Testing Concepts

Lesson 2: Types of Testing Techniques & Test Case Design

Lesson Objectives

To understand the following topics:

- Verification and Validation
- Types of Testing Techniques
- Static & Dynamic Testing Techniques
- Introduction to Static Testing Techniques
- Static Testing Techniques – Defects Detected & Benefits
- Review Process Success Criteria
- Introduction to Dynamic Testing
- Types of Dynamic Testing Techniques
- White Box Testing Techniques
- Black Box Testing Techniques
- Experienced based Testing Techniques
- Choosing a test technique
- Static vs. Dynamic Testing
- A good Test Case
- Test Case Lifecycle
- Test Case Design Techniques

Lesson Objectives

To understand the following topics:

- What is test data?
- Properties of Good Test Data
- Test Data team
- Test data lifecycle

Lesson Objectives

To understand the following topics:

- What is Positive Testing?
- Advantages/Limitations of positive testing
- What is negative testing?
- Advantages/Limitations of negative testing
- Positive & Negative test scenarios
- What is Basic test?
- Example on Basic test
- What is Alternate test?
- Example on Alternate test
- Importance of writing positive, negative, basic, alternate test while designing test cases
- Best practices for test case maintenance

Verification and Validation

Verification

- Verification refers to a set of activities which ensures that software correctly implements a specific function.
- Purpose of verification is to check: Are we building the product right?
- Example: code and document reviews, inspections, walkthroughs.
- It is a Quality improvement process.
- It is involve with the reviewing and evaluating the process.
- It is conducted by QA team.
- Verification is Correctness.

Verification and Validation (cont.)

Validation

- Purpose of Validation is to check : Are we building the right product?
- Validation refers to a different set of activities which ensures that the software that has been built is traceable to customer requirements.
- After each validation test has been conducted, one of two possible conditions exist:
  - 1. The function or performance characteristics conform to specification and are accepted, or
  - 2. Deviation from specification and a deficiency list is created.

Example : a series of black box tests that demonstrate conformity with requirements.

- It ensures the functionality.
- It is conducted by development team with the help from QC team.
- Validation is Truth.
- Validation is the following process of verification.

Types of Testing Techniques

Static Testing

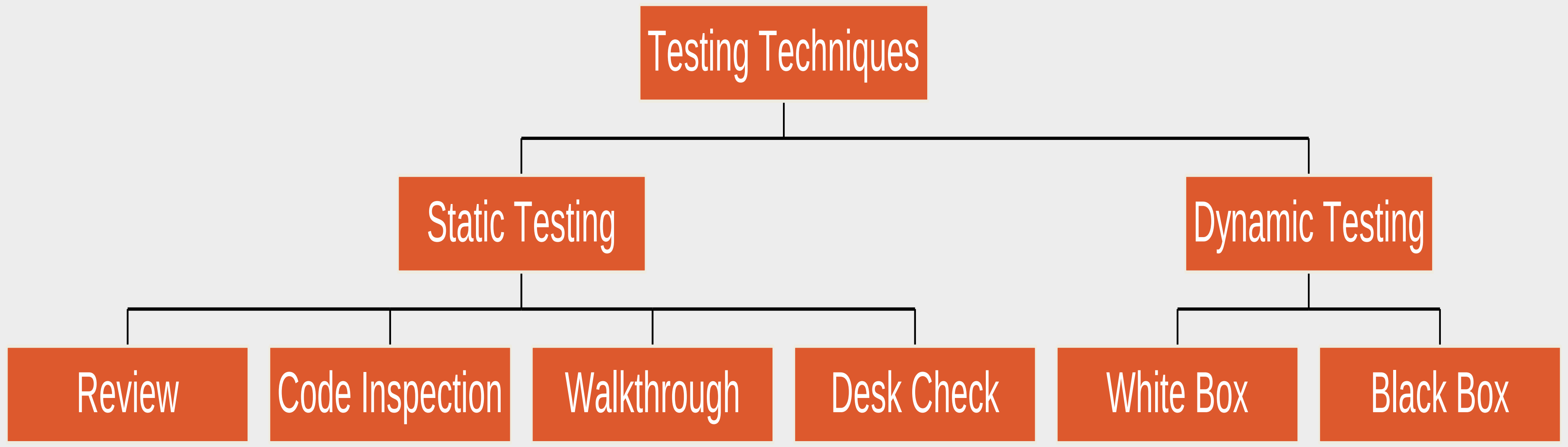
- It is a ***verification*** process
- Testing a software without execution on a computer. Involves just examination/review and evaluation
- It is done to test that software confirms to its SRS i.e. user specified requirements
- It is done for ***preventing*** the defects

Dynamic Testing

- It is a ***validation*** process
- Testing software through executing it
- It is done to test that software does what the user really requires
- It is done for ***detecting*** the defects

Static & Dynamic Testing Techniques

Types of Testing Techniques



Introduction to Static Testing Techniques

Static Testing is a process of reviewing the work product and reviewing is done using a checklist

Static Testing helps weed out many errors/bugs at an early stage

Static Testing lays strict emphasis on conforming to specifications

Static Testing can discover dead codes, infinite loops, uninitialized   and unused variables, standard violations and is effective in finding 30-70% of errors

Static Testing Methods

- Self Review
- Code Inspection
- Walk Through
- Desk Checking (Peer Review)
- Technical Review

Self Review

Self review is done by the person who is responsible for a particular program code

It is more of reviewing the code in informal way

It is more like who writes the code, understands it better

Self review is to be done by the programmer when he builds a new code

There are review checklists that helps programmer to verify with the common errors regarding the program code

Code Review Checklist

Data Reference Errors

- Is a variable referenced whose value is unset or uninitialized?

Data Declaration Errors

- Have all variables been explicitly declared?
- Are variables properly initialized in declaration sections?

Computation errors

- Are there any computations using variables having inconsistent data types?
- Is there any mixed mode computations?

Comparison errors

- Are there any comparisons between variables having inconsistent data types?

Control Flow errors

- Will every loop eventually terminate?
- Is it possible that, because of condition upon entry, a loop will never execute?

Interface errors

- Does the number of parameters received by these module equals the number of arguments sent by calling modules?
- Also is the order correct?

Input/output errors

- All I/O conditions handled correctly?

Code Inspection

Code inspection is a set of procedures and error detection techniques for group code reading.

Involves reading or visual inspection of a program by a team of people,

hence it is a group activity.

The objective is to find errors but not solutions to the errors

An inspection team usually consists of:

- A moderator
- A programmer
- The program designer
- A test specialist

Code Inspection

Before the Inspection

- The moderator distributes the program’s listing and design specification to the group well in advance of the inspection session

During the inspection

- The programmer narrates the logic of the program, statement by statement
- During the discourse, questions are raised and pursued to determine if errors exist
- The program is analyzed with respect to a check list of historically common programming errors

Code Inspection Helps in

- Detect Defects
- Conformance to standards/spec
- Requirements Transformation into product

Code Inspection

Goals Of Inspection:

1. To improve the quality of the document under inspection
2. Remove defects efficiently, as early as possible
3. Learn from defects found and improve processes in order to prevent recurrence of similar defects

Key characteristics of an Inspection:

1. It is usually led by a trained moderator
2. Rules and checklists are used during the preparation phase
3. The defects found are documented in a logging list or issue log
4. A formal follow-up is carried out by the moderator applying exit criteria

Code Walkthrough

Code Walkthrough is a set of procedures and error detection techniques for group reading.

Like code inspection it is also an group activity.

In Walkthrough meeting, three to five people are involved. Out of the three, one is moderator, the second one is Secretary who is responsible for recording all the errors and the third person plays a role of Test Engineer.

Solutions are also suggested by team members.

Walkthrough helps in

- Approach to Solution
- Find omission of requirements
- Style / Concepts Issues
- Detect Defects
- Educate Team Members

Code Walkthrough

Goals of Walkthrough:

1. To present the document to stakeholders in order to gather information
2. To explain (knowledge transfer) and evaluate the contents of the document
3. To establish a common understanding of the document

Key characteristics of walkthroughs:

1. The meeting is led by the authors
2. Scenarios and dry runs may be used to validate the content
3. Separate pre-meeting preparation for reviewers is optional

Desk Checking (Peer Review)

Human error detection technique

Viewed as a one person inspection or walkthrough

A person reads a program and checks it with respect to an error list and/or walks test data through it

Less effective technique

Best performed by the person other than the author of the program

Technical Review

- A technical review is a discussion meeting that focuses on achieving consensus about the technical content of a document
- Compared to inspections, technical reviews are less formal
- There is little or no focus on defect identification
- During technical reviews defects are found by experts, who focus on the content of the document.

Experts needed for Technical Review:

1. Architects
2. Chief designers
3. key users.

Technical Review

Goals of Technical review:

1. Assess the value of technical concepts and alternatives in the product and project environment
2. Establish consistency in the use and representation of technical concepts
3. Ensure, at an early stage, that technical concepts are used correctly
4. Inform participants of the technical content of the document

Key characteristics of a technical review:

1. It is a documented defect-detection process that involves peers and technical experts
2. It is often performed as a peer review without management participation
3. Ideally it is led by a trained moderator
4. A separate preparation is carried out during which the product is examined and the defects are found

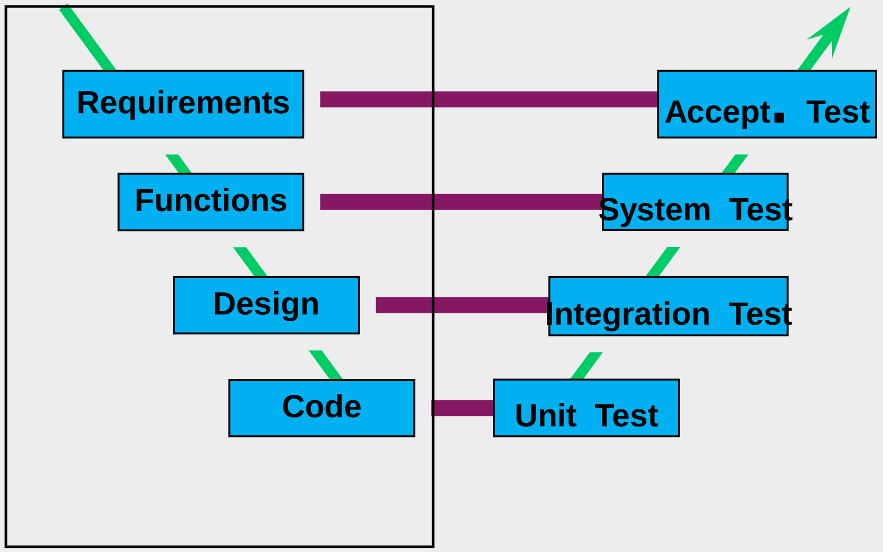
Static Testing Techniques – Defects Detected & Benefits

Types of defects found during static reviews:

- Deviations from standard
- Requirement defects
- Design defects
- Insufficient maintainability
- Incorrect interface specification

Benefits of Reviews:

- Early feedback on quality
- Development productivity improvement
- Reduced development timescales
- Reduced testing time and cost
- Lifetime cost reductions
- Reduced fault levels
- Increased awareness of quality issues



Review Process Success Criteria

Success factors for reviews:

Review should be with clear objective

Defects found are welcomed and expressed objectively

Management supports a good review process

The emphasize is on learning and process improvement

The right people for review are involved

Appropriate use of review techniques

Reviews are conducted in a fair & trustworthy atmosphere

Explicitly planning and tracking review activities

Train the participants in the review techniques – particularly the formal ones such as inspection

Introduction to Dynamic Testing

Dynamic Testing involves working with the software, giving input values and validating the output with the expected outcome

Dynamic Testing is performed by executing the code

It checks for functional behavior of software system , memory/CPU usage and overall performance of the system

Dynamic Testing focuses on whether the software product works in conformance with the business requirements

Dynamic testing is performed at all levels of testing and it can be either black or white box testing

Types of Dynamic Testing Techniques

White Box(Structure-based) Testing Techniques

- Code Coverage
  - Statement Coverage
  - Decision Coverage
  - Condition Coverage
  - Loop Testing
- Code complexity
  - Cyclomatic Complexity
- Memory Leakage

Black Box(Specification-based) Testing Techniques

- Equivalence Partitioning
- Boundary Value Analysis
- State Transition Testing
- Decision Table
- Use Case Testing

Experience-based Testing Techniques

- Error Guessing
- Exploratory Testing

White Box Test Techniques

White box is logic driven testing and permits Test Engineer to examine the internal structure of the program

Examine paths in the implementation

Make sure that each statement, decision branch, or path is tested with at least one test case

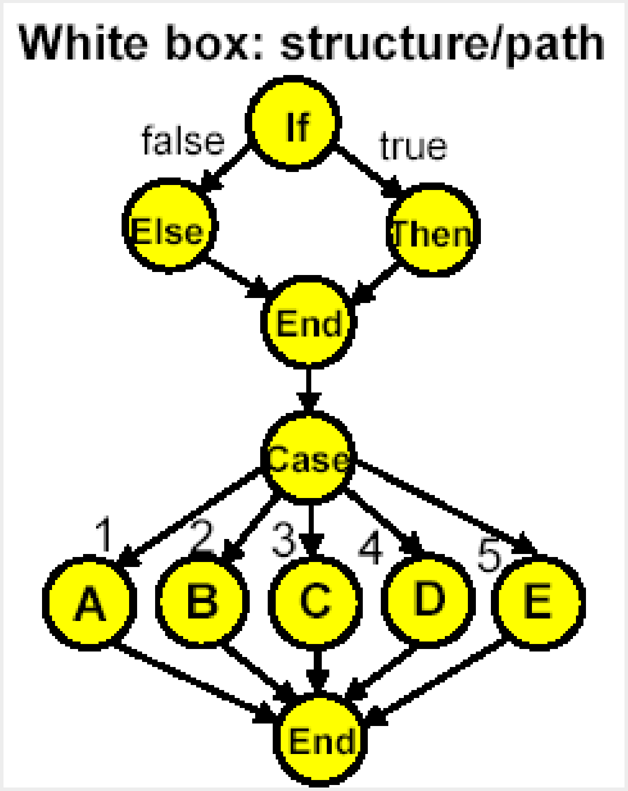
Desirable to use tools to analyze and track Coverage

White box testing is also known as structural, glass-box and clear-box

White Box Test Techniques

White Box Test Techniques

- Code Coverage
  - Statement Coverage
  - Decision Coverage
  - Condition Coverage
  - Loop Testing
- Code complexity
- Memory Leakage



Code Coverage

Measure the degree to which the test cases exercise or cover the logic (source code) of the program

Types

- Statement Coverage
- Decision Coverage
- Conditional Coverage
- Loop Testing

Statement Coverage

Test cases must be such that all statements in the program is traversed at least once

Consider the following snippet of code

void procedure(int a, int b, int **x)**

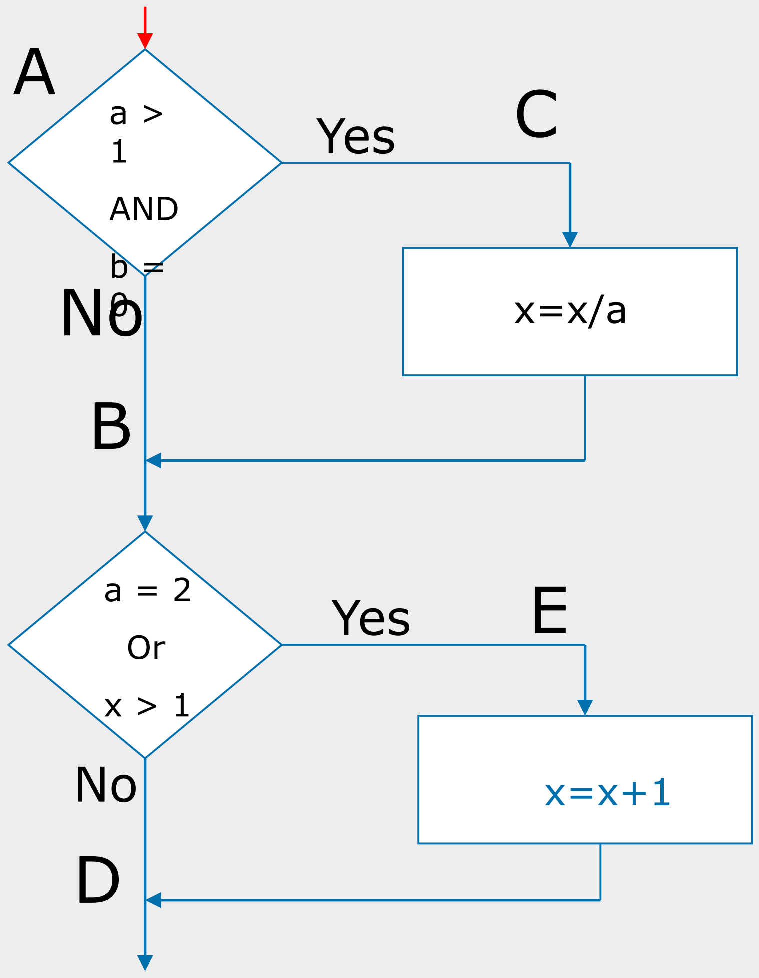
```
{    If (a>1) && (b==0)
                { x=x/a;         }
     If (a==2 || x>1)
                {  x=x+1; }
}
```

Statement Coverage

Test Case: a=2,b=0, x=3.

Every statement will be executed once.

But only path ACE will be covered and path ABD,ACD,ABE will not be covered.



Statement Coverage

In the above code one test case is sufficient to execute each of the two if statements at least once:

Test Case : a=2 , b=0 , x=3

(Decision1 is True, Decision2 is True)

However this test case does not help in detecting many of the many of the bugs which may go unnoticed as the false outcomes of the conditions a>1 & b=0 , a=2 or x>1 are not tested

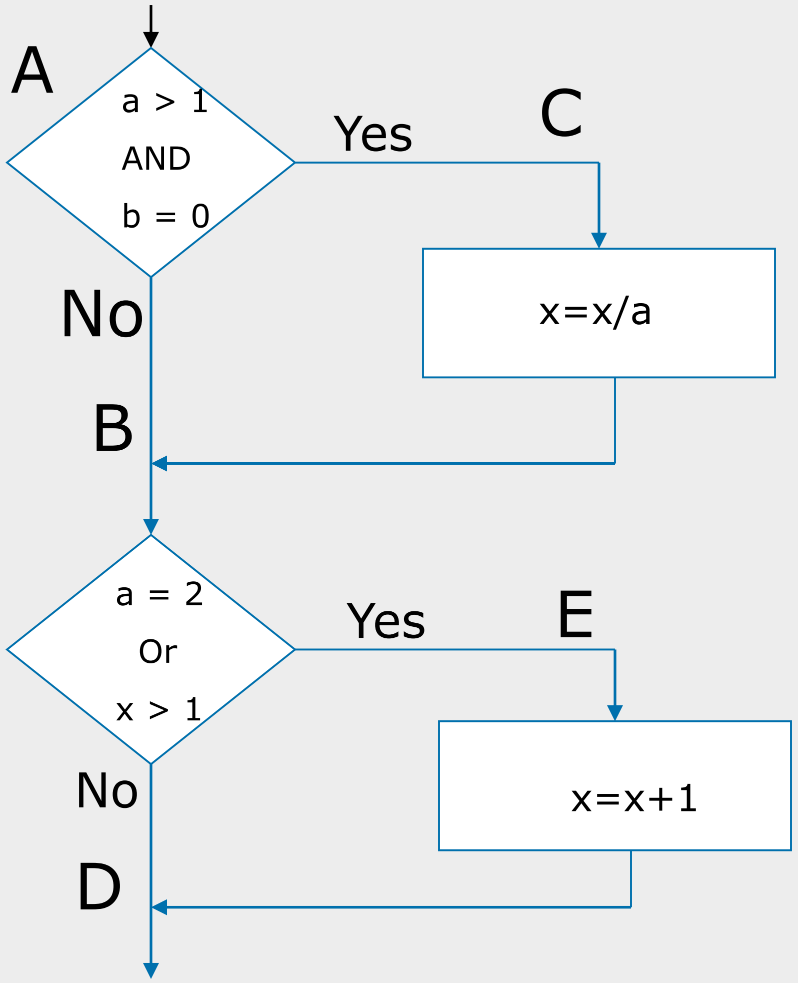
Decision Coverage

Test Case 1: a=2, b=0, x>1

(Decision1 is True, Decision2 is True) (Path ACE)

Test Case 2: a<=1 , b!=0, x<=1

(Decision1 is False, Decision2 is False) (Path ABD)



Condition Coverage

Test cases are written such that each condition in a decision takes on all possible outcomes at least once.

