
بسم الله الرحمن الرحيم

Test Case Design Through Black Box

(Continue)

Boundary Value Analysis (BVA)

(Test Data)

Boundary Value Analysis (BVA)

- Greater number of errors tends to occur at the boundaries of the input domain than in the center
- Uses same principal
  - Inputs & Outputs grouped into Classes
- Elements are selected such that each edge of the E.C. is subject of a test (Boundaries are always a good place to look for defects)

Boundary Value Analysis (BVA)

- Boundaries mark the point or zone of transition from one equivalence class to another.
- The program is more likely to fail at a boundary, so these are the best members of (simple, numeric) equivalence classes to use.
- If software can operate on the edge of its capabilities, it will almost certainly operate well under normal conditions.

Boundary Value Analysis (BVA)

- B.V.A. focuses on testing of values from the boundary of the class.
- Design a test case for the boundary value.
- Design a test case for one significant value on either side of the boundary.

Boundary Value Analysis (BVA)

- **E.C.P. and B.V.A. can be used together.**
- **Input: range of values**
  - Test Cases (valid) for the ends of the range
  - Test Cases (invalid) for conditions just beyond the ends
- **Example of BVA**
  - Input (real number- Range): 0.0 - 90.0. i.e. (0.0, 0.1, 0.2, 0.3…90.0)
    - Test Cases
      - **Valid**
        - 0.0, 90.0,
      - Invalid
        - -1, 90.001

Boundary Value Analysis (BVA)

- **Input: number of values**
  - Test Cases (maximum and minimum number of values)
  - One beneath and beyond these values
- **Example of BVA**
  - Input (file can contain 1-255 records)
    - Test Cases
      - **Valid**
        - 1, 255
      - **Invalid**
        - 0, 256 records.

Boundary Value Analysis (BVA)

- **Types of Boundary conditions**
  - Numeric, position, quantity, speed, location, size
- **Also, extremes**
  - first/last, min/max, start/finish, over/under, empty/full,

    Shortest/longest, slowest/fastest, largest/smallest

Boundary Value Analysis (BVA)

- **Use these guidelines for each output condition**
  - **Output: Monthly Deduction**
    - Minimum = 0.0, Maximum = 3500.50
  - **Test Cases to cause**
    - Valid
      - 0.0 deduction and 3500.50 deduction
    - If possible to design test cases to have negative deduction and deduction larger than 3500.50.
    - Invalid
      - -1 deduction and 3500.51 deduction

Boundary Value Analysis

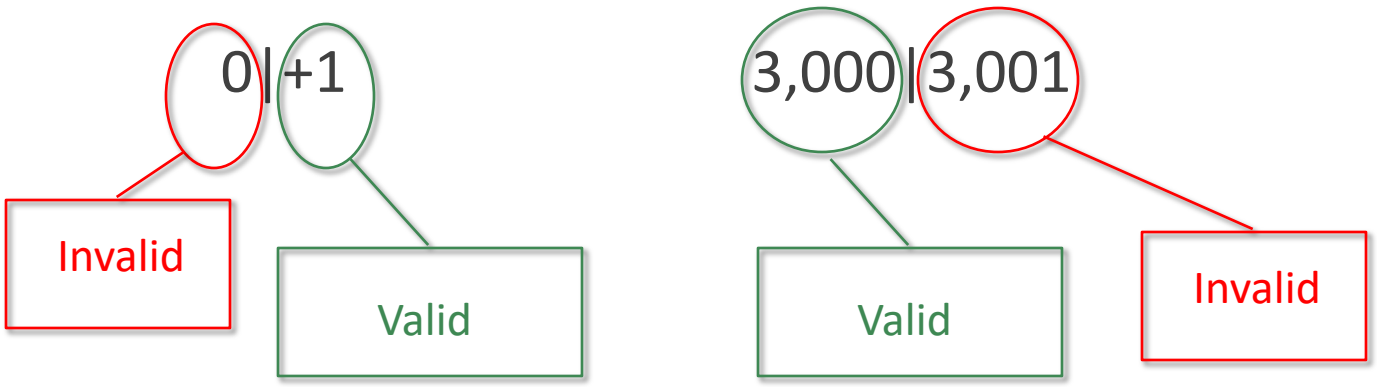
(Example)

Boundary Value Analysis (BVA)

- **Employees of an organization are allowed to get accommodation expenses while traveling on official tours.**
- **The program for validating expenses claims for accommodation has the following requirements**
  - There is an upper limit of Rs. 3,000 for accommodation expense claims
  - Any claim above Rs. 3,000 should be rejected and cause an error message to be displayed
  - All expense amounts should be greater than zero and an error message to be displayed if this is not the case

Boundary Value Analysis (BVA)

- **Inputs: Accommodation Expense**
- **Boundaries of the Input Values**
- **Better to show Boundaries Graphically**
  - Boundary: 0 < Expense ≤ 3,000



Boundary Value Analysis (BVA)

B.A. Example

Inputs

Test	Expenses	Boundary	Expected Output
1	-1	0	Error Msg
2	0	0	Error Msg
3	1	0	OK
4	2,900	3,000	OK
5	3,000	3,000	OK
6	3,001	3,000	Error Msg

Error Guessing

Error Guessing

- **Just 'guess' where the errors are ……**
- **Intuition and experience of tester**
- **Ad hoc, not really a technique**
- **Strategy:**
  - Make a list of possible errors or error-prone situations (often related to boundary conditions)
  - Write test cases based on this list

Error Guessing

- Most common error–prone situations (risk analysis).
- Try to identify critical parts of program (high risk sections).
- Parts with unclear specifications.
- Developed by junior programmer while he has problems.
- Complex specification and code.
- High-risk code will be more thoroughly tested.

Error Guessing

- **Defects' histories are useful**
- **Some items to try are:**
  - Empty or null lists/strings
  - Blanks or null characters in strings
  - Negative numbers
- **"Probability of errors remaining in the program is proportional to the number of errors that have been found so far"**

Black Box Testing

- **Black-box test data generation techniques.**
  - Equivalence partitioning
  - Boundary value analysis
  - Error guessing

Which one to Use ?

