tested computer system, based on user needs

Functional and Structural Testing

- **Functional Testing:**

- Treats a program as a black box. Outputs are verified for conformance to specifications from user's point of view.
- It never bother about program's implementation details
- FT takes user's point of view
- Functional testing is some times called as black box testing, no need to know about the coding of the program.
- Functional Testing usually describes 'what' the system does.
- Functions are tested by feeding them input and examining the output

- Some examples in this category include:

- Decision tables, equivalence partitioning, range testing, boundary value testing, database integrity testing, cause effect graphing, orthogonal array testing, array and table testing, exception testing, limit testing, and random testing.

Functional Testing

- Functional testing typically involves five steps:
  - The identification of functions that the software is expected to perform
  - The creation of input data based on the function's specifications
  - The determination of output based on the function's specifications
  - The execution of the test case
  - The comparison of actual and expected outputs
- Advantages of Functional Testing:
  - Simulate actual system usage
- Disadvantages of Functional Testing:
  - Includes the potential to miss logical errors in software

Structural Testing

- Looks at the implementation details: programming style, control method, source language, database & coding details.

- Structural testing is some times called as white box testing because knowledge of code is very much essential.

- Structural testing is often referred to as ‘white box’ or ‘glass box’ or ‘clear-box testing’

- structural testing we are interested in what is happening ‘inside the system/application’.

- For example, a structural technique wants to know how loops in the software are working. Different test cases may be derived to exercise the loop once, twice, and many times.

Structural Testing

Various Structural Testing are

- Stress Testing
- Execution Testing
- Operations Testing
- Recovery Testing
- Path Testing

- Advantages of Structural Testing:
  - Enables you to test the software logic
  - Enables you to test structural attributes, such as efficiency of code
- Disadvantages of Structural Testing:
  - Does not ensure that you've met user requirements that is, it focuses only on the internal logic and does not verify the logic to the specification.
  - Another disadvantage is that there is no way to detect missing paths and data-sensitive errors.
  - For example,if the statement in a program should be coded "if |a–b| < 10" but is coded "if (a–b) < 1," this would not be detectable without specification details.

Workbench concept

- In information technology workbenches are more frequently referred to as phases, steps ,or tasks.
- The workbench is a way of illustrating and documenting how a specific activity is to be performed.
- Defining workbenches is normally the responsibility of a process management committee.

Four components of workbench

Input : The entrance criteria or deliverables needed to perform work.

