



**Parul"
University**

Software Engineering

**Prof. Aseem Gupta Assistant Professor
AI & DS**





Unit-6

Software Testing and Quality Assurance



Contents- Software Testing

- **Software Testing Concepts**
- **Psychology of testing**
- **Levels of testing**
- **Testing Process**
 - **test plan, test case design, Execution**
- **Black-Box testing**
 - **Boundary value analysis**
- **Pair wise testing**
- **state based testing,**
- **White-Box testing**
 - **criteria and test case generation and tool support**



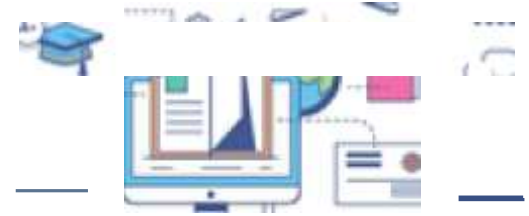
Contents- Quality Assurance

- Quality Control
- Assurance, Cost, Reviews
- Software Quality Assurance
- Approaches to SQA
- Reliability
- Quality Standards- ISO9000 And 9001



Concepts

- **Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.**
- **To understand testing techniques that are geared to discover program faults**
- **To introduce guidelines for interface testing**
- **To understand specific approaches to object-oriented testing**
- **To understand the principles of CASE tool support for testing**



Psychology of testing

- All tests should be traceable to customer requirements.
- Tests should be planned long before testing begins.
- The Pareto principle applies to software testing.
- Testing should begin “in the small” and progress toward testing “in the large.”
- Exhaustive testing is not possible.
- To be most effective, testing should be conducted by an independent third party.



Software testing phases

a Component testing

- Individual program components testing
- Tests are derived from the developer's experience



Software developer

Independent testing team

a Integration testing

- Testing of groups of components integrated to create a system or subsystem
- Tests are based on a system specification

Figure 6.1 Software testing phase



Testing process

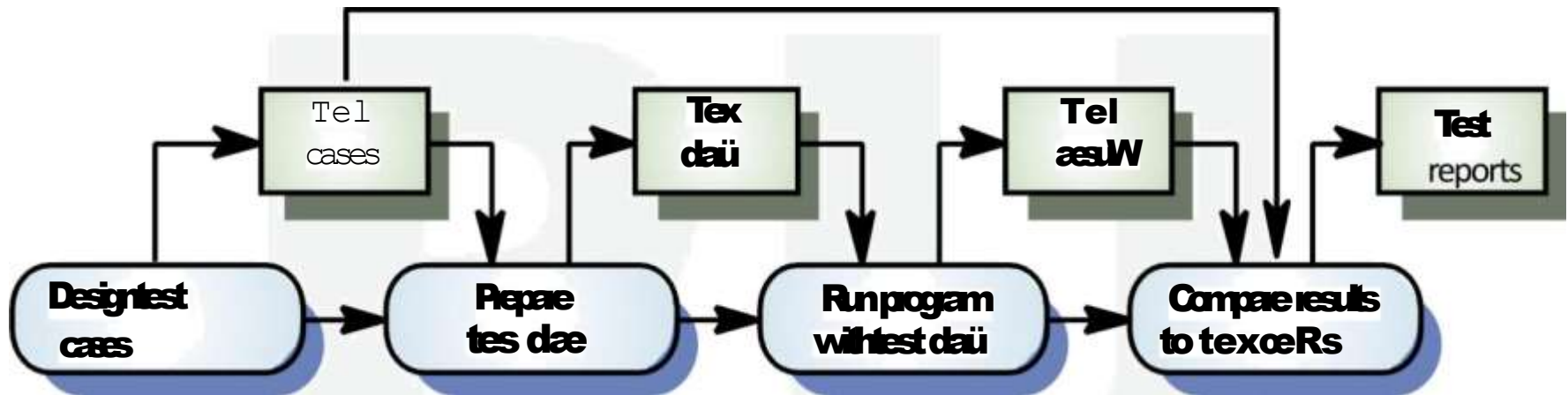


Figure 6.2 Software testing process



Test plan

- Test planning, the most important activity to ensure that there is initially a list of tasks and milestones in a baseline plan to track the progress of the project.
- It also defines the size of the test effort.
- It is the main document often called as master test plan or a project test plan and usually developed during the early phase of the project.



Test Planning Activities

- To determine the scope and the risks that need to be tested and that are NOT to be tested.
- Documenting Test Strategy.
- Deciding Entry and Exit criteria.
- Evaluating the test estimate.
- Planning when and how to test and deciding how the test results will be evaluated, and defining test exit criterion.

Ensuring that the test documentation generates repeatable test

s



Test case design

- Test case design methods provide a mechanism that can help to ensure the **completeness** of tests
- Provide the highest likelihood for uncovering **errors** in software.

Any product or system can be tested on two ways:

1. **Knowing** the specified function that a product has been designed to perform; (**Black Box**)
2. knowing **the** internal workings of a product, tests can be conducted to ensure that **"all** gears mesh," that **is**, internal operations are performed according to specifications (**White box** testing)

Test Execution

- Test execution is the process of executing the **code** and comparing *the* expected and **actual results**.
- Following factors are to be considered for a test execution process:
 - Based **On** a risk, select a subset of test suite **to** be **executed** for this cycle.
 - **Assign the test cases** in each **test suite** to testers for **execution**.
 - Execute tests, report bugs, and capture **test** status continuously.
 - Resolve blocking **issues as they arise**.
 - Report Status, **adjust assignments**, and reconsider plans and priorities **daily**.
 - Report test **cycle** findings and status.