Black Box Testing

- **Which one to use ?**
  - None is complete
  - All are based on some kind of heuristics
  - They are complementary
  - **Always use a combination of methods**

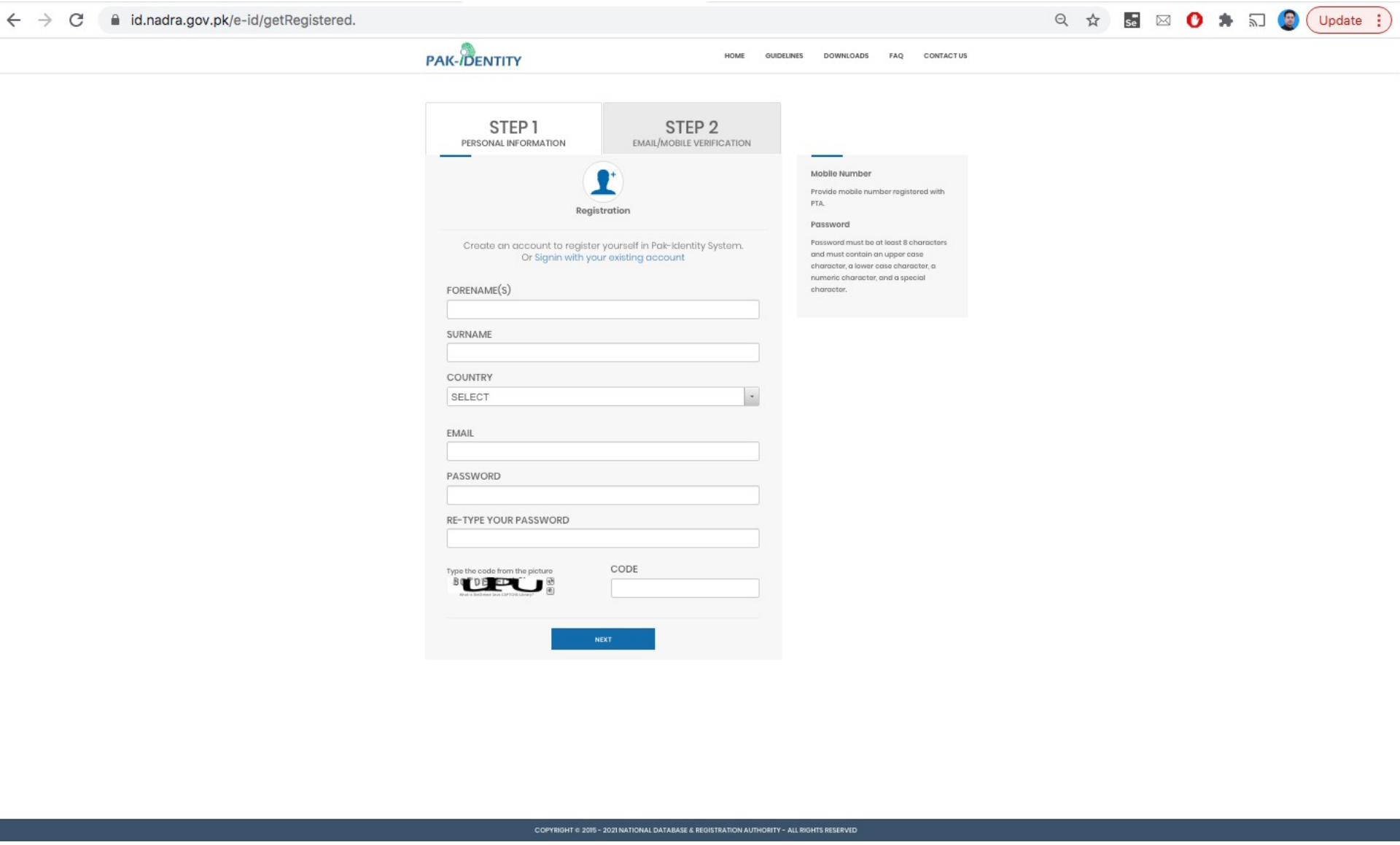
Practical Example of Testing

Practical Example

- **A person wants to "Create a New PAK IDENTITY Account" through NADRA online system i.e. *https://id.nadra.gov.pk/e-id/getRegistered*.**
- **There are multiple fields, which need to be filled for successful registration.**
  - **FR # 01: "Forename(s)"**
    - User shall be able to add only alphabets for their forename's. If user enters numeric then the system should show "*Invalid Characters*" error.
  - **FR # 02: "Surname"**
    - User shall be able to add only alphabets for their surname. If user enters numeric then the system should show "*Invalid Characters*" error.
  - **FR # 03: "Email"**
    - User shall be able to add only valid email id format. If user enters invalid email format then the system should give the "*Invalid Email Address!*" error.
  - **FR# 04: "Re-type Your Email"**
    - User shall be able enter same email id that he/she have entered before for validation. If email id is not matched with previous one then "*Email does not match*" error should be displayed.

Practical Example..Cont

- **FR # 05: "Primary Contact Number"**
  - For primary contact number user should select his/her country then should enter his/her phone #. User should not be able to enter alphabets in phone number. If user enters alphabets then system should give "*Only digits allowed and first character cannot be zero. Invalid Characters*" error. Along with that the first digit cannot be "0"
- **FR # 06: "Mobile Operators"**
  - User must select his/her mobile operator. E.g. ufone, mobilink etc. through radio button.
- **FR # 07: "Password"**
  - User should enter a strong password e.g. Alpha@642@211, and the system should indicate a weak, good & strong message for password strength. Password must be at least 8 characters long and must contain an upper case character, a lower case character, a numeric, and a special character i.e. !@#$%^&*(), if not, then system should give an error message i.e. "*Password must be at least 8 characters and must contain an upper case character, a lower case character, a numeric character, and a special character !@#$%^&*()*"
- **FR # 08: "Re-type Your Password"**
  - User should enter the same password as entered before or else system should give "*Password does not match*" error message
- **FR # 09: "captcha, agree, save & continue"**
  - User will then enter captcha, then will check on agree check box and then will click on "Save & Continue" button. If some thing is missing or invalid e.g. captcha the form will not continue.


id.nadra.gov.pk/e-id/getRegistered.
Update
PAK-IDENTITY
HOME
GUIDELINES
DOWNLOADS
FAQ
CONTACT US
STEP 1
PERSONAL INFORMATION
STEP 2
EMAIL/MOBILE VERIFICATION
Registration
Create an account to register yourself in Pak-Identity System.
Or Signin with your existing account
FORENAME(S)
SURNAME
COUNTRY
SELECT
EMAIL
PASSWORD
RE-TYPE YOUR PASSWORD
Type the code from the picture
CODE
NEXT
Mobile Number
Provide mobile number registered with PTA.
Password
Password must be at least 8 characters and must contain an upper case character, a lower case character, a numeric character, and a special character.
COPYRIGHT © 2015 - 2021 NATIONAL DATABASE & REGISTRATION AUTHORITY - ALL RIGHTS RESERVED

Practical Example

(Possible Solution)

Test Data FR # 01

- **FR # 01: “Forename(s)”**
  - User shall be able to add only alphabets for their forename's. If user enters numeric then the system should show “*Invalid Characters*” error.