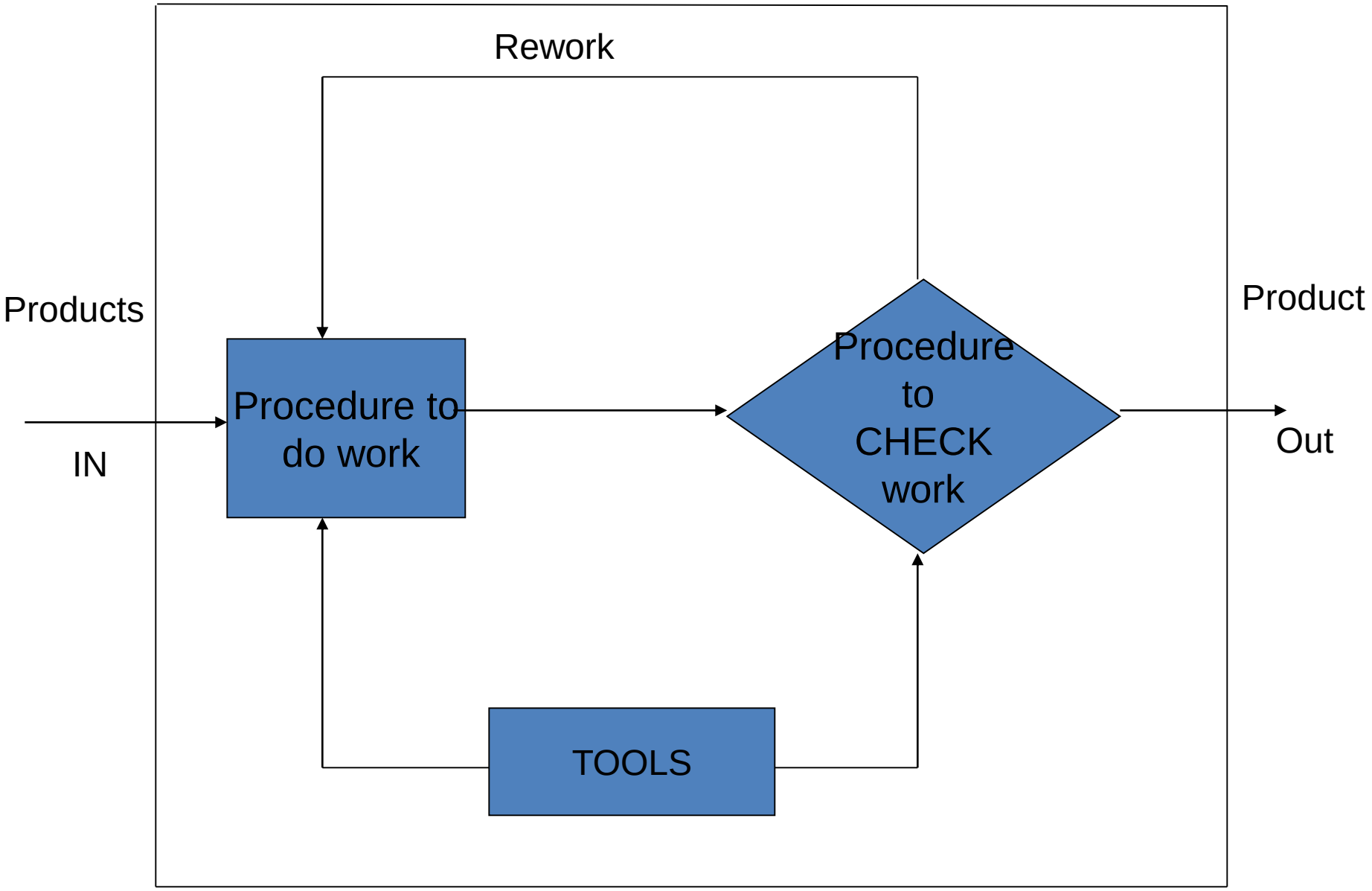
Procedures to do: The work tasks or processes that will transform the input into the output.

Procedures to check: The processes that determine that the output meets the standard.

Output: The exit criteria or deliverables produced from the workbench.

The programmer's workbench consists the following steps

- Input *products* (program specs) are given to the producer (programmer).
- Work is *performed* (e.g., coding / debugging); a procedure is followed; a product or interim deliverable (e.g., a program/module/unit) is produced.
- Work is *checked* to ensure product meets specs and standards ,and that the procedure was followed.
- If check finds no problems, product is released to the next workbench.
- If check finds problem, product is sent back for rework


Rework
Products
Product
Procedure to do work
Procedure to CHECK work
IN
Out
TOOLS

Eight Considerations in Developing Testing Methodology

1. Acquire and study the Test Strategy.
2. Determine the type of development project.
3. Determine the type of software system.
4. Determine the project scope.
5. Identify the tactical risks.
6. Determine when testing should occur.
7. Build the system test plan.
8. Build the unit test plan.

1. Acquire and study the Test Strategy

The test team needs to acquire and study the test strategy, focusing on the following questions:

- What is the relationship of importance among the test factors?
- Which of the high-level risks are the most significant?
- Who has the best understanding of the impact of the identified business risks?
- What damage can be done to the business if the software fails to perform correctly?
- What damage can be done to the business if the software is not completed on time?

2. Determine the type of development project

- The type of project refers to the environment/methodology in which the software will be developed.
- As the environment changes , so does the risk of testing.

3. Determine the type of software system

- The type of software system refers to the processing that will be performed by that system.
- This step contains sixteen different software system types.

- **Batch(general).**Can be run as a normal batch job and makes no unusual hardware or input-output actions
- **Event control.** Does real-time processing of data resulting from external event.
- **Process control.**Receives data from an external source and issues commands to that source to control its actions based on the received data.
- **Procedure Control.** Controls other software ; for example , an operating system that controls execution of time-shared and batch computer programs

- **Advanced mathematical models.** Resembles simulation and business strategy software, but has the additional complexity of heavy use of mathematics.
- **Message processing.** Handles input and output messages, processing the text, or information contained therein.
- **Diagnostic software.** Detects and isolates hardware errors in the computer where it resides, or in other hardware that can communicate with that computer

- **Sensor and signal processing.** Similar to that of message processing, but requires greater processing to analyze and transform the input into a usable data processing format
- **Simulation.** Simulates an environment, mission situation ,other hardware; inputs from these to enable a ore realistic evaluation of a computer program or a piece of hardware.