Levels of testing

- There are different levels during the process of testing.
- Levels of testing include different methodologies that can be used while conducting software testing.

The main levels of software testing are -

- Functional Testing
- Non-functional Testing



Functional Testing

* **This** is a type of black box testing that is based **on** the specifications of the software that is to **be** tested.

• Functional testing **of** a software **is** conducted **on** a complete, integrated system to evaluate the system's compliance with its specified requirements.

There are five steps that are involved when testing an application for functionality.

1. The **determination** of the functionality that **the** **intended** application **is** meant to perform.
2. The creation of test **data** based on the specifications of the application.
3. The output based **on** the test data and the specification of the application.
4. The writing of test scenarios and the execution of test cases.
5. The comparison of actual **and** expected results based on the executed test cases.



Unit Testing

- This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases
- Unit testing is performed by the respective developers on the individual units of source code assigned areas.
- The developers use test data that is different from the test data of the quality assurance team.
- The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.



Integration Testing

- Integration testing is defined as the testing of combined parts of an application to determine if they function correctly.
- Integration testing can be done in two ways : Bottom-up integration testing and Top-down integration testing.

1. Bottom-up integration

This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.

2. Top-down integration

In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.



System Testing

- System testing **tests** the system **as a** whole.
 - This **type of** testing **is** performed by a specialized testing team.
- System testing is important because of the following reasons-
- System testing **is** the first step in the Software Development **Life** Cycle, **where the** application **is tested** as a **whole**.
 - The application **is** tested **thoroughly to** verify **that it meets** the functional **and technical** specifications.
 - The application **is** tested in an environment that **is** very close to the production environment where the application **will** be deployed.
 - System testing enables **us** to test, verify, and validate both the business **requirements** as well **as** the application architecture.



Non-Functional Testing

• **Non-functional testing** involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc.

Some of the important and commonly used non-functional testing types are discussed below.

- Performance Testing
- Load Testing
- Stress Testing
- **Usability Testing**
- **Security Testing**
- **Portability Testing**

Black-Box testing

- . An approach to testing where the program is considered as a 'black-box'
- . The program test cases are based on the system specification
- . Test planning can begin early in the software process

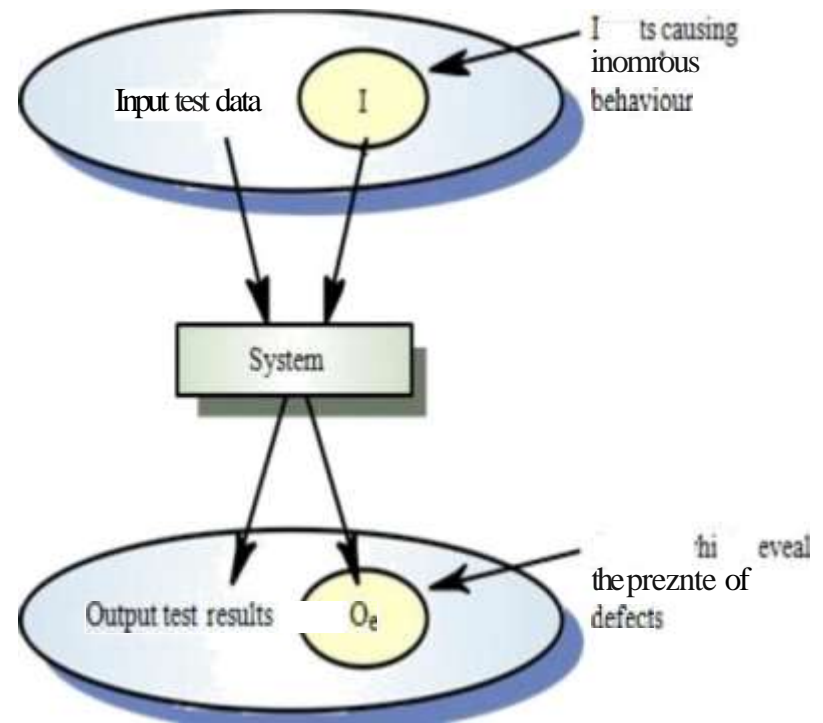


Figure 6.3 Black box testing



Continue__

- Also called *behavioral testing*, **focuses on the functional requirements of the software.**
- **It enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.**
- **Black box testing is not an alternative to white box techniques but it is complementary approach.**
- **Black box testing aims to find errors in the following categories:**
 - **Incorrect or missing functions,**
 - **Interface errors,**
 - **Errors in data structures or external data base access.**
 - **Behavior or performance errors,**
 - **Initialization and termination errors.**



Continue__

- Black-box testing purposely ignored control structure, attention is **FOCUSED** on **the** information **dD**main.
- Tests **are** designed **to** answer **the** **fD**llowing questions:
 - How **is** functional **validity** tested?
 - **HDw** **is** system behavior **and** performance **tested**?
 - What **classes** **of** input will make **gD**od test cases?
- By applying black-bDx techniques, we derive a set **Df** test cases that satisfy **the** following criteria
 - Test cases **that** **reduce** the number **of** additional **test** cases that must **be** **designed** **to** achieve reasonable testing (i.e minimize effort and time)
 - Test cases that tell us something about the presence **or** absence of **classes** **of** errors



Guidelines for BVA

- If an input condition specifies a range bounded by values a and b , test cases should be designed with values a and b and just above and just below a and b .
- If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.
- Apply guidelines 1 and 2 to output conditions.
- If internal program data structures have prescribed boundaries be certain to design a test case to exercise the data structure at its boundary.