Inputs: Forename

Partition the Input Values (ECP):

Valid:

{a,b,c,d,….z}

Invalid:

Outside valid

Test Case FR # 01

Test Case ID	1	2
Input	hasnain	123
Partition Tested	{a, b, c, d,….z}	Outside valid
Expected Output	Name Entered Successfully	*Invalid Characters*
Actual Output	“	“

Test Data FR # 02

- **FR # 02: "Surname"**
  - User shall be able to add only alphabets for their surname. If user enters numeric then the system should show "*Invalid Characters*" error.

Inputs: Surname

Partition the Input Values (ECP):

Valid:

{a,b,c,d,….z}

Invalid:

Outside valid

Test Case FR # 02

Test Case ID	**3**	**4**
Input	Abbas	@ABBAS
Partition Tested	{a, b, c, d,….z}	Outside valid
Expected Output	Name Entered Successfully	*Invalid Characters*
Actual Output	“	“

Test Data FR # 03

- **FR # 03: "Email"**
  - User shall be able to add only valid email id format. If user enters invalid email format then the system should give the "*Invalid Email Address!*" error.

Inputs: Email id

Partition the Input Values (ECP):

Valid

Valid Email address	Reason
email@domain.com	Valid email
firstname.lastname@domain.com	Email contains dot in the address field
email@subdomain.domain.com	Email contains dot with subdomain
firstname+lastname@domain.com	Plus sign is considered valid character
email@123.123.123.123	Domain is valid IP address
email@[123.123.123.123]	Square bracket around IP address is considered valid
"email"@domain.com	Quotes around email is considered valid
1234567890@domain.com	Digits in address are valid
email@domain-one.com	Dash in domain name is valid
_____@domain.com	Underscore in the address field is valid
email@domain.name	.name is valid Top Level Domain name
email@domain.co.jp	Dot in Top Level Domain name also considered valid (use co.jp as example here)
firstname-lastname@domain.com	Dash in address field is valid

In Valid

Invalid Email address	Reason
plainaddress	Missing @ sign and domain
#@%^%#$@#$@#.com	Garbage
@domain.com	Missing username
Joe Smith <email@domain.com>	Encoded html within email is invalid
email.domain.com	Missing @
email@domain@domain.com	Two @ sign
.email@domain.com	Leading dot in address is not allowed
email.@domain.com	Trailing dot in address is not allowed
email..email@domain.com	Multiple dots
あいうえお@domain.com	Unicode char as address
email@domain.com (Joe Smith)	Text followed email is not allowed
email@domain	Missing top level domain (.com/.net/.org/etc)
email@-domain.com	Leading dash in front of domain is invalid
email@domain.web	.web is not a valid top level domain
email@111.222.333.44444	Invalid IP format
email@domain..com	Multiple dot in the domain portion is invalid

Test Case FR # 03

Test Case ID	5	6
Input	hasnain@yahoo.com	Hasnain.yahoo.com
Partition Tested	Valid (as shown in image)	Outside valid (as shown in image)
Expected Output	Email Entered Successfully	*Invalid Email Address!*
Actual Output	“	“

Test Data FR # 05

- **FR # 05: “Primary Contact Number”**
  - For primary contact number user should select his/her country then should enter his/her phone #. User should not be able to enter alphabets in phone number. If user enters alphabets then system should give "*Only digits allowed and first character cannot be zero. Invalid Characters*” error. Along with that the first digit cannot be “0”.

Inputs: Primary Contact Number

Partition the Input Values (BVA):

Valid:

{0,1,2,3,4,5,6,7,8,9}

Digit length $\leq$ 12

First Digit $\neq$ 0

Invalid:

Outside valid

Test Case FR # 05

Test Case ID	7	8
Input	+92331765456	+92asdhasdh
Partition Tested	{0,1,2,3,4,5,6,7,8,9} Digit length < 12 First Digit = 0	Outside valid
Expected Output	Email Entered Successfully	*Only digits allowed and first character cannot be zero. Invalid Characters*
Actual Output	“	“

Test Data FR # 07

- **FR # 07: "Password"**
  - User should enter a strong password e.g. hasnain@896@211, and the system should indicate a weak, good & strong message for password strength. Password must be at least 8 characters long and must contain an upper case character, a lower case character, a numeric, and a special character i.e. !@#$%^&*(), if not, then system should give an error message i.e. "*Password must be at least 8 characters and must contain an upper case character, a lower case character, a numeric character, and a special character !@#$%^&*()*".
- Inputs: Password

Partition the Input Values (ECP):

Valid:

characters Length ≥8
{0, 1,2, 3 ,4, 5,6 ,7 ,8 ,9}
Minimum 1 upper case character
Minimum 1 lower case character
Minimum 1 special character (!@#$%^&*()

Invalid:

Outside valid

Test Case FR # 05

Test Case ID	9	10
Input	Hasnain@2018	hasnain
Partition Tested	characters>=8 1 upper case character 1 lower case character 1 special character (!@#$%^&*()	Outside valid
Expected Output	Password Entered Successfully	*Password must be at least 8 characters and must contain an upper case character, a lower case character, a numeric character, and a special character !@#$%^&*(*
Actual Output	“	“

Test Data Retype Password

Inputs: Re Type Password

Partition the Input Values (ECP):

Valid:

Same as Password

Invalid:

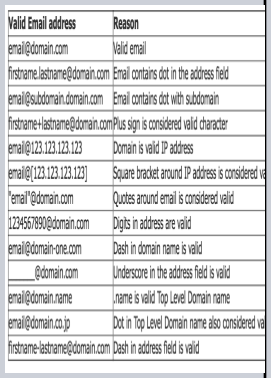
Outside valid

Test Case

Test Case ID	22	23
Input	“Re Type Password”	“Type Mismatch Password”
Partition Tested	Same as password	Outside valid
Expected Output	Password Entered Successfully	*Passwords do not match*
Actual Output	“	“

How To Represent Multiple Test Cases Which Are Dependent On Each Other ?