Test Case1 : a=2, b=0, x=3

(Condition1 is True,Condn2 is True)

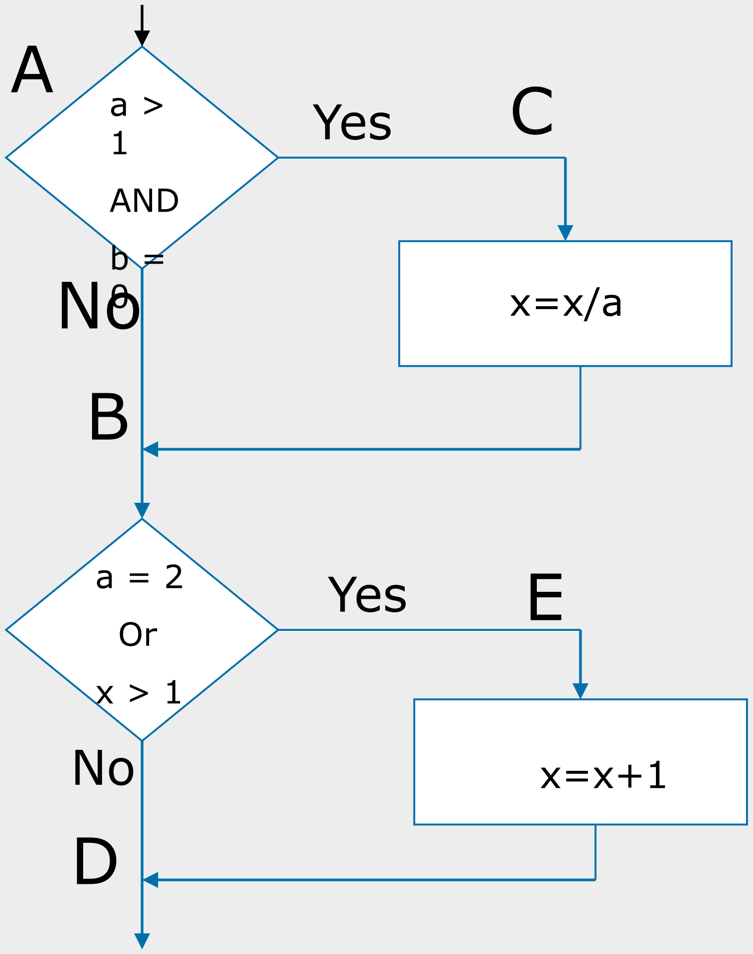
(Path ACE)

Test Case2: a=3, b=0, x=0

(Condn1 is True,Condn2 is False,Condn3

is False)

(Path ACD)



Condition Coverage

Test Case3 : a=1, b=0, x=3

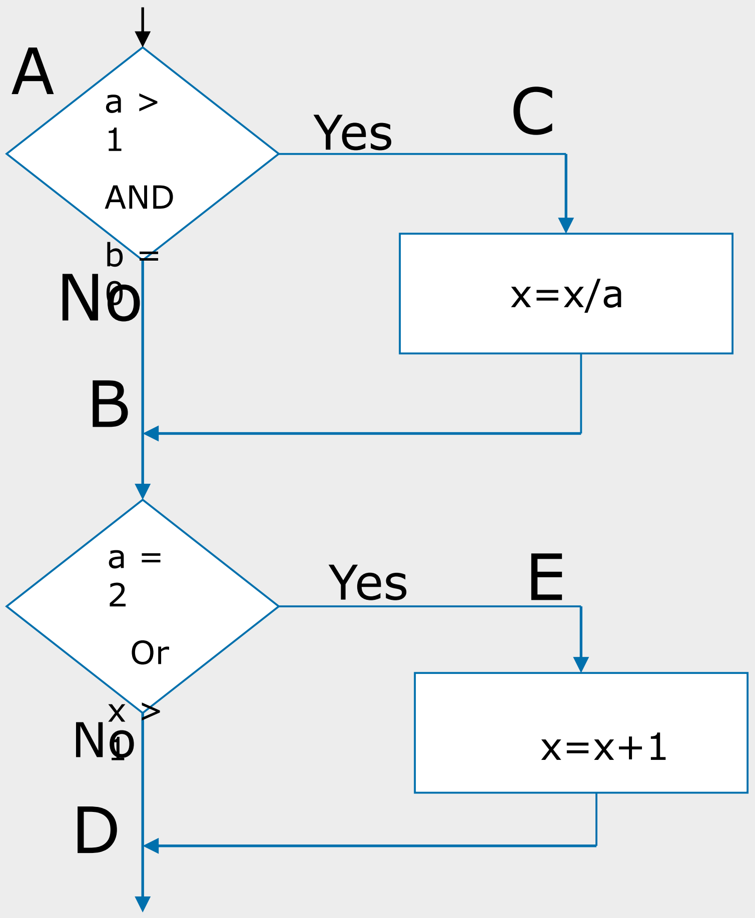
(Condition1 is False,Condition2 is True)

(Path ABE)

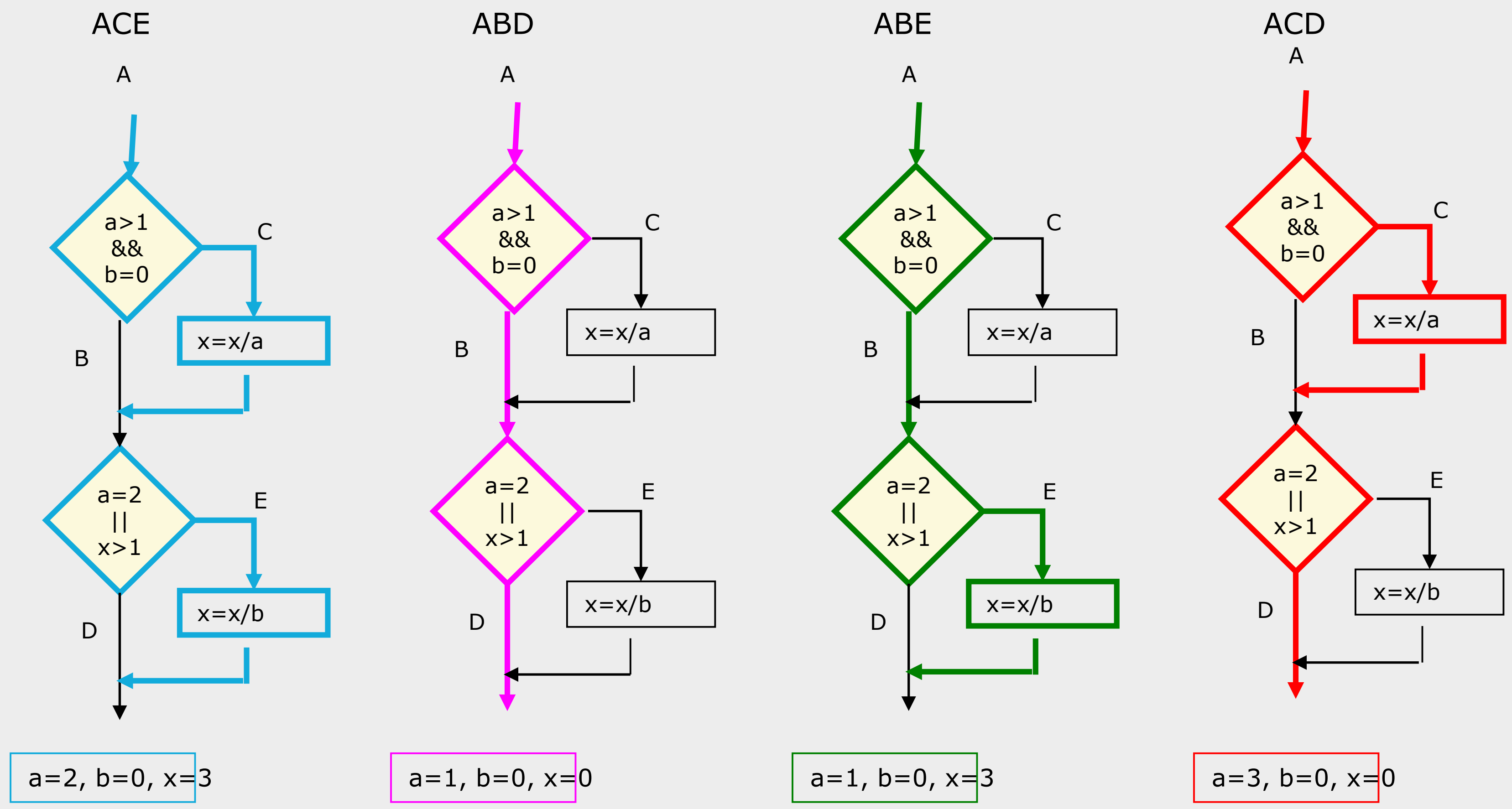
Test Case4: a=1, b=1, x=1

(Condition1 is False,Condition2 is False)

(Path ABD)



Condition Coverage



Loop Testing

Loops testing is a white box testing technique that focuses exclusively on validity of Loop construct

Types of loops

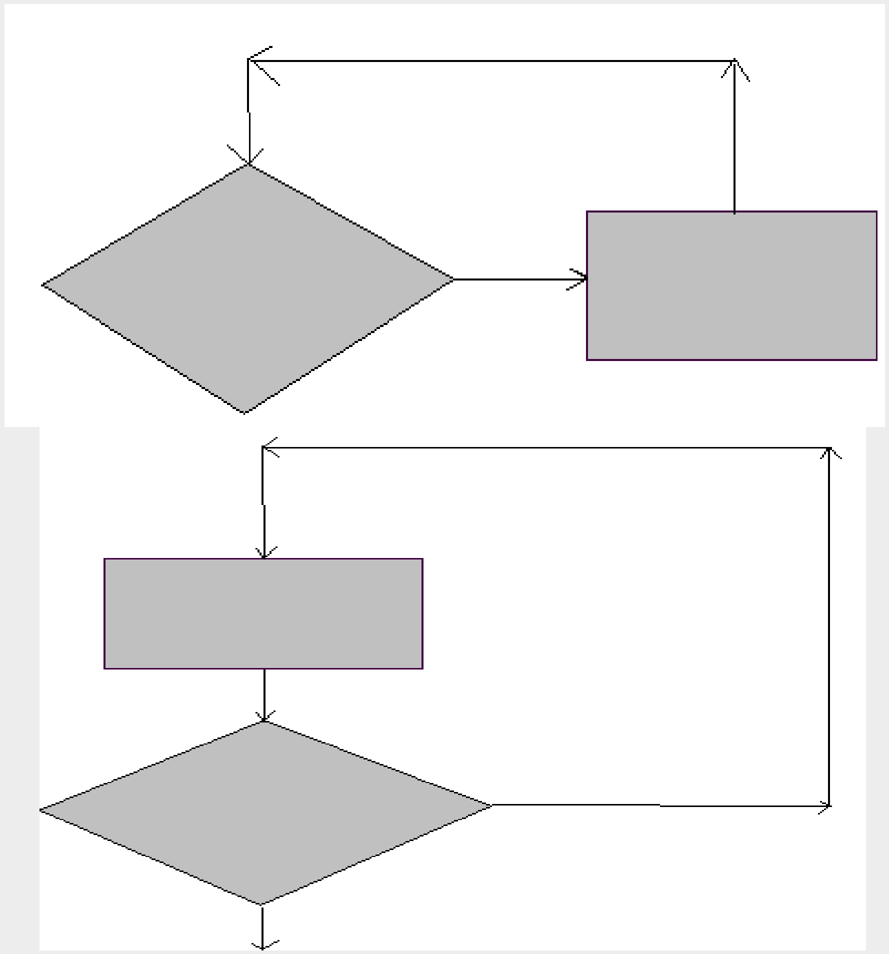
- Simple Loop
- Nested Loop
- Concatenated Loop
- Spaghetti Loop

Loop Testing

Simple Loop Testing Procedure:

- skip the entire loop
- only one pass through the loop
- make 2 passes through loop
- m passes through loop where m<n
- n-1, n, n+1 passes through the loop