- **Database management.** Manages the storage and access of (typically large) groups of data.
- **Data acquisition.** Receives information in real time and stores it in some form suitable for later processing ;
- **Data presentation.** Formats and transforms data , as necessary , for convenient and understandable displays for humans.
- **Decision and planning aids**. Uses artificial intelligence techniques to provide an expert system to evaluate data and provide additional information and consideration for decision and policy makers.

- **Pattern and image processing.** Generates and processes computer images. Such software may analyze terrain data and generate images based on stored data.
- **Computer system software.**Provides services to operational computer programs.
- **Software development tools.** Provides services to aid in the development of software.

4. Determine the project scope

- The project scope refers to the totality of activities to be incorporated into the software system being tested
- The scope of the testing effort is usually defined by the scope of the project.
- New system development has a much different scope from modifications to an existing system.
- When defining the scope, consider the following characteristics and then expand the list to encompass the requirements of the specific software system being tested.

New Systems Development

- Automating manual business process?
- Which business processes will or won't be affected?
- Which business areas will or won't be affected?
- Interfacing to existing systems?
- Existing systems will or won't be affected?

Changes to Existing Systems

- Corrective only?
- Maintenance re-engineering standards?
- Correction to known latent defects in addition to enhancements?
- Other systems affected?
- Risk of regression?

5. Identify the tactical risks

- Tactical risks are divided into three categories:

1) Structural Risks

These risks are associated with the application and the methods used to build it.

2) Technical Risks

These risks are associated with the technology used to build and operate the application.

3) Size risks

These risks are associated with the magnitude in all aspects of the software.

6. Determine when testing should occur

- Requirements Phase Activities

1) Determine test strategy
2) Determine adequacy of requirements
3) Generate functional test conditions

- Design Phase Activities

1) Determine consistency of design with requirements
2) Determine adequacy of design
3) Determine adequacy of the test plans
4) Generate structural and functional test conditions

- Program (Build) Phase Activities

1) Determine consistency with design
2) Determine adequacy of implementation
3) Generate structural and functional test conditions for modules and units

- Test Phase Activities

1) Test application system
2) Installation Phase Activities
3) Place tested system into production

- Maintenance Phase Activities

1) Modify and retest

7. Build the system test plan

- Using information from the prior steps, develop a System Test Plan to describe the testing that will occur.
- This plan will provide background information on the system being tested, test objectives and risks, the business functions to be tested, and the specific tests to be performed.
- The plan is then decomposed into specific tests and lower-level plans. After execution, the results from the specific tests are rolled up to produce a Test Report.