Test Case (Valid)

Test Case ID	11				
Input	Hasnain	Abbas	hasnain@yahoo.com	+92331765456	Home@2018
Partition Tested	{a, b, c, d,….z}	{a, b, c, d,….z}		{0,1,2,3,4,5,6,7,8,9}	characters>=8 1 upper case character 1 lower case character 1 special character (!@#$%^&*()
Expected Output	No error	No error	No error	No error	Data Entered Successfully
Actual Output	“	“	“	“	“

Valid Email address	Reason
email@domain.com	Valid email
firstname.lastname@domain.com	Email contains dot in the address field
email@subdomain.domain.com	Email contains dot with subdomain
firstname+lastname@domain.com	Plus sign is considered valid character
email@123.123.123.123	Domain is valid IP address
email@[123.123.123.123]	Square bracket around IP address is considered va
"email"@domain.com	Quotes around email is considered valid
1234567890@domain.com	Digits in address are valid
email@domain-one.com	Dash in domain name is valid
_____@domain.com	Underscore in the address field is valid
email@domain.name	.name is valid Top Level Domain name
email@domain.co.jp	Dot in Top Level Domain name also considered va
firstname-lastname@domain.com	Dash in address field is valid

Test Case (Invalid)

Test Case ID	12				
Input	Hasnain123	Abbas123	Hasnain.abbas.com	+92317hasnain	hasnain
Partition Tested	Outside Valid	Outside Valid	Outside Valid	Outside Valid	Outside Valid
Expected Output	*Invalid Characters*	*Invalid Characters*	*Invalid Email Address!*	*Only digits allowed and first character cannot be zero. Invalid Characters*	*Password must be at least 8 characters and must contain an upper case character, a lower case character, a numeric character, and a special character !@#$%^&*(*
Actual Output	“	“	“	“	“

Test Case (Valid & Invalid)

Test Case ID	13						
Input	Hasnain123	Abbas123	Hasnain	Hasnain.abbas.com	+923317654 56	+92317hasnain	hasnain
Partition Tested	Outside Valid	Outside Valid	{a, b, c, d,….z}	Outside Valid	{0,1,2,3,4,5,6,7,8,9}	Outside Valid	Outside Valid
Expected Output	*Invalid Characters*	*Invalid Characters*	No error	*Invalid Email Address!*	No error	*Only digits allowed and first character cannot be zero. Invalid Characters*	*Password must be at least 8 characters and must contain an upper case character, a lower case character, a numeric character, and a special character !@#$%^&*(*
Actual Output	“	“	“	“	“	“	“

Lets Test These Test Cases Manually

Test Case (Valid)

Test Case ID	11				
Input	Hasnain (FORENAME(S))	Abbas (SURNAME)	hasnain@yahoo.com (Email)	+92331765456 (Phone)	Home@2018 (PASSWORD)
Partition Tested	{a, b, c, d,….z}	{a, b, c, d,….z}		{0,1,2,3,4,5,6,7,8,9}	characters>=8 1 upper case character 1 lower case character 1 special character (!@#$%^&*()
Expected Output	No error	No error	No error	No error	Data Entered Successfully
Actual Output	“	“	“	“	“
P/F (Pass/Fail)					

Valid Email address	Reason
email@domain.com	Valid email
firstname.lastname@domain.com	Email contains dot in the address field
email@subdomain.domain.com	Email contains dot with subdomain
firstname+lastname@domain.com	Plus sign is considered valid character
email@123.123.123.123	Domain is valid IP address
email@[123.123.123.123]	Square bracket around IP address is considered va
"email"@domain.com	Quotes around email is considered valid
1234567890@domain.com	Digits in address are valid
email@domain-one.com	Dash in domain name is valid
_____@domain.com	Underscore in the address field is valid
email@domain.name	.name is valid Top Level Domain name
email@domain.co.jp	Dot in Top Level Domain name also considered val
firstname-lastname@domain.com	Dash in address field is valid

Test Case (In Valid)

Test Case ID	11				
Input	Hasnain123 (FORENAME(S))	Abbas123 (SURNAME)	Hasnain.abbas.com (EMAIL)	+92317hasnain	Hasnain (PASSWORD)
Partition Tested	Outside Valid	Outside Valid	Outside Valid	Outside Valid	Outside Valid
Expected Output	*Invalid Characters*	*Invalid Characters*	*Invalid Email Address!*	*Only digits allowed and first character cannot be zero. Invalid Characters*	*Password must be at least 8 characters and must contain an upper case character, a lower case character, a numeric character, and a special character !@#$%^&*(*
Actual Output					
P/F (Pass/Fail)					

Testing Automation Tools

(Selenium IDE & Katalon)

Selenium - History

- **Developed in 2004 by Jason Huggins as a JavaScript library used to automate his manual testing routines.**

- **In 2008, Selenium and WebDriver merged technologies and intellectual intelligence to provide the best possible test automation framework.**

Selenium IDE

- **Open source record and playback test automation for the web.**
- **Selenium IDE is a portable framework for testing web applications.**
- **Selenium IDE is a Chrome and Firefox plugin which records and plays back user interactions with the browser.**

Selenium IDE

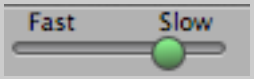
- Has a recording feature that records a user's live actions that can be exported in one of many programming languages.

- Use this to either create simple scripts.

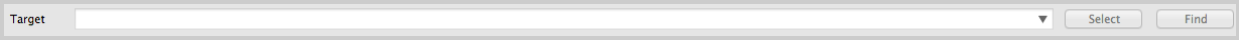
- Selenium can be extended through the use of plugins.

- Only for web.

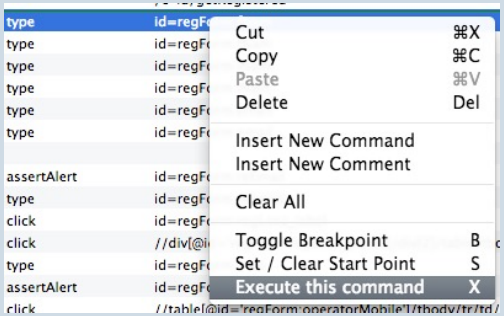
Selenium IDE – Basics

Play Entire Test Suite	
Play Current Test Case	
Pause/Resume	
Recording	
Value	
Test Case/Suite Speed	

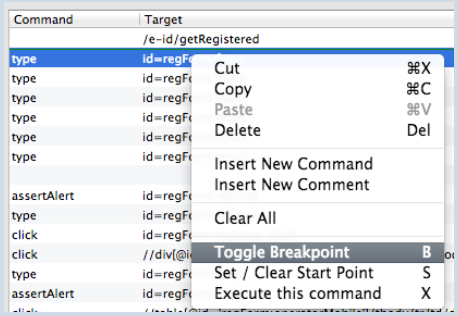
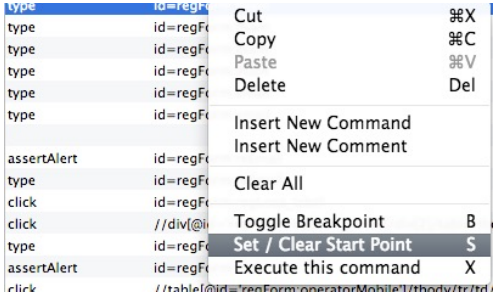
Selenium IDE – Basics

Command	
Target	

Selenium IDE – Commands (Examples)

Command	Functionality
Execute This Command (x) 	Execute This Specific Command Only
captureScreenshot	The screenshot is displayed in the "Screenshot tab". From there, you can export it.