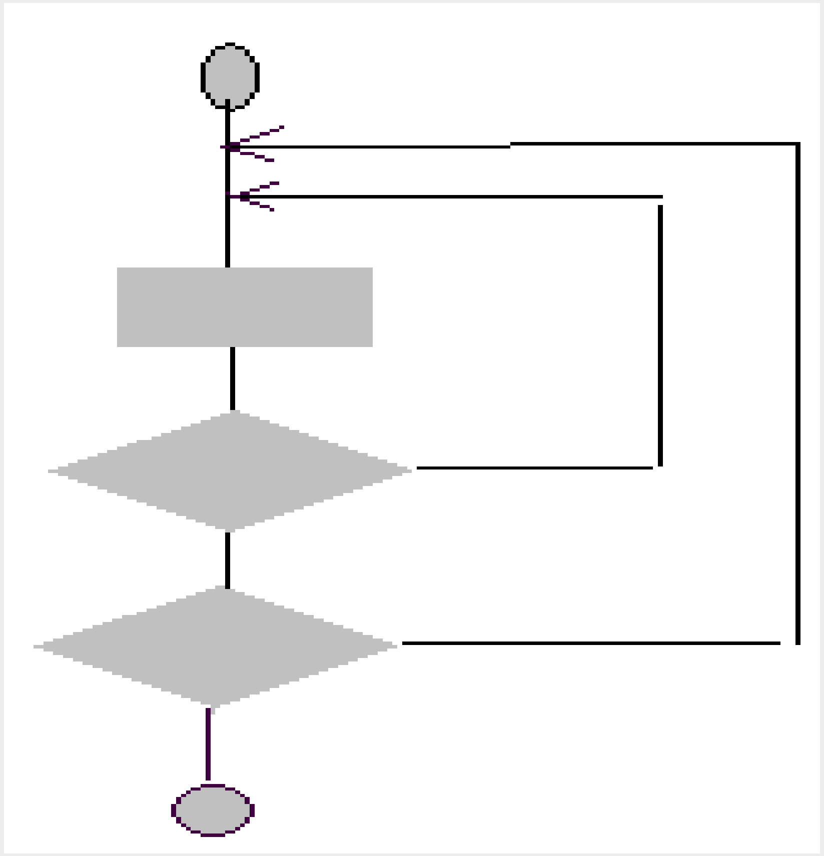
Where n is the maximum number of allowable passes through the loop



Loop Testing

Nested Loop Testing Procedure:

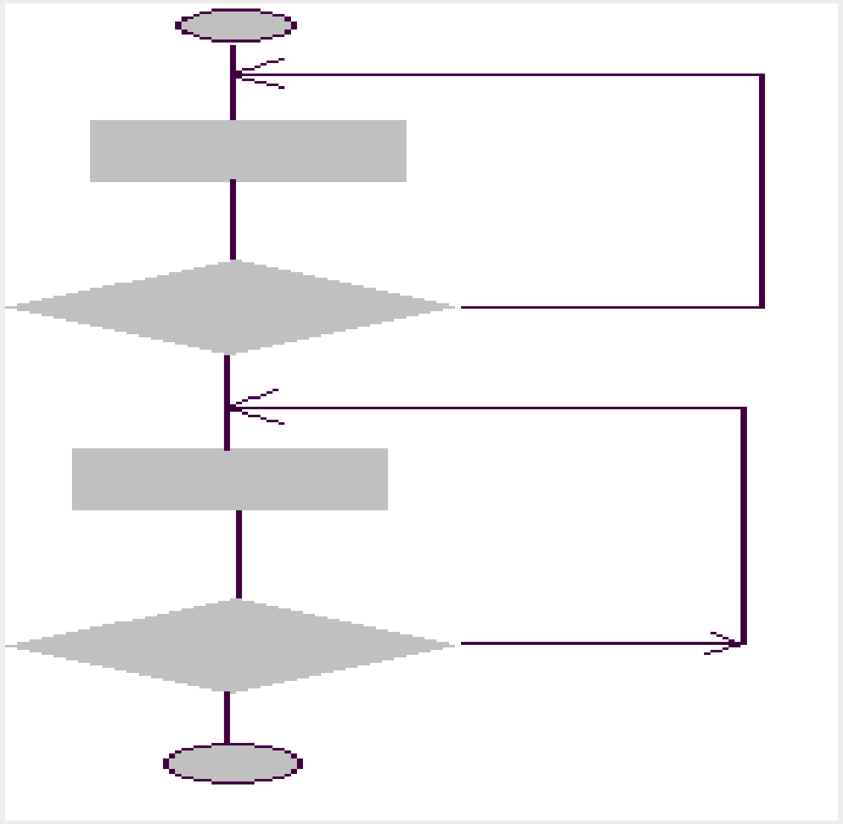
- start at the innermost loop
- conduct simple loop test for the innermost loop
- work outward, conducting tests for the
- next loop but keeping all other loops at minimum
- continue until all the outer loops are tested



Loop Testing

Concatenated Loop Testing Procedure:

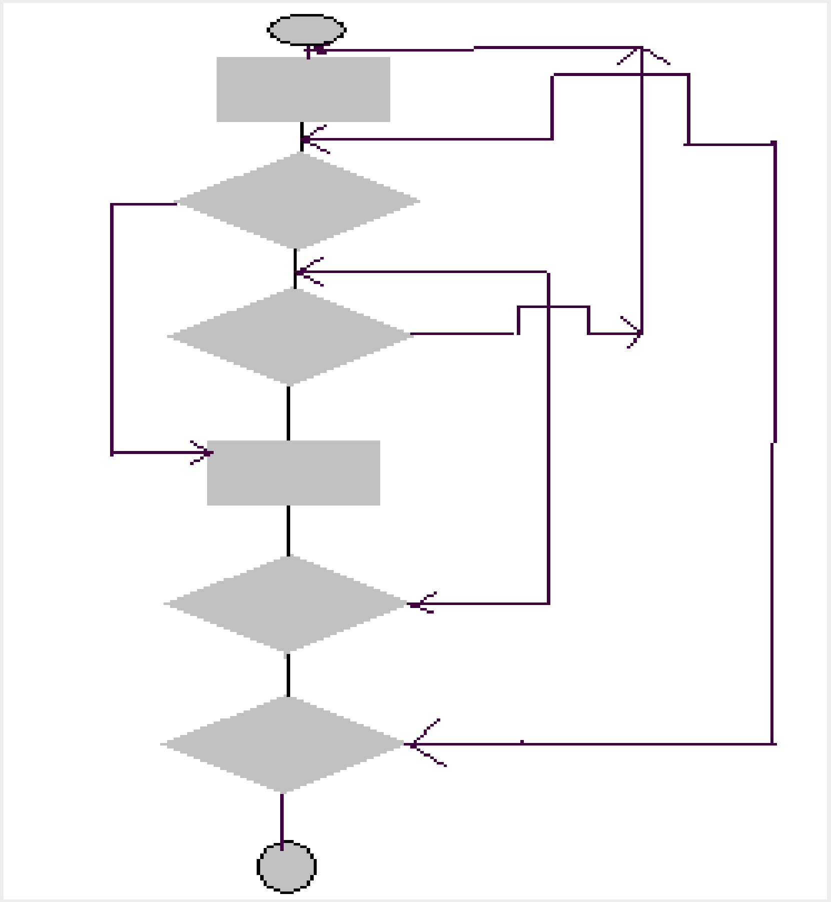
- If each loop is independent of the other, test them as simple loops, else test them as nested loops



Loop Testing

Spaghetti loops Testing Procedure:

- Redesign using structured constructs



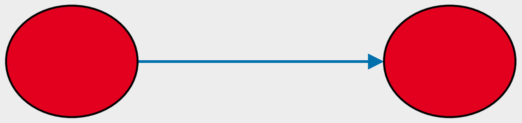
Flow Graph

- Main tool for test case identification
- Shows the relationship between program segments , which is the sequence of statements having the property that if the first member of the sequence is executed then all other statements in that sequence will also be executed

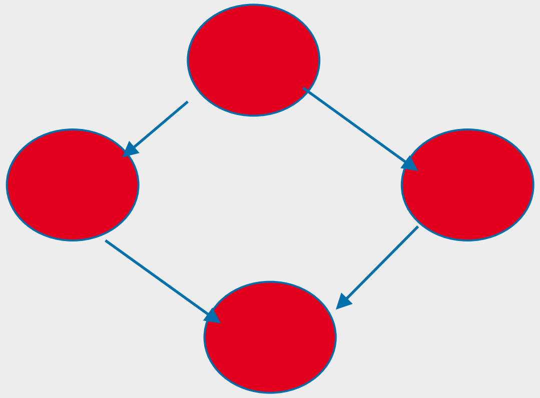
Flow Graph Symbols

- Nodes represent one program segment
- Areas bounded by edges and nodes are called regions
- An independent path is any path through the program that introduces at least one new set of processing statements or a new condition
- Each node containing a condition is called a predicate node

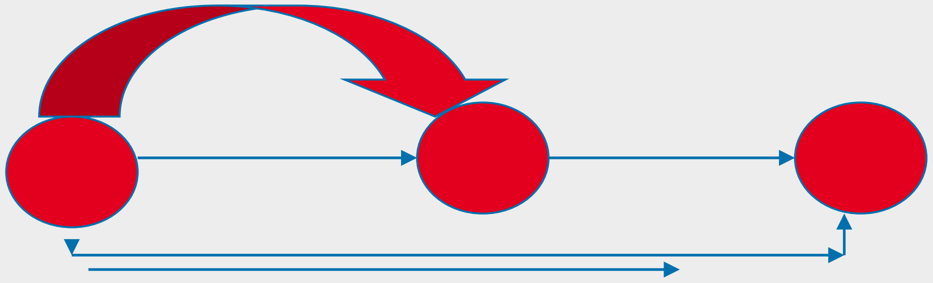
Flow Graph Symbols



Sequence



IF



WHILE

Cyclomatic Complexity

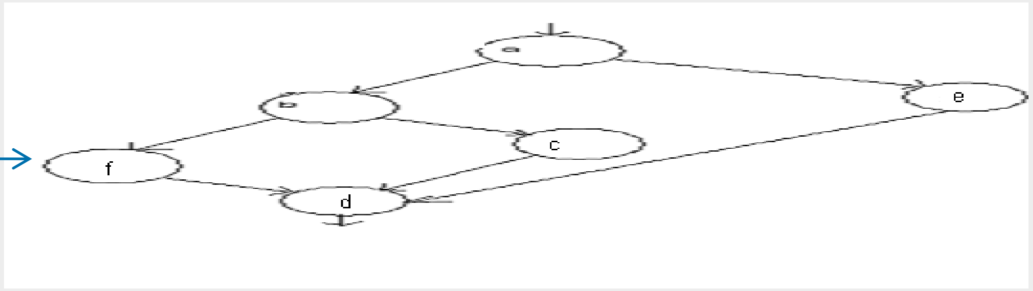
- Cyclomatic Complexity (Code Complexity) is a software metric that provides a quantitative measure of logical complexity of a program
- When Used in the context of the basis path testing method, value for cyclomatic complexity defines number of independent paths in basis set of a program
- Also provides an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once
- Cyclomatic complexity is often referred to simply as program complexity, or as McCabe's complexity