8. Build the unit test plans

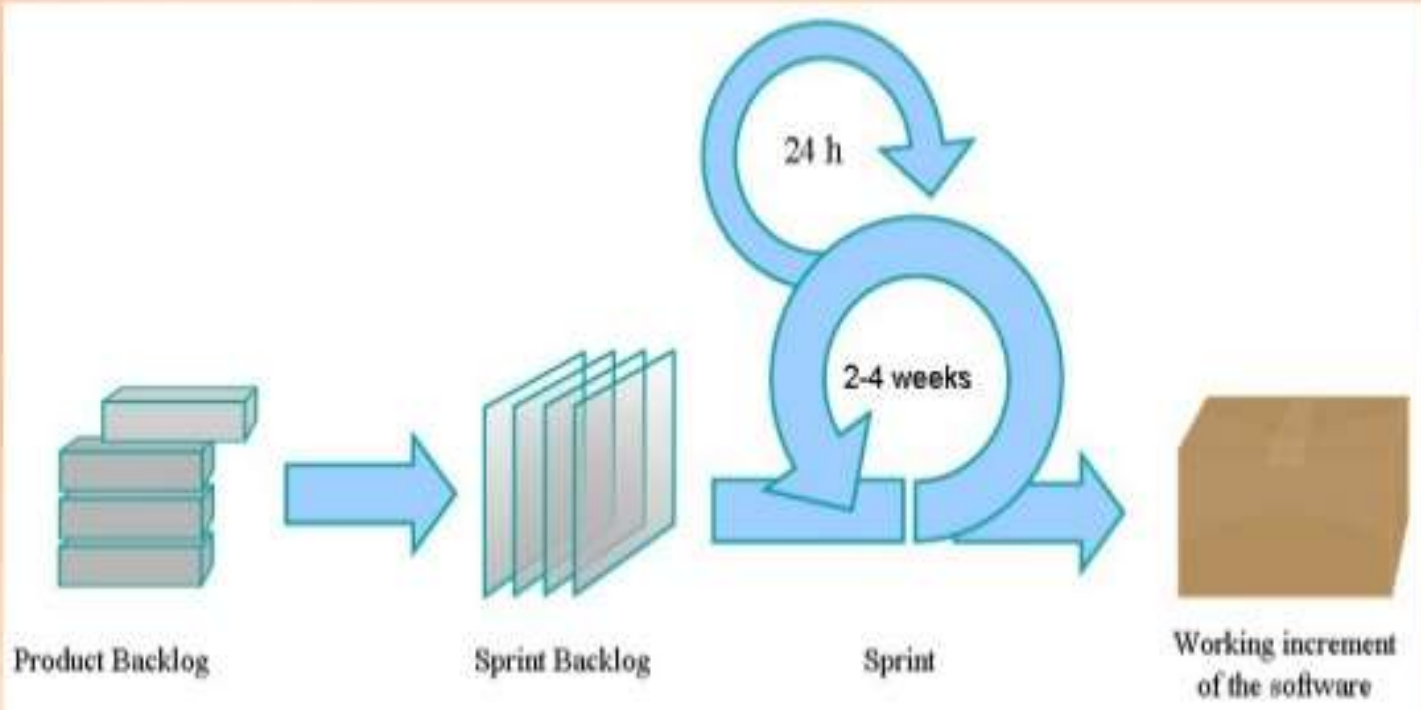
- During internal design, the system is divided into the components or units that perform the detailed processing. Each of these units should have an individual Test Plan.
- The importance of a Unit Test Plan is to determine when unit testing is complete.
- The extra effort spent in developing Unit Test Plans, testing units, and assuring that units are defect free prior to integration testing can have a significant payback in reducing overall test costs.

Agile Testing Process

Process Overview

Software development methodology

- Agile – iterative development methodology, where requirements evolve through collaboration between the customer and self-organizing teams. Agile business approach aligns development with customer needs.



Wrong ways of using a tester

- A tester is not the one who can not program and causes breaks only.
- Inadequate use of the person's potential and experience.
- A tester does not participate in planning.
- A tester's work starts after the functional development.

Why teams switch to Agile

- Simplicity of principles and apparent easiness of their employment.
- More transparent communication with the customer and accurate planning.
- Agile is in fashion.
- Examples of other teams.

Problems of switching to Agile

- Difficulty of leaving the traditional understanding of roles.
- Resistance to changes.
- In the adaptation phase the basic Agile principles may be destroyed.
- Agile is not a panacea to solve all the problems. Agile principles just reveal the problems, and it is people who are to solve them.

Adequate approach to a tester

- A tester is a team member!
- The tester's experience in development should be fully used.
- The tester's functions are largely determined by the tester himself.
- Participation in planning and requirements analysis.

Changes in testing

- Start of a sprint – start of testing.
- Testing activities are the same as in other methodologies.
- Testing tasks – usual sprint tasks.
- All the team is responsible for the result, including quality.
- Testing tasks – common tasks of all the team.
- Maximum communication.
- Provide continuous feedback.
  - To the customer.
  - To the programmers.

Types of testing

- Test planning.
- Build acceptance testing.
- Functional testing.
- Regression testing.
- Demo testing.
- Test automation.

Off-testing time

- Analysis of requirements and risks; test planning.
- Test automation.
- Setting up test environments.
- Preparing the necessary documentation.
- Work with external resources.
- Providing help to developers.

Problems with testing shortage

- Lack of testing resources.
- Inefficient test planning.
- A lot of manual testing.
- Inefficient testing process.
- Increasing functional for regression testing.

Ways of problem solving

- Detailed test planning.
- Automation, possibly with the developers' help.
- Increased unit test code coverage.
- Implementation of stabilization sprints.
- Increase in the amount of testing resources.

Benefits of being an Agile testers

- Less risk of compressed test period.
- Test all the time, not just at the end.
- Work together as one team towards a common goal.