Selenium IDE – Commands (Examples)

Command	Functionality
Toggle Breakpoint (b) 	Stop Here
Set/Clear Start Point (s) 	Start From Here

Selenium IDE – Commands

- **Selenium IDE Commands (pdf)**

Lets Test These Test Cases Through Automation

(Selenium IDE)


بسم الله الرحمن الرحيم

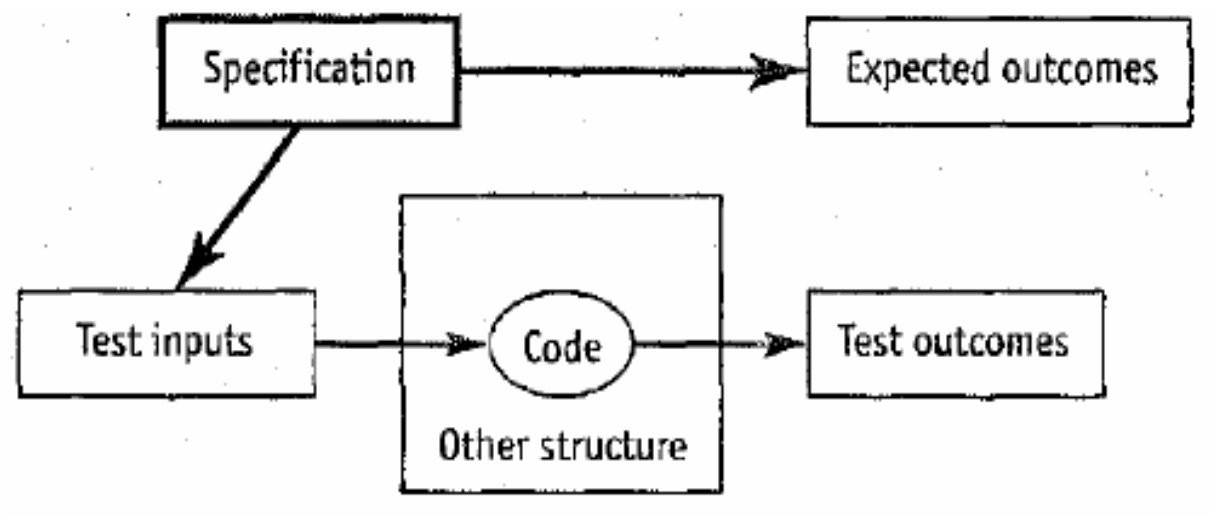
Test Case Design Through *White Box*

White Box Testing

- White box testing is also know as Glass Testing or Open Box testing. It is detailed examination of internal structure and logic of code.

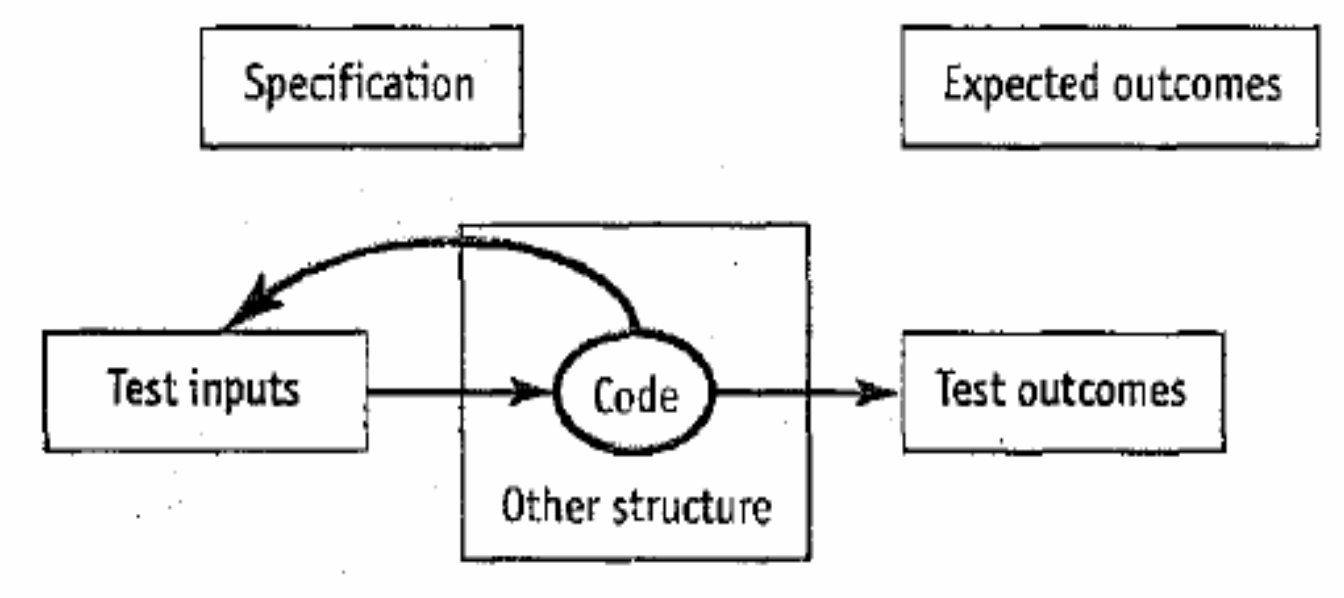
- This examination is demonstrated through test cases creation by looking at the code to detect any potential failure scenarios.

Test Case Design



Black Box Testing

Test Case Design based upon Source Code



White Box Testing

Challenges in Test Case Designing based upon Source Code

- This approach generates test inputs, but a test also requires expected outcomes to compare against.

- Code-based test case design cannot tell whether the outcomes produced by the software are the correct values, it is the specification for the code that tells you what it should do.

- So this approach is incomplete, since it cannot generate expected outcomes.