Calculating Cyclomatic Complexity

- The cyclomatic complexity of a software module is calculated from a flow graph of the module , when used in context of the basis path testing method
- Cyclomatic Complexity V(G) is calculated one of the three ways:
  - V(G) = E - N + 2 , where E is the number of edges and N = the number of nodes of the graph
  - V(G) = P+1, where P is the number of predicate nodes
  - V(G) = R , where number of region in the graph

Calculating Cyclomatic Complexity : Example

In the given figure a and b are predicate nodes



1. Cyclomatic Complexity, V(G) for a flow Graph G is V(G) = E – N + 2

E = Number of Edges in the graph (7 in the above figure)

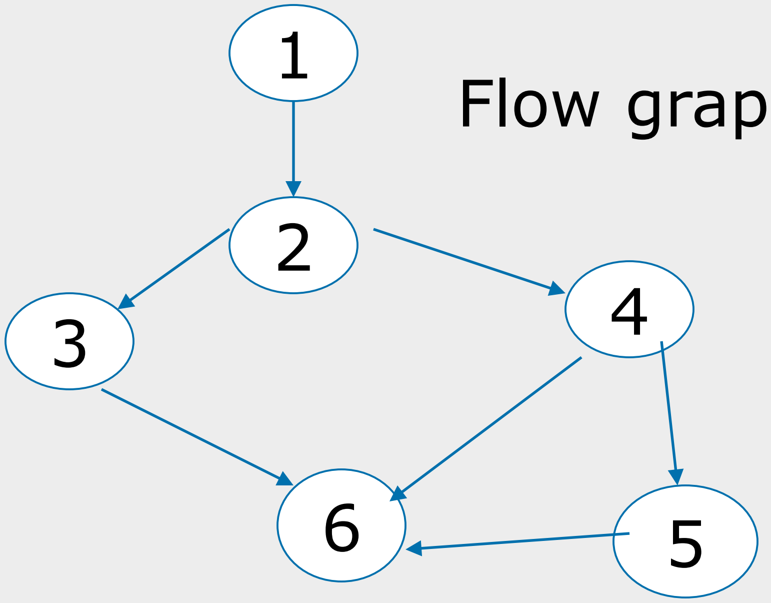
N = number of flow graph Nodes (6)

R = number of Regions (3)

Hence V(G) = 7-6+2 = 3

2. V(G) can also be calculated as V(G) = P+1, where P is the number of predicate nodes. Here V(G) = 2+1 = 3
3. Also V(G) can be calculated as V(G) = R hence V(G) = 3

Calculating Cyclomatic Complexity : Example



Flow graph

Edges(E) = 7 Nodes(N) = 6

Regions (R) = 3 , Predicate nodes (P) = 2

CC = E - N + 2 = 7- 6 + 2 = 3

CC = P + 1 = 2 + 1 = 3

CC = R = 3

Cyclomatic complexity gives minimum number of test cases to be performed to cover all the program statements.

Memory Leak

Memory leak is present whenever a program loses track of memory.

Memory leaks are most common types of defect and difficult to detect

Performance degradation or a deadlock condition occurs

Memory leak detection tools help to identify

- memory allocated but not deallocated
- uninitialized memory locations

Memory Leak

Find the error in the following snippet of code

```
void read_file(char*);
void test(bool flag)
{
      char* buf = new char[100];
       if (flag) {
                read_file(buf);
                delete [] buf;
                }
 }
```

Memory Fragmentation and Overwrites

Memory Fragmentation

- caused by frequent allocation and deallocation of memory
- can degrade an application's performance
- occurs when a large chunk of memory is divided into much smaller, scattered pieces
- May not be never allocated again

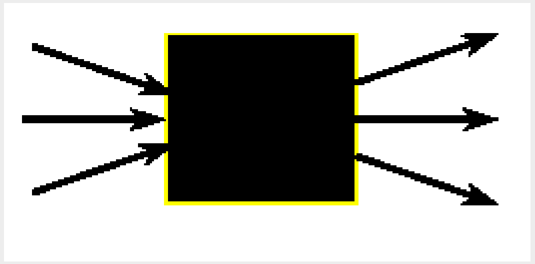
Memory Overwrites

- too little memory is allocated for an object
- can include memory corruption and intermittent failures.
- program may work correctly some times and fail at other times

Black Box Testing

- Black box is data-driven, or input/output-driven testing
- The Test Engineer is completely unconcerned about the internal behavior and structure of program
- Black box testing is also known as behavioral, functional, opaque-box and closed-box

Input



Outpu
t

Black Box Testing

Black Box Testing

- Tests are designed to answer the following questions:
- How is functional validity tested ?
- What classes of input will make good test cases?
- Is the system particularly sensitive to certain input values?
- What effect will specific combinations of data have on system operations?

Black Box(Specification-based) Testing Techniques

There are various techniques to perform Black box testing Techniques;

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table
- State transition testing

Equivalence Partitioning

This method divides the input domain of a program into categories of data for deriving test cases

Identify equivalence classes - the input ranges which are treated the same by the software

- Valid classes: legal input ranges
- Invalid classes: illegal or out of range input values

The aim is to group and minimize the number of test cases required to cover these input conditions

Assumption:

If one value in a group works, all will work

One from each partition is better than all from one

Thus it consists of two steps:

- Identify the Equivalence class
- Write test cases for each class

Equivalence Partitioning

Examples of types of equivalence classes

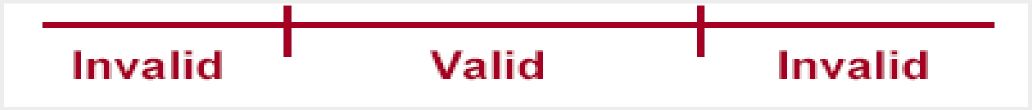
If an input condition specifies a continuous range of values, there is one valid class and two invalid classes

Example: The input variable is a mortgage applicant’s income. The valid range is $1000/mo. to $75,000/mo

- Valid class: {1000 > = income < = 75,000}
- Invalid classes: {income < 1000}, {income > 75,000}

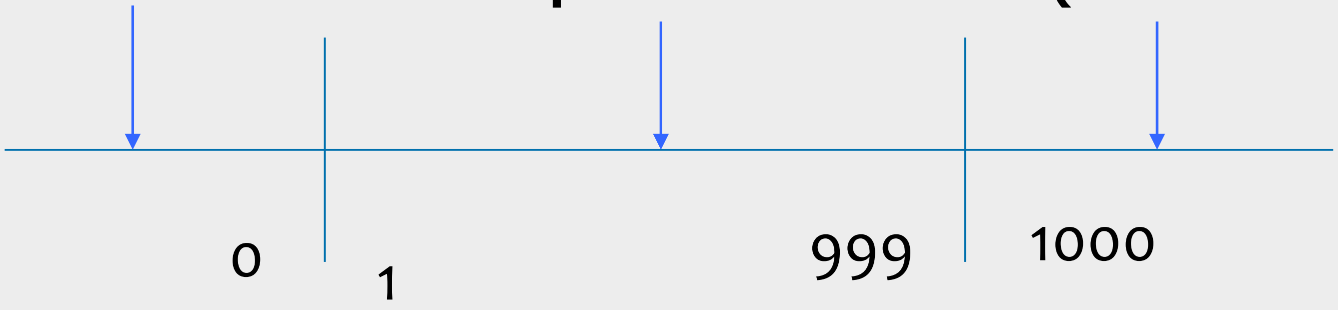
Equivalence Partitioning

If an input condition specifies that a variable, say count, can take range of values(1 - 999)



Identify - one valid equivalence class (1 < count < 999)

- two invalid equivalence classes (count < 1) & (count >999)



Equivalence Partitioning

If a “must be” condition is required, there is one valid equivalence class and one invalid class

Example: The mortgage applicant must be a person

- Valid class: {person}
- Invalid classes:{corporation, ...anything else...}

Boundary Value Analysis

"Bugs lurk in corners and congregate at boundaries ….." *Boris Beizer*

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

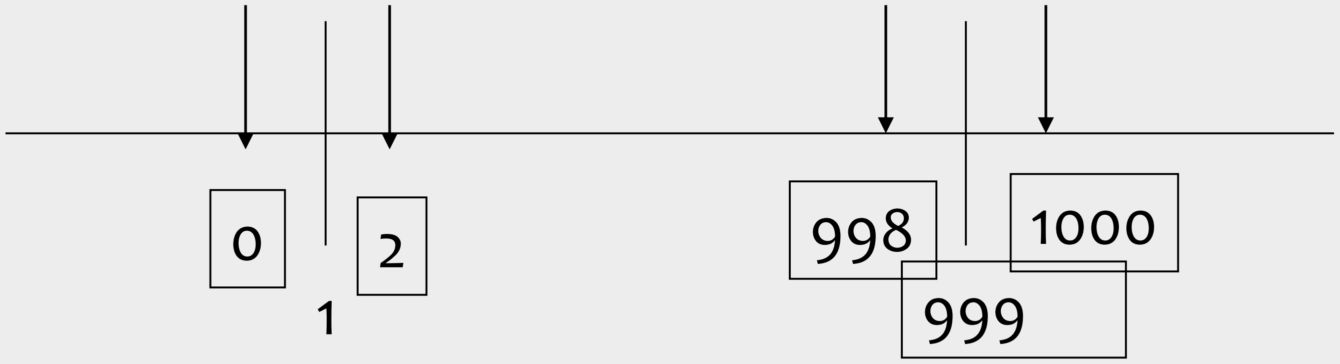
Boundary value analysis is a test case design technique that complements Equivalence partitioning.

Test cases at the boundary of each input Includes the values at the boundary, just below the boundary and just above the boundary.

Boundary Value Analysis

From previous example, we have the valid equivalence class as (1 < count < 999)

Now, according to boundary value analysis, we need to write test cases for count=0, count=1,count=2,count=998,count=999 and count=1000 respectively



Boundary Value Analysis

Guidelines

- If an input condition specifies a range of values A and B, test cases should be designed with values A and B, just above and just below A and B respectively
- Similarly with a number of values

Boundary Value Analysis

Example :

If we have to test function int Max(int a , int b) the Boundary Values for the arguments of the functions will be

Arguments	Valid Values	Invalid Values
A	-32768, -32767, 32767, 32766	-32769,32768
B	-32768, -32767, 32767, 32766	-32769,32768

Decision Tables

Decision Tables can be used when the outcome or the logic involved in the program is based on a set of decisions and rules which need to be followed.

A decision table lists the various decision variables, the conditions (or values) assumed by each of the decision variables and the actions taken in each combination or conditions.

Variables that contribute to the decision table are listed as the columns of the table

Last column of the table is the action to be taken for combination of values of the decision variables.