Pair wise testing

- Pair-wise Testing is a type of software testing in which permutation and combination method is used to test the software.
- used to test all the possible discrete combinations of the parameters involved.
- It is a P&C based method, in which to test a system or an application, for each pair of input parameters of a system, all possible discrete combinations of the parameters are tested.
- as using the conventional or exhaustive testing approach it may be hard to test the system
- but by using the permutation and combination method it can be easily done.



Advantages of Pair-wise Testing

The advantages of pair-wise testing are:

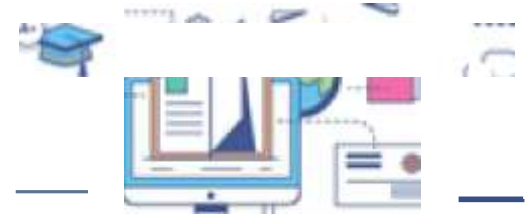
- Pair-wise testing reduces the number of execution of test cases.
- Pair-wise testing increases the test coverage almost up to hundred percentage.
- Pair-wise testing increases the defect detection ratio.
- Pair-wise testing takes less time to complete the execution of the test suite.
- Pair-wise testing reduces the overall testing budget for a project.



Disadvantages of Pair-wise Testing

The disadvantages of pair-wise testing are:

- Pair-wise testing is not beneficial if the values of the variables are inappropriate.
- In pair-wise testing it is possible to miss the highly probable combination while selecting the test data.
- In pair-wise testing, defect yield ratio may be reduced if a combination is missed.
- Pair-wise testing is not useful if combinations of variables are not understood correctly.



State based testing

- **To understand the objects that are modeled in software and the relationships that connect these Objects.**
- **Next step is to define a series of tests that verify “all objects have the expected relationship to one another.**
- **Stated in other way:**
 - **Create a graph of important objects and their relationships**
 - **Develop a series of tests that will cover the graph, So that each Object and relationship is exercised and errors are uncovered.**

Begin by creating graph -

- **a collection of nodes that represent objects**
- **links that represent the relationships between Objects**
- **node weights that describe the properties of a node**
- **link weights that describe some characteristic of a link.**



Continue...