Challenges in Test Case Designing based upon Source Code

- Another problem with this approach is that it only tests code that exists and cannot identify code that is missing.
- It is also testing that 'the code does what the code does.'
- This is generally not the most useful kind of testing, since we normally want to test that ‘the code does what it should do.’
  - Software works as coded' (not 'as specified').

White Box Testing Test Case Basic Templae

Test ID	Input	Output	Pass/Fail

White Box Testing

- White box testing utilizes the logical flow through a program to propose test cases.
- Logical flow means the way in which certain parts of a program may be executed as we run the program.
- Logical flow of a program can be represented by a Control Flow Graph (CFG).
- Most common white box testing methods are:
  - Statement Coverage
  - Decision/Branch Coverage
  - Condition Coverage
  - Path Coverage

Control Flow Graph
(CFG)

Control Flow Graph (CFG)

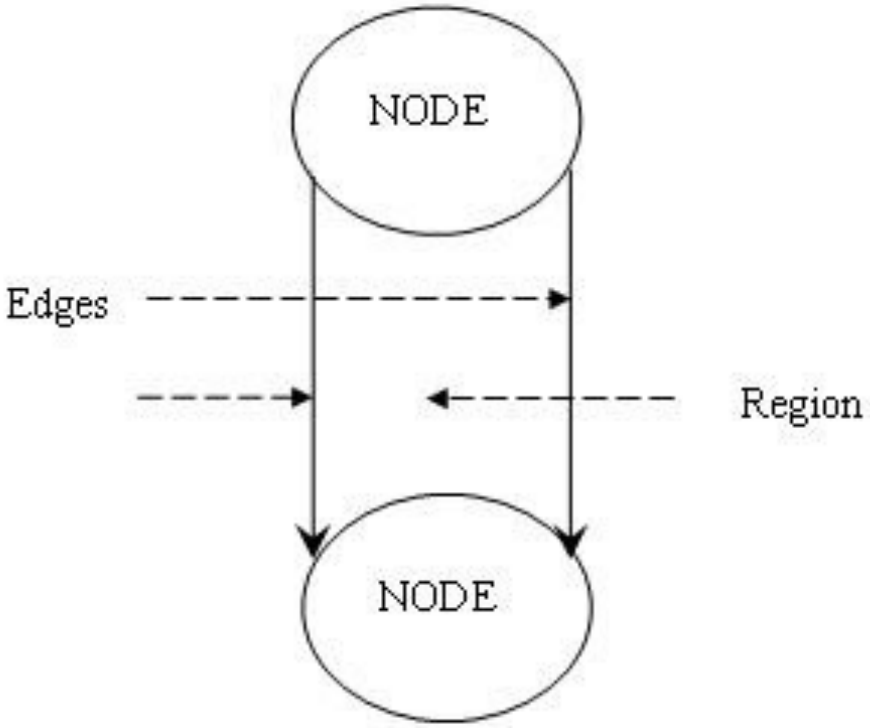
- The entire structure design and code of the software have to be studied for this type of testing.
- This method is implemented with the intention to test logic of the code so that the required results or functionalities can be achieved.
- Its applicability is mostly to relatively small programs or segments of larger programs, thus it is mostly used for unit testing.
- It catches 50% of all the bugs caught during unit testing. That amounts to 33% of all the bugs caught in the program.

Control Flow Graph

```
9  if ( $Number == 3 ) {
10
11 echo "Display Something, if the Number is 3";
12
13 } else {
14
15                ,,,,,
16
17 }
```

Failure

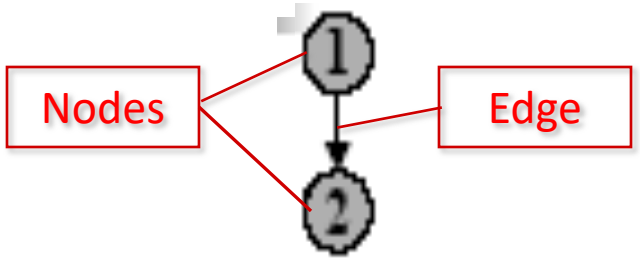
Control Flow Graph



Control Flow Graph (Example 1)

How to Draw a Control Flow Graph

- Number all the statements of a program



Sequence

1. a = 5;
2. b = a * b – 4;

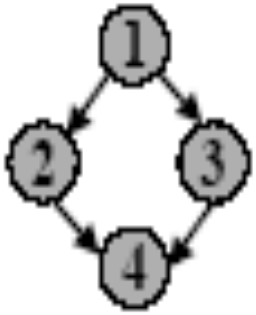
Numbered statements:

- Represent nodes of the flow graph

An edge from one node to another node exists:

- If execution of the statement representing the first node can result in transfer of control to the other node

Control Flow Graph (Example 2)



Selection

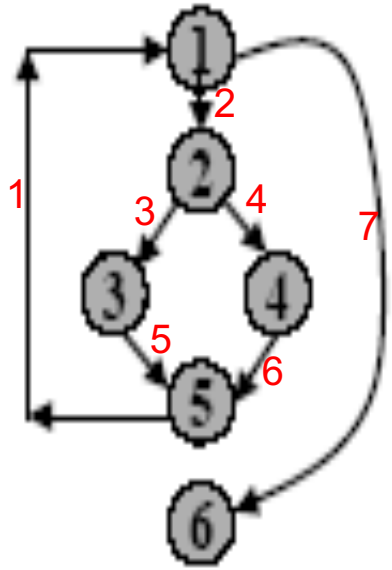
1. If (a > b)
2. c = 3;
3. Else c = 5;
4. c = c*c;

How many edges = 4

How many nodes = 4

Control Flow Graph (Example 3)

Flow Graph




```
int f1(int x, int y){
1.  while (x != y){
2.      if (x>y) then
3.          x=x-y;
4.      else y=y-x;
5.  }
6.  return x;
```

How many edges = 7

How many nodes = 6

How Much Percentage of Coverage ?