Decision Tables – Representation

A table listing all possible “conditions” (inputs) and all possible “actions” (outputs)

There is a “rule” for each possible combination of “conditions”

For each “condition”, it is identified as a “yes” (present), a “no” (not present) or an “X” for immaterial (the result is the same for either yes or no)

Considers all possible combinations

		Condition Entry			
Condition Stub		Rule - 1	Rule - 2	------	Rule - p
	Conditions				
	Condition - 1				
	Condition - 2				

	Condition - m				
Action Stub	**Actions**				
	Action - 1				
	Action - 2				

	Action - n				
		Action Entry			

Decision Tables - Approach

The steps for using Decision Table testing are as given below:

STEP 1: Analyze the given test inputs or requirements and list out the various conditions in the decision table.

STEP 2: Calculate the number of possible combinations.

STEP 3: Fill columns of the decision table with all possible combinations.

STEP 4: Find out cases where the values assumed by a variable are immaterial for a given combination. Fill the same by “don’t care” symbol.

STEP 5: For each of the combination of values, find out the action or expected result.

STEP 6: Create at least one Test case for each rule. If the rules are binary, a single test for each combination is probably sufficient. Else if a condition is a range of values, consider testing at both the low and high end of range.

Decision Tables – Example Problem

Engineering Examination Result for a student is based on the following conditions:

If the student has 80% attendance and has attended 3 internal tests with an average of 10 or more or has attended 2 internal tests with an average of 15 or more marks and has taken up the external examination and scored more than 35, then the student can be considered as pass in that subject.

Decision Tables – Solution

Based on the Decision Table Technique, the Decision table conditions and actions can be drafted as given below:

C1 = Condition 1 – Attendance > 80%

C2 = Condition 2 – Attended 3 internal Tests with an average of 10 or more marks

C3 = Condition 3 - Attended 2 internal Tests with an average of 15 or more marks

C4 = Condition 4 – Cleared External exam & scored more than 35 marks

A1 = Action 1 – Pass

A2 = Action 2 – Fail

R1 = Rule 1- C1, C2, C4 are satisfied

R2 = Rule 2 – C1, C3, C4 are satisfied

(Note: List the remaining rules - R3 to R16 accordingly)

The total number of alternatives = $2^{\text{NumberOfConditions.}} = 2^{4} = 16$

Decision Tables – Solution

Decision Table

		Condition Entry															
Condition Stub		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16
	Conditions																
	C1	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
	C2	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F
	C3		T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
	C4	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F	T
Action Stub	Actions																
	A1	T	T	T													
	A2				F	F	F	F	F	F	F	F	F	F	F	F	F
		Action Entry															

State Transition Testing

A testing techniques that aids to validate various states when an program moves from one visible state to another

It is a techniques in which test cases are designed to execute valid and invalid state transition

Excellent tool to capture certain types of system requirements and document internal system design.

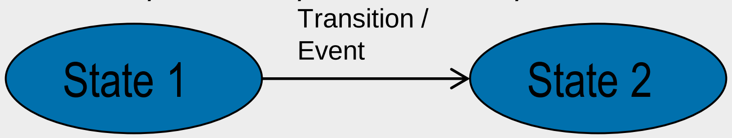
State Transition Based Testing - Approach

STEP 1:

- Understand the various states that the system, user, or object can be in, including the initial and final states.
- Examples of states can be: ‘User raising a purchase order’ or ‘leave request is accepted’. These states will be represented as:

STEP 2:

- Identify transitions, events, conditions, and actions that can - and can’t - apply in each state.



STEP 3:

- Use a graph or table to model the system. This graph or table also serves as an oracle to predict correct system behavior along with a requirements specification.

STEP 4:

- For each event and condition - that is, each transition - verify that the correct action and next state occurs.

STEP 5:

- Create test cases in such a way that all states are visited at least once, all events are triggered at least once and all paths are executed at least once (i.e. all transitions in the system are tested at least once)

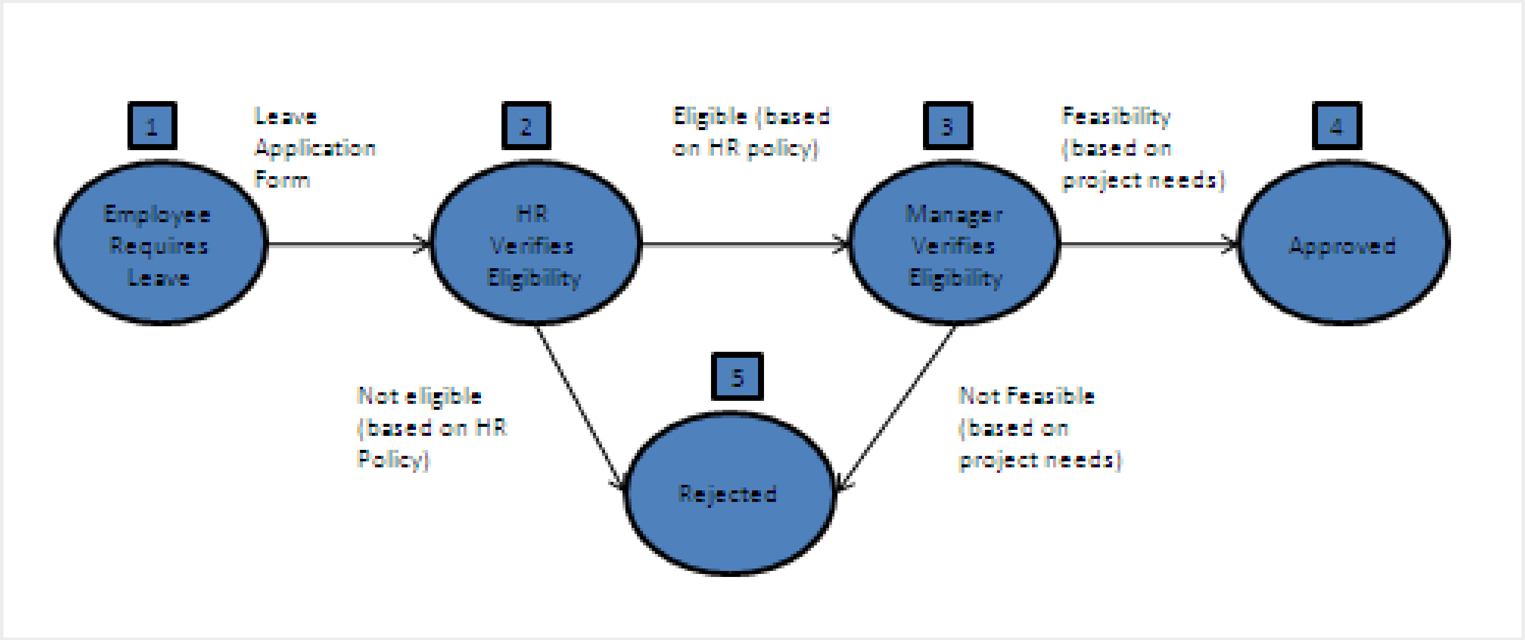
State Transition Based Testing – Example Problem

Consider a leave application system in an organization. An employee can raise a request for a leave, and if he is eligible for a leave(based on the number of days he has already taken etc), the application is sent to the manager for approval. The manager then validates and approves or rejects the leave based on the duration, reason for taking leave etc.

Now this problem can be represented in the form of a simple state transition graph

State Transition Based Testing – Solution

The state transition diagram for the given problem can be drawn as follows:



1
Employee Requires Leave
Leave Application Form
2
HR Verifies Eligibility
Eligible (based on HR policy)
3
Manager Verifies Eligibility
Feasibility (based on project needs)
4
Approved
Not eligible (based on HR Policy)
5
Rejected
Not Feasible (based on project needs)

The Independent Paths are

- 1 – 2 – 3 – 4
- 1 – 2 – 5
- 1 – 2 – 3 – 5

Hence, 3 test cases are required to test the given scenario

Experience Based Techniques - Error Guessing

- Based on experience and intuition one may add more test cases to those derived by following other methodologies
- It is an ad hoc approach
- The success of error guessing is very much dependent on the skill of the tester, as good testers know where the defects are most likely to lurk
- This is why error guessing testing technique should be used along with other formal testing techniques
- The basis behind this approach is in general people have the knack of “smelling out” errors
- There are no rules for error guessing
- The tester is encouraged to think of situations in which the software may not be able to cope.

Experience Based Techniques - Error Guessing

Make a list of possible errors or error-prone situations and then develop test cases based on the list

Defects' history are useful

Probability that defects that have been there in the past are the kind that are going to be there in the future

Some examples :

- Empty or null lists/strings
- Zero occurrences
- Blanks or null character in strings
- Negative numbers

Experience Based Techniques - Error Guessing

Example : Suppose we have to test the login screen of an application

An experienced test engineer may immediately see if the password typed in the password field can be copied to a text field which may cause a breach in the security of the application

Error guessing testing for sorting subroutine situations

- The input list empty
- The input list contains only one entry
- All entries in the list have the same value
- Already sorted input list

Experience Based Techniques - Exploratory Testing

Exploratory testing is minimum planning and maximum test execution

Also known as “Random” testing or “Ad-hoc” testing

Exploratory testing is simultaneous learning, test design, and test execution. (…James Bach)

How Exploratory Testing is useful:

1. When there are no or poor specifications
2. when time is severely limited
3. Helping to establish greater confidence in the software
4. Check on the formal test process by helping to ensure that the most serious defects have been found

Experience Based Techniques - Exploratory Testing - Tips

Test design Crafting

Careful Observation

Critical thinking

Diverse Ideas

Pooling resources (knowledge, learnings)

Choosing a Test Technique

The choice of which test techniques to use depends on a number of factors,

Internal Factors:

Models used

Tester knowledge/experience

Likely defects

Test objective

Documentation

Life cycle model

External Factors:

Level of Risk

Customer/contractual requirements

Type of system

Regulatory requirements

Time and budget

Static vs. Dynamic Testing

Static Testing	Dynamic Testing
It is the process of confirming whether the software meets its requirement specification	It is the process of confirming whether the software meets user requirements.
Examples : Inspections, walkthroughs and reviews	Examples : structural testing, black-box testing, integration testing, acceptance testing
It is the process of inspecting without executing on computer	It is the process of testing by executing on computer
It is conducted to prevent defects	It is conducted to correct the defects
It can be done before compilation	It takes place only after compilation and linking

Introduction – Test Case Construction & Test Data Preparation

Test cases construction and test data preparation are the first stages of testing

Test cases are prepared based on test ideas

“A test idea is a brief statement of something that should be tested. “

- For example, if you're testing a square root function, one idea for a test would be ‘test a number less than zero’

“ The idea of preparing a test case is to check if the code handles an error case.”

Test Case

Test Case is a set of inputs, execution preconditions, and expected outcomes developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.

In other words, Test Case is a planned sequence of actions (with the objective of finding errors)

Test cases may be designed based on -

- Values – Valid/Invalid/Boundary/Negative
- Test conditions

Test case will be complex if there is more than one expected result.

Test Case Terminologies

Pre Condition

- Environmental and state which must be fulfilled before the component/unit can be executed with a particular input value.