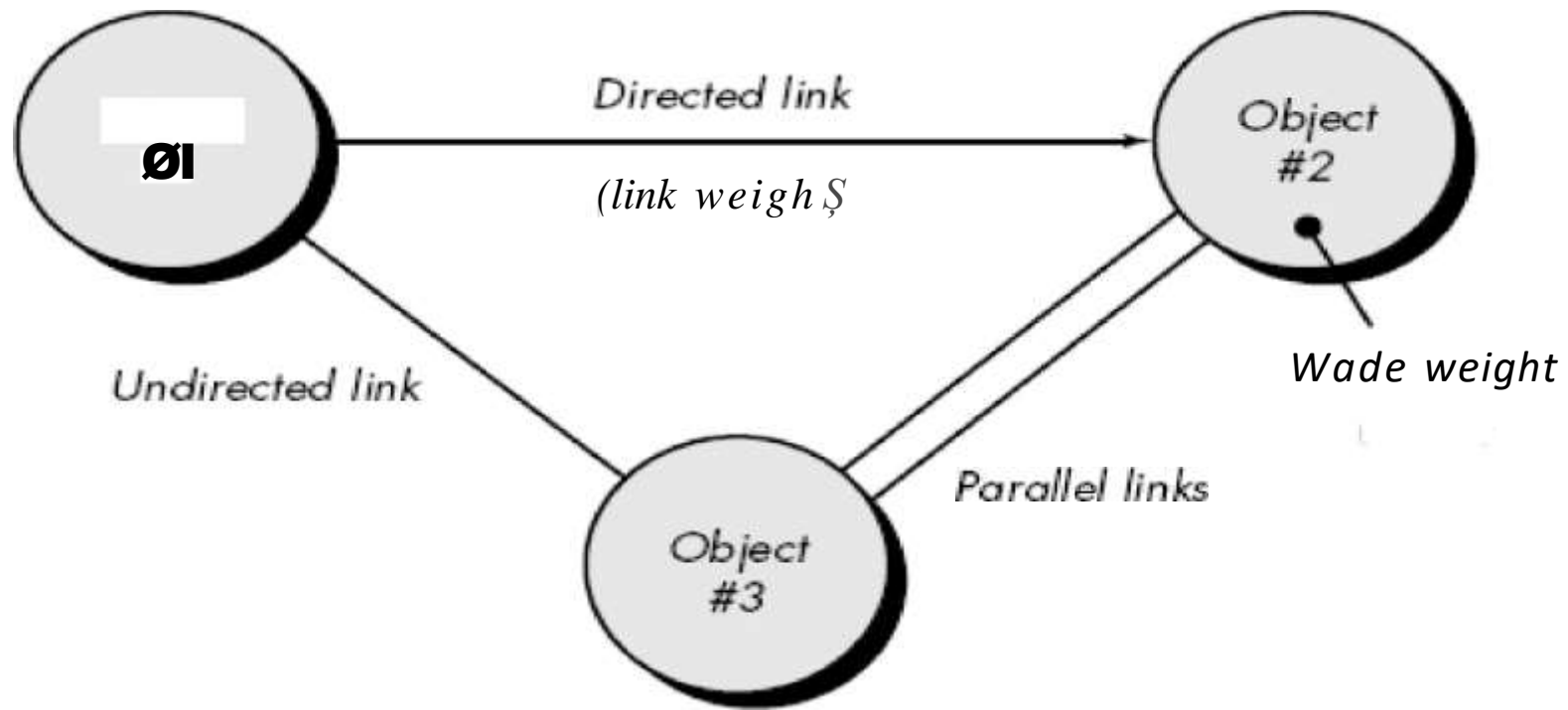


Figure 6.4 Graph notation





Continue...

- Nodes are represented as circles connected by links that take a number of different forms.
- A directed link (represented by an arrow) indicates that a relationship moves in only one direction.
- A bidirectional link, also called a symmetric link, implies that the relationship applies in both directions.
- Parallel links are used when a number of different relationships are established between graph nodes.



Continue...

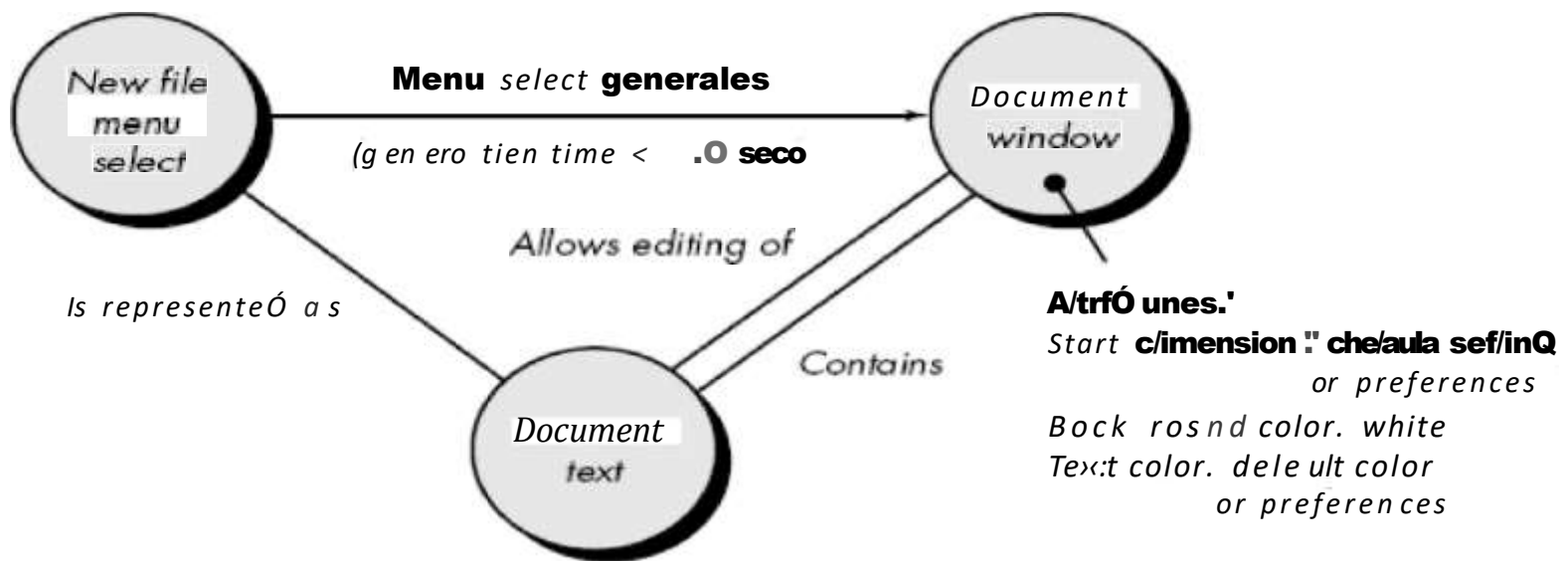


Figure 6.5 Simple example



Continue...

- Object d1 = new file menu select
- Object #2 = document window
- Object d3 = document text

Referring to example figure, a menu select on new file generates a document window.

- The link weight indicates that the window must be generated in less than 1.0 second.
- The node weight of document window provides a list of the window attributes that are to be expected when the window is generated.
- An undirected link establishes a symmetric relationship between the new file menu select and document text,
parallel links indicate relationships between document window and document text



Continue...

- Number of behavioral testing methods that can make use of graphs:

Transaction flow modeling.

- The nodes represent steps in some transaction and the links represent the logical connection between steps

Finite state modeling.

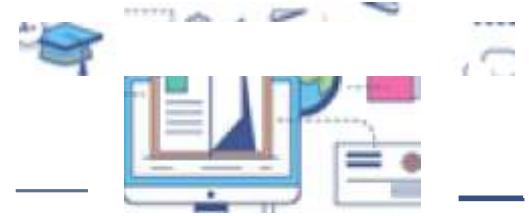
- The nodes represent different user observable states of the software and the links represent the transitions that occur to move from state to state. (Starting point and ending point)

Data flow modeling.