- **Statement Coverage=** $\frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$
- **Decision Coverage=** $\frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100$
- **Branch Coverage=** $\frac{\text{Number of executed branches}}{\text{Total number of branches}} \times 100$
- **Condition Coverage=** $\frac{\text{Number of executed operands}}{\text{Total number of operands}} \times 100$

Example
(CFG)

CFG For The Following Pseudocode Code

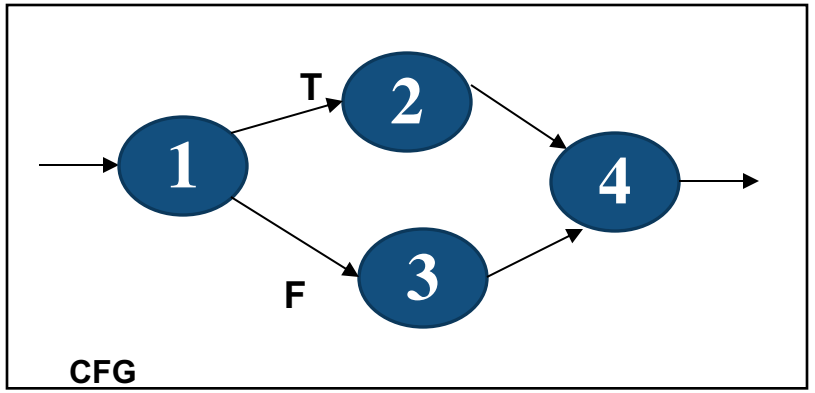
```
if X > 0 then
   Statement1;
else
   Statement2;
```

```
while X < 10 {
   Statement1;
   X++; }
```

Possible Solution

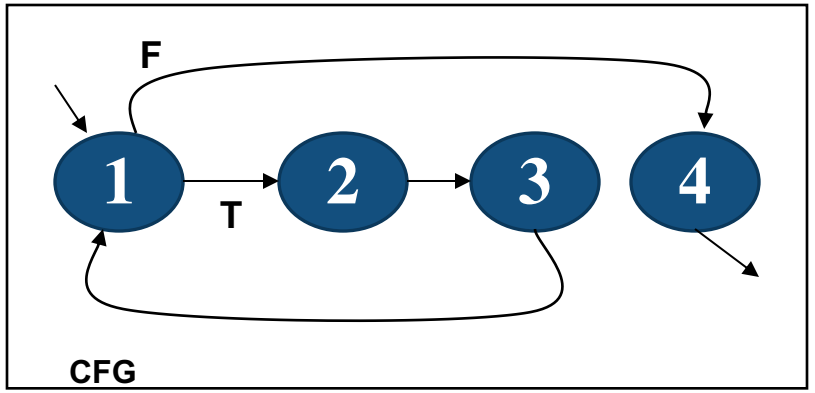
```
if X > 0 then
   Statement1;
else
   Statement2;
end
```

1
2
3
4




```
while X < 10 {
   Statement1;
   X++; }
end
```

1
2
3
4



We Can Also Use Flow Chart To

Understand The Logic of Code

Statement Coverage

Statement Coverage

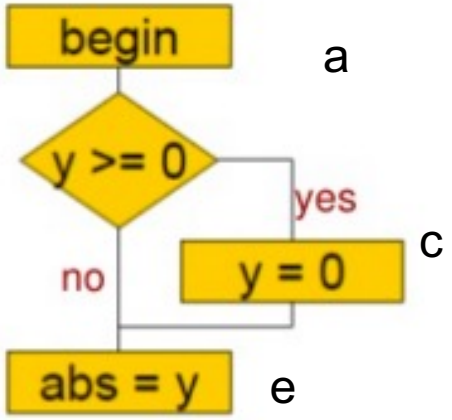
- The goal is to design test cases so that every statement of the program is executed at least once

- Code that has not been executed during testing is more likely to contain errors
  - Often this is the “low-probability” code

Statement Coverage

- Exercise all statements at least once

```
1. Begin
2. if (y >= 0)
3.       then y = 0;
4. abs = y;
5. end;
```



Test Case ?

Input: y= ?

Test ID	Input	Output	Pass/Fail
WBT 2(SC)	Y=o	Path (ace)	Pass

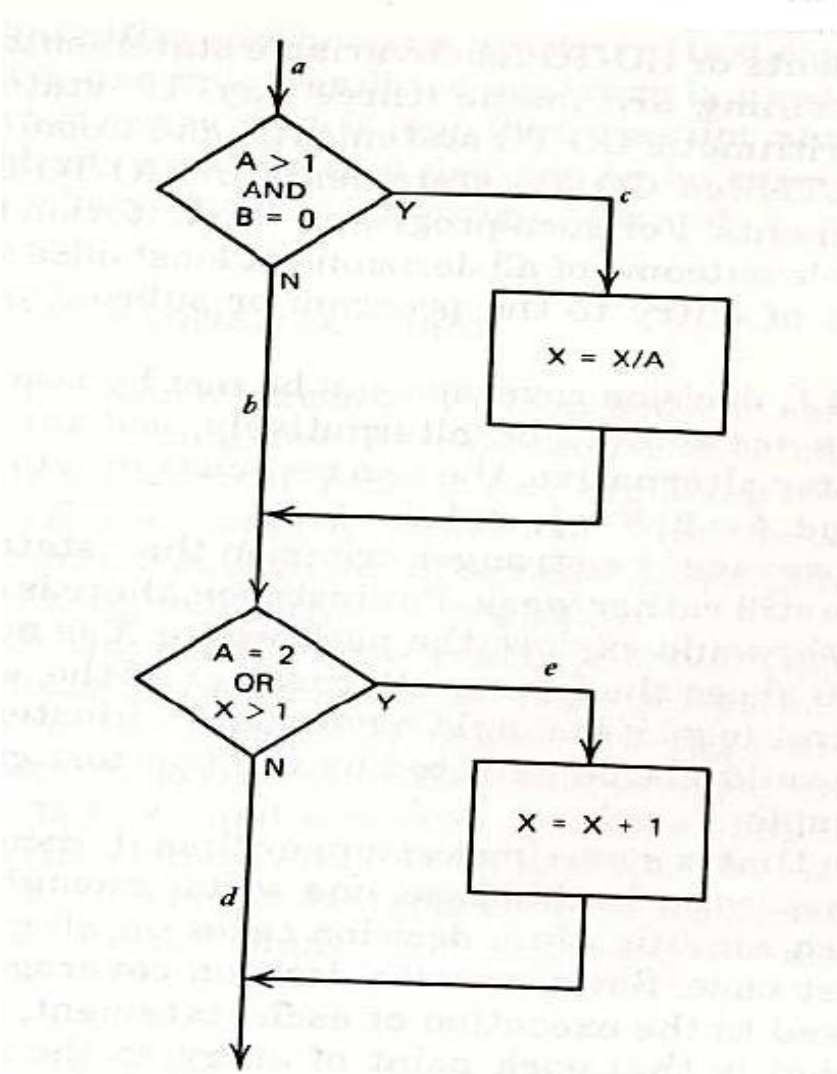
100% statement coverage

Statement Coverage

Test case:

A=2 and B=0 (ace)

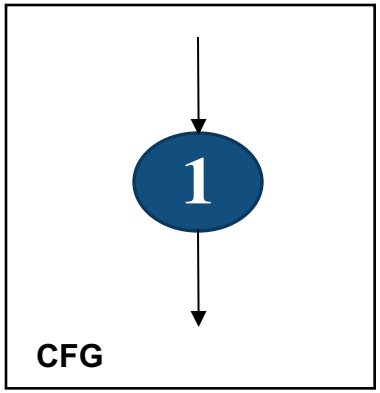
Test ID	Input	Output	Pass/Fail
WBT 1 (SC)	A=2 and B=0	Path (ace)	Pass



Multiple Statements

```
Statement1;
Statement2;
Statement3;
Statement4;
```

Can be represented as **one** node as there is no branch.