Test Analysis

- is a process for deriving test information by viewing the Test Basis
- For testing, test basis is used to derive what could be tested

Test basis includes whatever the test are based on such as System Requirement, technical specification, code or Business process

- A Technical specification
- The code itself (for structural testing)
- A business process

Test Condition

- It is a set of rules under which a tester will determine if a requirement is partially or fully satisfied
- One test condition will have multiple test cases

Test Case Terminologies (cont.)

Test Scenario

- It is an end-to-end flow of a combination of test conditions & test cases integrated in a logical sequence, covering a business processes
- This clearly states what needs to be tested
- One test condition will have multiple test cases

Test Procedure (Test Steps)

- A detailed description of steps to execute the test

Test Data/Input

- Inputs & its combinations/variables used

Expected Output

- This is the expected output for any test case or any scenario

Actual Output

- This is the actual result which occurs after executing the test case

Test Result/Status

- Pass / Fail – If the program works as given in the specification, it is said to Pass otherwise Fail.
- Failed test cases may lead to code rework

Other Terminologies

Test Suite – A set of individual test cases/scenarios that are executed as a package, in a particular sequence and to test a particular aspect

- E.g. Test Suite for a GUI or Test Suite for functionality

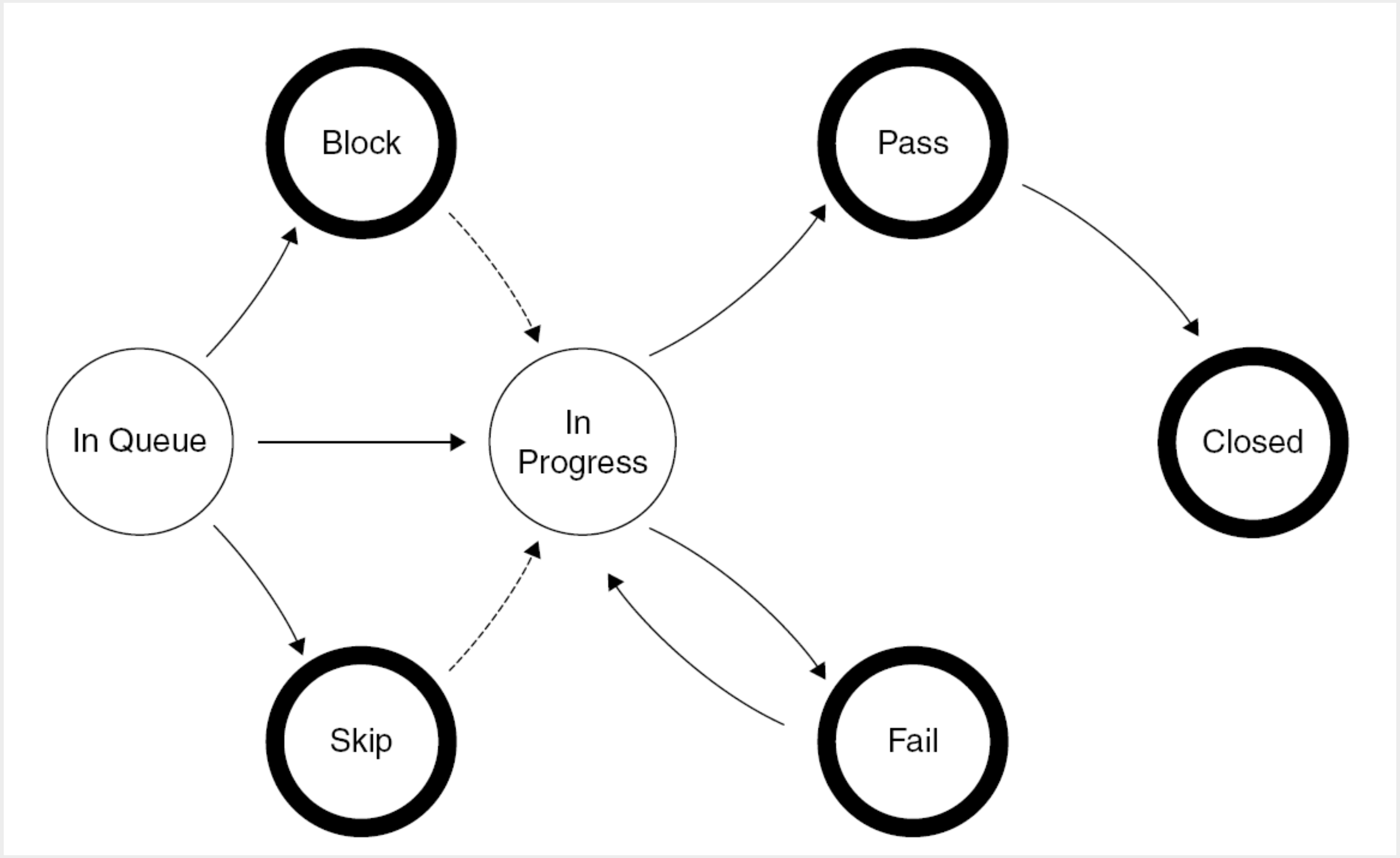
Test Cycle – A test cycle consists of a series of test suites which comprises a complete execution set from the initial setup to the test environment through reporting and clean up.

- E.g. Integration test cycle / regression test cycle

A good Test Case

Has a high probability of detecting error(s)
Test cases help us discover information
Maximize bug count
Help managers make ship / no-ship decisions
Minimize technical support costs
Assess conformance to specification
Verify correctness of the product
Minimize safety-related lawsuit risk
Find safe scenarios for use of the product
Assure quality

Test Case Lifecycle



Test Case Design Techniques

As Exhaustive testing is impractical, test case design techniques will help us to select test cases more intelligently

What is a Test Design Technique?

- A procedure for selecting or designing tests
- Based on a structural or functional model of the software
- Successful at finding faults
- Best' practice
- A way of deriving good test cases
- way of objectively measuring a test effort

Advantages of Test Design Techniques:

- Different people: similar probability find faults
- Effective testing: find more faults
- Efficient testing: find faults with less effort

Test Case Design Techniques (Cont.)

Test cases are designed based on the following techniques

- Static (non - execution)
  - Examination of documentation, source code listings, etc.
- Specification-based - Black Box testing techniques
  - Boundary value analysis, Equivalence partitioning, decision table
- Structure – based – White Box testing techniques
  - Code coverage, decision coverage, statement coverage
- Experience based techniques
  - Exploratory testing, fault attack, error guessing

What is test data?

Test Data

- An application is built for a business purpose. We input data and there is a corresponding output. While an application is being tested we need to use dummy data to simulate the business workflows. This is called test data.
- A test scenario will always have an associated test data.Tester may provide test data at the time of executing the test cases or application may pick the required input data from the predefined data locations.
- The test data may be any kind of input to application, any kind of file that is loaded by the application or entries read from the database tables. It may be in any format like xml test data, stand alone variables, SQL test data etc.

If you are testing with bad or unstable data, how can you be sure your test results are accurate!!!

Properties of Good Test Data

Realistic – accurate in context of real life

- E.g. Age of a student giving graduation exam is at least 18

Practically valid – data related to business logic

- E.g. Age of a student giving graduation exam is at least 18 says that 60 years is also valid input but practically the age of a graduate student cannot be 60

Cover varied scenarios

- E.g. Don't just consider the scenario of only regular students but also consider the irregular students, also the students who are giving a re-attempt, etc.

Exceptional data

- E.g. There may be few students who are physically handicapped must also be considered for attempting the exam

Test Data team

Test Data team should have Data Coordinator and team members. Test data teams are structured in various different ways.

- Dedicated test data team
- Development team as a test data team
- QA team as a test data team

Data Coordinator’s role and responsibility - Data coordinator will be the point of contact between the main stakeholders. He will be responsible for gathering all data requirements.

- Documentation of knowledge of interfaces and test data, mentoring and advising test team on data use, and support on the end-to-end flow;
- Data Coordinator will be responsible of the data prioritization, according to the timelines fixed by data team for executing and delivering the requested data.

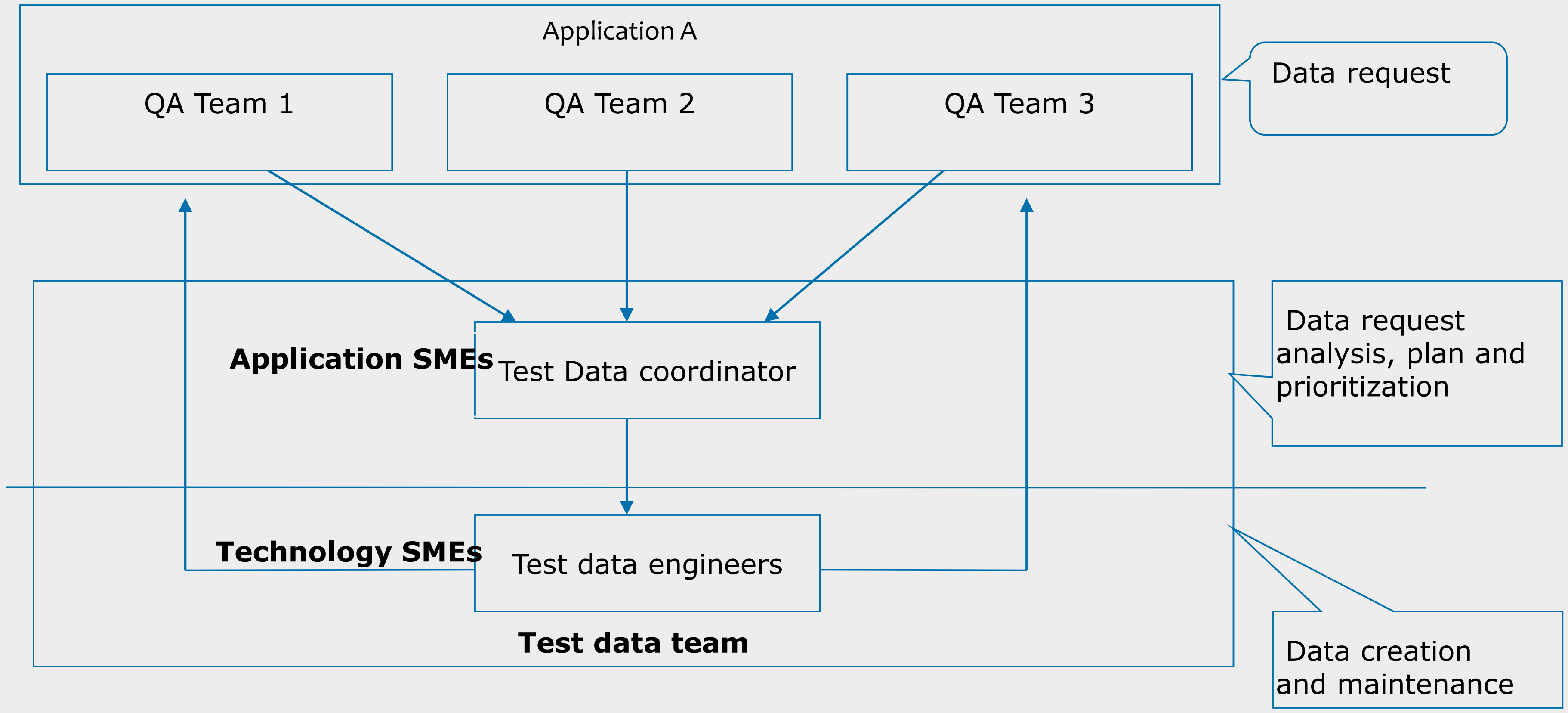
Test Data team (Cont.)