- The nodes are data objects and the links are the transformations that occur to translate one data object into another.

Timing modeling.

- The nodes are program objects and the links are the sequential connections between those objects.
- Link weights are used to specify the required execution times as the program executes.



White-Box testing

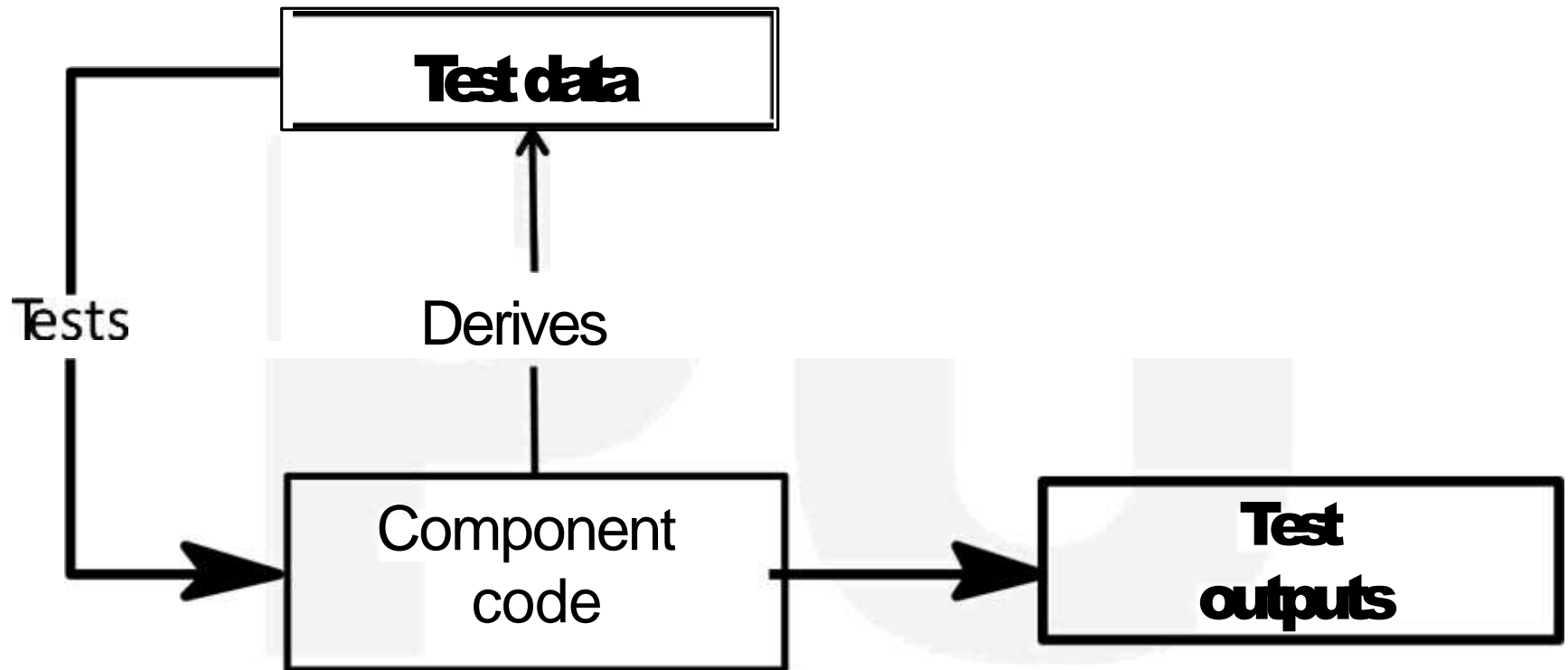
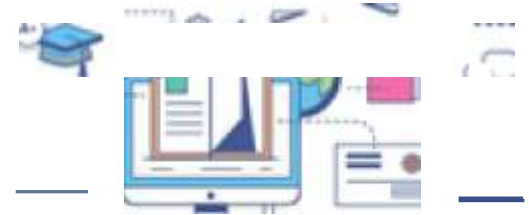


Figure 6.6 Architecture of White -Box testing



Continue...

- White-box testing **is** a test case **design** method
- **Uses the control** structure **of** the procedural **design to** derive test cases
- Using white-box testing methods, the software engineer **can** derive test cases that
- Guarantee that **all** independent paths within a module have been exercised at **least** once
- Exercise all logical decisions **on** their true and **false** sides
- Execute all **loops** at their boundaries and within their operational bounds
- Exercise internal data structures to ensure their validity.



Basis path testing

Basis path testing is a white-box testing technique

- **To derive** a logical complexity measure of a procedural design.
- **Test cases derived to exercise the basis** set are guaranteed **to execute every statement in the program at least one time.**

Methods:

- 1** Flow graph notation
- 2** Independent program paths **or Cyclomatic complexity**
- 3** Deriving test **cases**
- 4** Graph Matrices

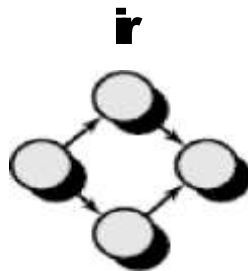


Flow Graph Notation

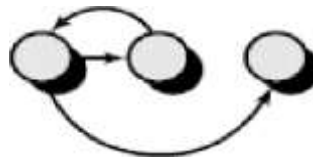
- Start with simple notation for the representation of control flow (called flow graph). It represent logical control flow.

The structured constructs in flow graph form:

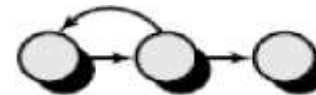
Sequence



While



Until



Case

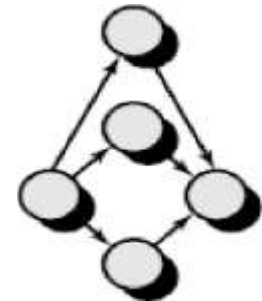


Figure 6.7 Flow graph notation

Continue_

- A sequence of process boxes and decision diamond can map into a single node.
- The arrows on the flow graph, called edges or links, represent flow of control and are parallel to flowchart arrows.
- An edge must terminate at a node, even if the node does not represent any procedural statement.
- Areas bounded by edges and nodes are called regions.
- When compound condition are encountered in procedural design, flow graph becomes slightly more complicated.

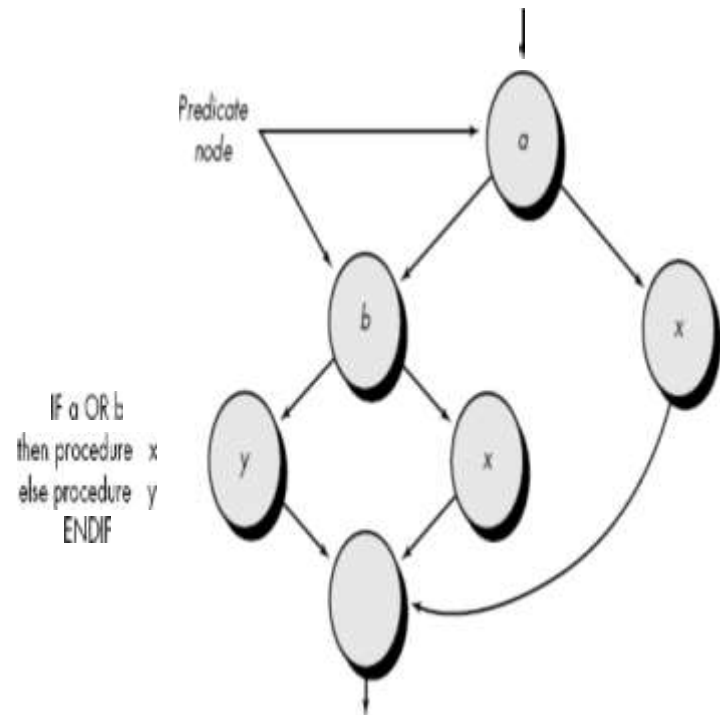


Figure 6.8 Compound logic



Continue...

When we translating PDL segment into flow graph, separate node is created for each condition.

Each node that contains a condition is called predicate node and is characterized by two or more edges coming from it.



Independent program paths or Cyclomatic complexity

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

For example, a set of independent paths for flow graph:

Path 1: 1-11

Path 2: 1-2-3-4-5-10-1-11

Path 3: 1-2-3-6-8-9-1-11

Path 4: 1-2-3-6-7-9-1-11

- Note that each new path introduces a new edge.
 - The path 1-2-3-4-5-10-1-2-3-6-8-9-1-11 is not considered to an independent path because it is simply a combination of already specified paths and does not traverse any new edges.
 - Test cases should be designed to force execution of these paths (basis set).



Continue...

- Cyclomatic complexity is a software metrics that provides a quantitative measure of the logical complexity of a program. It defines no. of independent paths in the basis set and also provides number of test that must be conducted.

One of three ways to compute cyclomatic complexity:

1. The no. of regions corresponds to the cyclomatic complexity.
 2. Cyclomatic complexity, $V(G)$, for a flow graph, G , is defined as

$$V(G) = E - N + 2$$
 where E is the number of flow graph edges, N is the number of flow graph nodes.
 3. Predicate complexity, $V(G)$, for a flow graph, G , is also defined as

$$V(G) = P + 1$$
 where P is the number of predicate nodes edges.
- So the value of $V(G)$ provides us with upper bound of test cases.



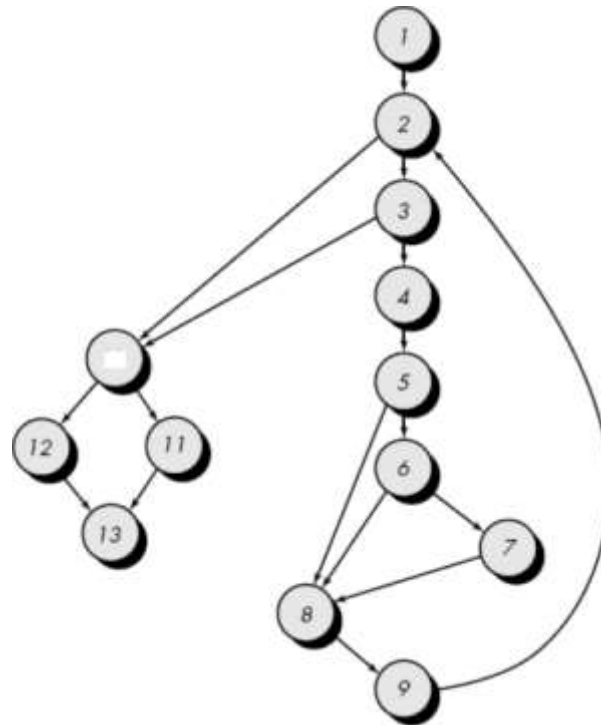
Deriving Test Cases

- It is a series of steps method.
- The procedure average depicted in PDL
- Average, an extremely simple algorithm, contains compound CDNs and loops.