- Participation in test case planning, collection and providing data to support test cases for development and execution
- Providing help in handling, managing and manipulating test data
- Data Coordinator must ensure the correct application of masking rules, since each test case can have a set of static data necessary for execution

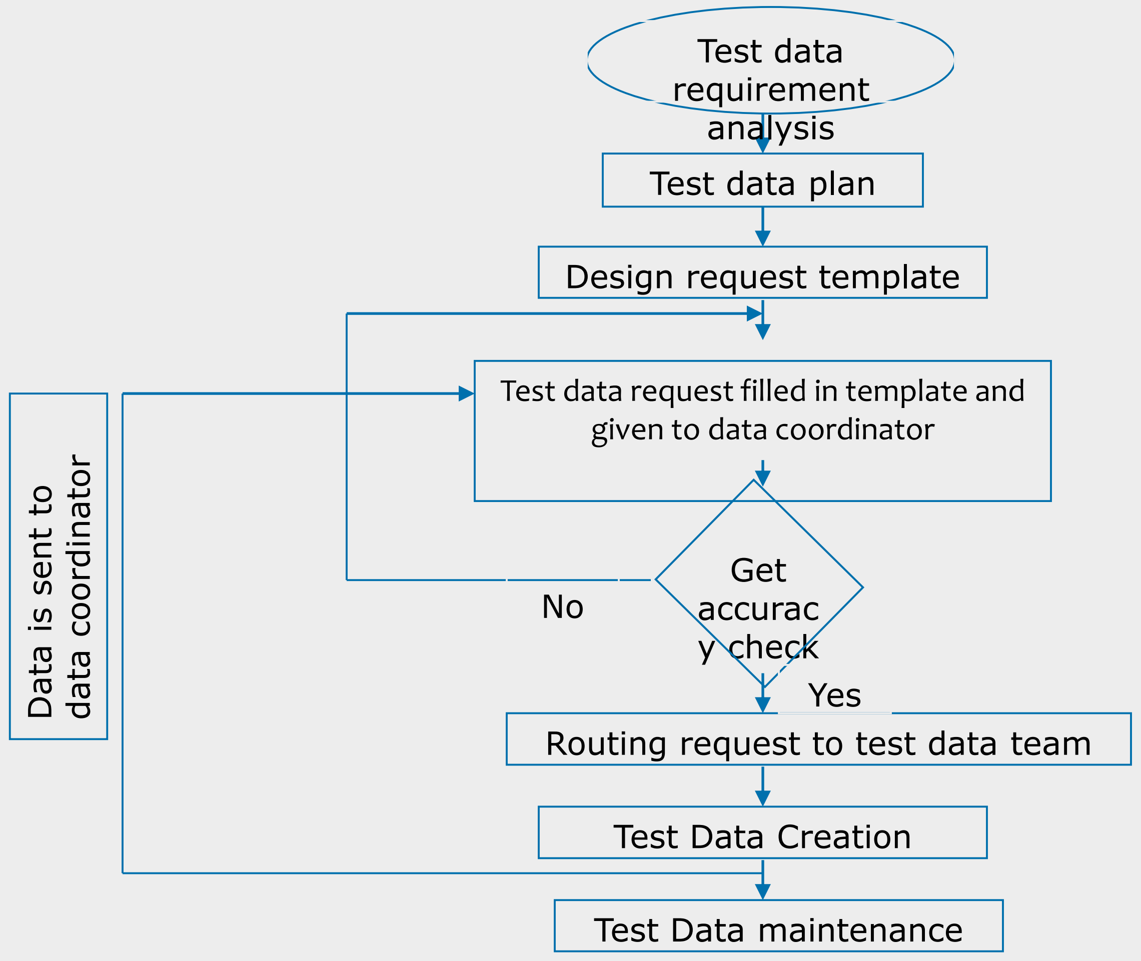
Test Data Engineer’s role and responsibility – Test data engineer will be responsible to create the data as per the requirement . He should be doing following activities during data creation

- Understand the Requirements
- Understand the DB and Table structures of the applications in case data generation for database testing
- Understand the volume of data required
- Automate the process of data generation if volume is huge

Test Data team (Cont.)



Test data lifecycle


Test data requirement analysis
Test data plan
Design request template
Test data request filled in template and given to data coordinator
Get accuracy check
No
Yes
Routing request to test data team
Test Data Creation
Test Data maintenance
Data is sent to data coordinator

What is Positive Testing?

Positive testing can be performed on the system by entering the valid data as input

When tester test the application from positive point of mind then it is known as positive testing

Testing aimed at showing software works

Also known as “test to pass” or “Happy path testing”

It is generally the first form of testing that a tester performs on an application

Example : Consider a scenario where you want to test an voting application which contains a simple textbox to enter age and requirement is that it should take only integers values and the value should be greater than 18.

Age: 19 Enter only integer values > 18 (Positive Testing)

Advantages/Limitations of positive testing

Advantages of positive testing

- Positive testing proves that a given product and project always meets the requirements and specifications
- Positive testing ensures that the business use case is validated

Limitations of Positive testing:

- Positive tests check for only valid set of values

What is negative testing?

The purpose of Negative testing is to break the system and to verify the application response for invalid inputs

This is to test the application that does not do anything that it is not supposed to do

When tester/User test the application from negative point of mind then it is known as negative testing

Testing aimed at showing software does not work. Also known as “test to fail”

Example of Negative testing:

- In the voting application scenario, Negative testing can be performed by testing by entering alphabets characters from A to Z or from a to z. Age text box should not accept the values or it should throw an error message for these invalid data inputs.

Age: ABC223 Enter only integer values > 18

Advantages/Limitations of negative testing

Advantages of Negative Testing:

- Negative testing helps to improve the testing coverage of your software application under test
- Negative testing discovers 'hidden' errors from application under test
- Negative testing help to find more defects & improve the quality of the software application under test
- negative testing ensures that the delivered software has no flaws

Limitation of Negative Testing

- Negative tests check for only invalid set of values

Positive & Negative test scenarios

Let’s take example of Positive testing scenarios:

- If the requirement is saying that password text field should accepts 5 – 15 characters and only alphanumeric characters.

Positive Test Scenarios:

- Password textbox should accept 5 characters
- Password textbox should accept up to 15 characters
- Password textbox should accepts any value in between 5-15 chars length
- Password textbox should accepts all numeric & alphabets values

Negative Test Scenarios:

- Password textbox should not accept less than 5 characters
- Password textbox should not exceeds more than 15 characters
- Password textbox should not accept special characters

What is Basic test?

Basic tests are used to test very basic functionality of software

The basic tests also verifies end to end builds

Basic test are always positive tests

Basic test can be smoke test or sanity test

Example on Basic test

Customer Relationship Management (CRM) application is business philosophy towards customers. To focus on their needs and improve customer relationships, with view to maximize customer satisfaction. So, in CRM application customer creation is basic functionality that should work. So the basic test focus is on Login and then customer creation. The basic test for this CRM application is Customer login and then customer creation with mandatory fields.

What is Alternate test?

Sometimes there maybe more than one way of performing a particular function or task with an intent to give the end user more flexibility or for general product consistency

This is called alternate testing

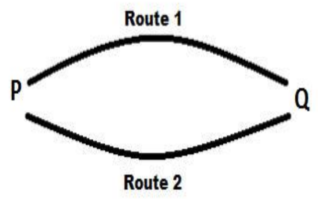
Alternate test is a kind of positive testing

In alternate path testing the test is again performed to meet its requirements but using different route than the obvious path

The test scenario would even consume the same kind of data to achieve the same result

Example on Alternate test

Alternate test



P is a starting point and Q is the end point. There are two ways to go from P to Q. Route 1 is the generally taken route and Route 2 is an alternative route. Therefore in such a case, happy path testing would be traversing from point P to Q using Route 1 and the alternate test would comprise taking Route 2 to go from P to Q. Observe that the result in both the cases is the same.

Importance of writing positive, negative, basic, alternate test while designing test cases

Approach of writing positive, negative, basic, alternate test are useful to design effective test cases which help to improve quality of software

These approach to test case design are help to improve the test case design coverage

By using these approach test cases are written for real life scenarios. It ensures real life scenarios are tested before moving software live

By designing positive and negative test cases ensures that the application works as per the requirements and specifications

By executing effective test cases, helps to find more defects before releasing software, so it builds confidence in system

Best practices for test case maintenance

Have Approved Test Case template in place

Identify the location of Test Cases storage with good access control

Have Test Case Review & Approval SOP in place

Appropriate Training for Testers for Test Case Authoring / Reviews / Executions / Maintenance

Test Cases attributes can be discussed and agreed upon:

- Should it contain Navigational OR click-by-click Test Steps
- Should it contain Test Data set up steps within Test Case or it should be done separately
- Test Case modification protocol
- Reusable components development

Best practices for test case maintenance

Audit Trail for every update in Test Case

Good management of Impact assessments with every update in requirement(s) w.r.t Test Cases coverage

Better management of Trace Matrix with “every” update

Maintenance of record of Executed Test Cases and Defects for reference of updates

It is advisable to store test cases in version control tool so that any subsequent changes can be tracked easily

Use test case creation and maintenance tools. One such tool is Quality Center from HP

Summary

In this lesson, you have learnt:

- The test case techniques discussed so far need to be combined to form overall strategy
- Each technique contributes a set of useful test cases, but none of them by itself contributes a thorough set of test cases

Review Question

Question 1: ______ testing can discover dead codes

Question 2: The objective of walkthrough is to find errors but not solutions

- Option: True / False

Question 3: For calculating cyclomatic complexity, flow graph is mapped into corresponding flow chart

- Option: True / False

Question 4: How many minimum test cases required to test a simple loop?

Question 5: Incorrect form of logic coverage is :

- Statement coverage
- Pole coverage
- Condition coverage
- Path coverage

Review Question

Question 6: One test condition will have ____________ test cases.

Question 7: For Agile development model conventional testing approach is followed.

- Option: True / False

Question 8: A test case is a set of _________, _____________, and ___________ developed for a particular objective.

Question 9: An input field takes the year of birth between 1900 and 2004. State the boundary values for testing this field.

- 0, 1900,2004,2005
- 1900, 2004
- 1899, 1900, 2004, 2005
- 1899, 1900, 1901, 2003, 2004, 2005

Review Question: Match the Following

1. Code coverage
2. Interface errors
3. Code complexity

A. Flow graph
B. Loop testing
C. Black box testing
D. Flow chart
E. Condition testing
F. White box testing