To derive basis set, follow the steps.

1. Using the design or code as a foundation, draw a corresponding flow graph.
 - A flow graph is created by numbering those PDL statements that will be mapped into corresponding flow graph nodes.

Deriving Test Cases



Flow graph for the procedure average



Continue...

2. Determine the time complexity of the resultant flow graph.
 - VC can be determined without developing a flow graph by counting all conditional statements in the PDL (for the procedure average, compound conditions count as 1) and adding 1
 - $fiG) - 6$ regions
 - $V(C) = 17 \text{ edges} - 13 \text{ nodes} + 2 = 6$
 - $fiG) - 5 \text{ predicate nodes} + 1 = 6$
3. Determine a basis set of linearly independent paths
 - The value of $U(C)$ provides the number of linearly independent paths through the program control structure.
 - path 1. 1-2-10-11-13
 - path 2. 1-2-10-12-13
 - path 3. 1-2-3-10-11-13
 - path 4. 1-2-3-4-5-B-9-2-..
 - path 5. 1-2-5-4-5-6-8-9-2-...
 - path 6. 1-2-3-4-5-6-7-B-9-2-...
 - The ellipsis (..) following paths 4, 5, and 6 indicates that any path through the remainder of the control structure is acceptable.



Continue...

4. Prepare test cases that will force execution of each path in the basis set.
 - Data should be chosen so that conditions at the predicate nodes are appropriately set as each path is tested.
 - Each test case is executed and compared to expected results.
 - Once all test cases have been completed, the tester can be sure that all statements in the program have been executed at least once.

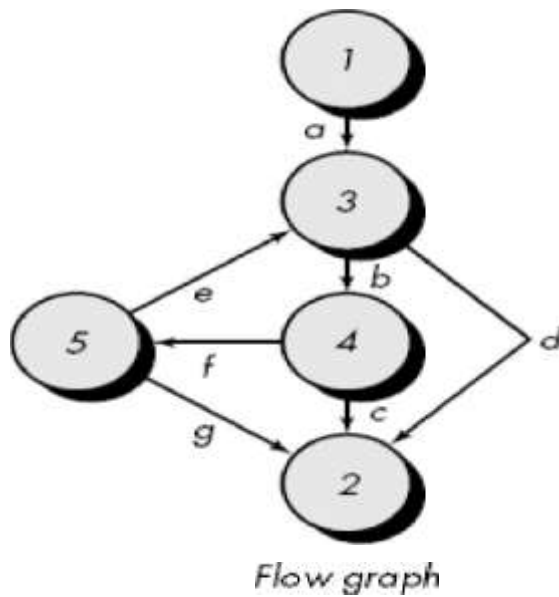


Graph Matrices

A **graph matrix** is a **square** matrix whose **size** (i.e., number of **rows** and **columns**) **is equal** to the number of nodes on the flow **graph**.

- **Each row and column** corresponds to an identified **node**, and matrix entries correspond to connections **(an edge)** between nodes.
- **Each node** on the flow graph is identified by **numbers**, while each **edge** is identified by **letters**.
- **The** graph matrix is nothing more than a tabular representation of a flow graph.
- By adding a **link weight** to each matrix entry, the graph matrix can become a powerful tool for evaluating **planned** control structure during testing. **The link weight** provides additional information about control flow. In its simplest form, the link **weight** is **1** (a connection exists) or **0** (a connection does not exist).

Continue...



Connected to
Node

4

e

f

Graph matrix

Connection matrix



Figure 6.9 Connection matrix



Connection matrix

- Each letter has been replaced with a **1**, indicating that a connection exists (this graph matrix is called a *connection matrix*).
- In fig. (connection matrix) each row with two or more entries represents a predicate node
- We can directly measure cognitive complexity value by performing arithmetic operations

$$\text{Connections} = \text{Each row Total no. of entries} - 1$$

$$V(G) = \text{Sum of all connections} + 1$$



Quality Control - Quality

- **Quality** as “a characteristic or attribute of something.”
- Two **kinds** of quality may be encountered:
 - **Quality of design** of a product increases, if the product is manufactured according to specifications.
 - **Quality of conformance** is the degree to which the design specifications are followed during manufacturing.
- In software development
 - **Quality of design** encompasses requirements, specifications, and the design of the system.
 - **Quality of conformance** is an issue focused primarily on implementation.
- User satisfaction = compliant product + good quality + delivery within budget and schedule



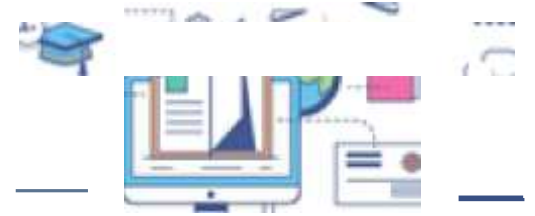
Quality control

- Quality control invokes the series of inspections, reviews, and tests used throughout the software process.
- Quality control includes a feedback loop to the process.
- A key concept of quality control is that all work products have defined, measurable specifications to which we may compare the output of each process.
- The feedback loop is essential to minimize the defects produced.



Quality Assurance

- **Quality assurance** consists of the auditing and reporting functions of management.
- If the data provided through quality assurance identify problem
- it is management's responsibility to address the problems and
- apply the necessary resources to resolve quality issues.



Cost of Quality

- The **cost of quality** includes all **costs incurred** in the pursuit of quality or in performing quality-related activities.
- Quality **costs** may be divided 3 mode of cost:
 - Prevention
 - Appraisal
 - Failure.

Continue_

- **Prevention costs include:**
 - **Quality planning**
 - **Formal technical review** vs
 - **Test equipment**
 - **Training**
- **Appraisal costs include activities to gain insight into product**
 - **In-process and Inter-process inspection**
 - **Equipment calibration and maintenance**
 - **Testing**

Continue_

- **Failure costs**
 - **Internal Failure Cost**
 - **rework**
 - **repair**
 - **failure mode analysis**
- **External Failure Cost**
 - **complaint resolution**
 - **product return and replacement**
 - **help line support**
 - **warranty work**



Software Quality Assurance

Definition:

Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

- **Definition serves to emphasize three important points:**
 - **Software requirements are the foundation from which quality is measured.**
 - **If specified standards criteria are not followed, lack of quality will almost surely result.**
 - **A set of implicit requirements often goes unmentioned.**



SQA group Activity

- SQA group made up of software engineers, project managers, customers, sales people, and the individuals members.
- SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis and reporting.
- Software engineers address quality activities @ applying technical methods and measures, conducting formal technical reviews, and performing well-planned software testing.
- SQA group is to assist the software team in achieving a high quality end product.

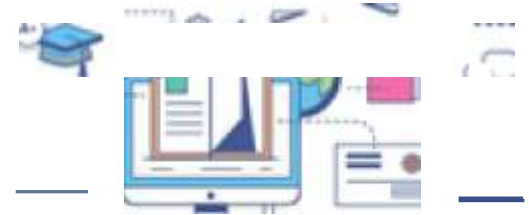


Role of an SQA group

1. Prepares an **SQA** plan for a project.

The plan is developed during project planning and is reviewed by all stakeholders.

- The plan identifies
 - Evaluations to be performed
 - Audits and reviews to be performed
 - Standards that are applicable to the project
 - Procedures for error reporting and tracking
 - Documents to be produced by the **SQA** group
 - Amount of feedback provided to the software project team



Continue...

2. Participates in the development of the project's software process description.
3. Reviews software engineering activities to verify compliance with the defined software process.
8. Audits designated software work products to verify compliance with those defined as part of the software process.
5. The SQA group reviews selected work products;



Software Reliability

- **Software reliability is defined in statistical terms as "the probability of failure-free Operation of a computer program in a specified environment for a specified time".**

- **What is meant by the term failure?**

- **Failure is nonconformance to software requirements.**

- **Correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures.**

Software reliability can be measured directly and estimated using historical and developmental data.



Measures of Reliability and Availability

- A simple measure of reliability is mean-time-between-failure (MTBF), where

$$MTBF = MTTF + MTTR$$

- The acronyms MTTF and MTTR are mean-time-to-failure and mean-time-to-repair, respectively.
- MTBF is a far more useful measure than defects/KLOC or defects/FP.
- Stated simply, an end-user is concerned with failures, not with the total error count.
- a reliability measure, we must develop a measure Df availability



Continue...

- **Software availability** : It is the probability that a program is operating according to requirements at a given point in time and is defined as

$$\text{Availability} = \left[\frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \right] \times 100\%$$

- The 'MTBF' reliability measure is equally sensitive to MTTF and MTTR.
- The availability measure is somewhat more sensitive to MTTR, an indirect measure of the maintainability of software



Quality Standards- ISO9000 And 9001

SQA Standards

• Software quality assurance standards can be classified into two main classes —

1. **Software** quality assurance management standards, including certification and assessment methodologies (quality management standards)
2. Software **project** development process standards (**project** process standards)



Quality Management Standards

- These focus on the organization's SQA system, infrastructure and requirements,
- while leaving the choice of methods and tools to the organization.
- With quality management standards, organizations can steadily assure that their software products achieve an acceptable level of quality.

Example —ISO 9000-3 and the Capability Maturity Model (CMIM)



ISO 9001 Certification

- **ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies.**
- **ISO technical committees prepare the International Standards.**
- **ISO collaborates closely with the International Electro-technical Commission (IEC) on all matters of electro-technical standardization.**
- **International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2**
- **Draft of the International Standards adopted by the technical committees is circulated to the member bodies for voting.**
- **ISO 9001 was prepared by Technical Committee ISO/TC 176, Quality management and quality assurance, Subcommittee SC 2, Quality systems.**



References

- Pressman, Roger S. Software engineering: a practitioner's approach. Palgrave Macmillan, 2005.
- Sommerville, Ian. Software Engineering: Pearson New International Edition. Pearson Education Limited, 2013.
- Jalote, Pankaj. An integrated approach to software engineering. Springer Science & Business Media, 2012.
- www.google.com

- DIGITAL LEARNING CONTENT

0



Parul[®] University



www.paruluniversity.ac.in

00