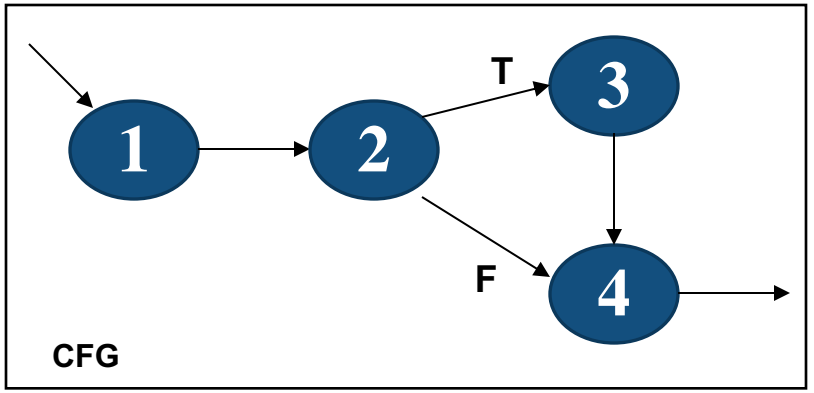

```
Statement1;
Statement2;

if X < 10 then
   Statement3;

Statement4;
```

1

2

3

4



Branch Coverage

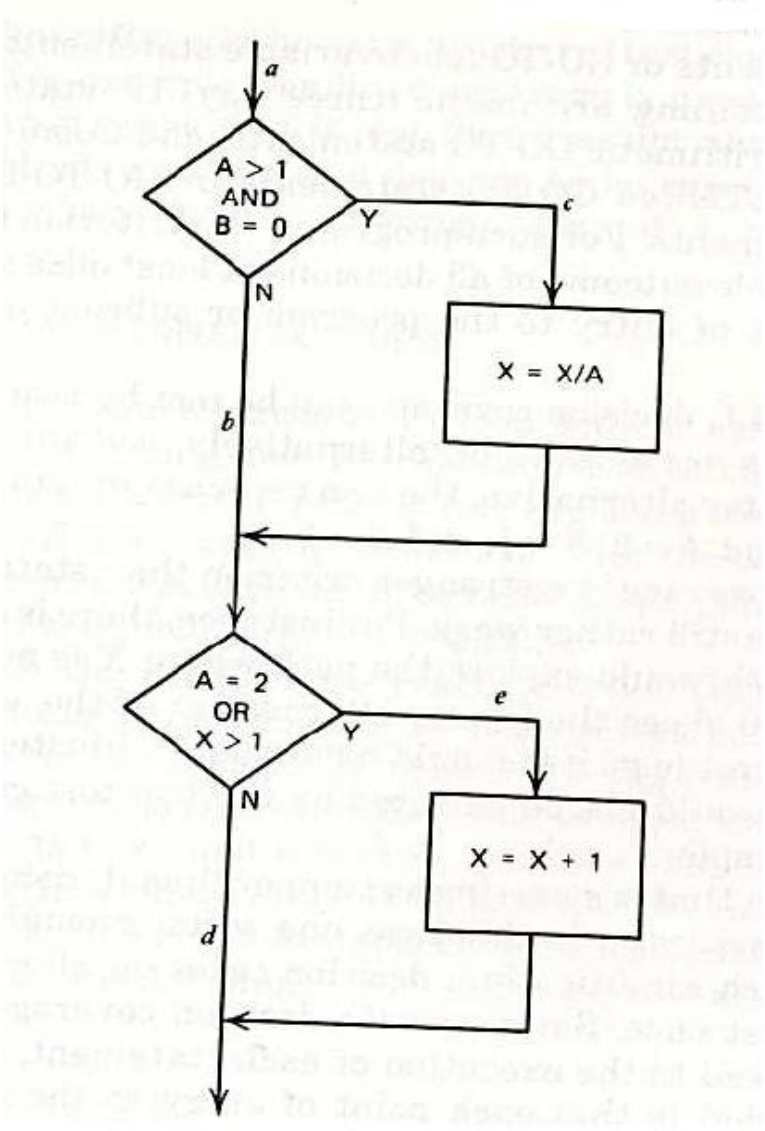
Branch Coverage/Decision Coverage

- In many codes if statement run and else does not runs, in branch coverage we run both if and if else
- The goal is to design test cases to ensure that each branch is executed once
- Each decision is evaluated to *true* and *false,* at least once traversed
- Generally satisfies statement coverage
- Branch coverage has problems with multiple conditions within a decision (&&, ||)

Branch Coverage/Decision Coverage

- **Each decision has a true and a false outcome at least once**

Test ID	Input	Output	Pass/Fail
WBT 3 (DC)	A=2 and B=0	Path (ace)	Pass
WBT 4(DC)	A=1 and X=1	Path (abd)	Pass



Branch Coverage

```
/* x is valid if 0 ≤ x ≤ 100 */
int check (int x ){
        if( x ≥ 0 && x ≤ 100 )
                return TRUE;
        else return FALSE;
        }
```

Test ID	Input	Output	Pass/Fail
WBT 5 (DC)	X=5	Run True	Pass
WBT 6(DC)	X=-5	Run False	Pass

Branch Coverage

```
/* finding the maximum of two integers x and y
*/
int Max_check(int x, y, max) {
If (x>y) max = x;
else max = y;
  }
```

Test ID	Input	Output	Pass/Fail
WBT 7(DC)	X=3 Y=2	Max=4	Pass
WBT 8(DC)	X=2 Y=4	Else max=y	Pass

Condition Coverage

Condition Coverage

- The goal is to design test cases to ensure that each condition in a decision takes on all possible outcomes at least once.
- Ensures that every condition within a decision is covered
- Consider the conditional expression:

  ((c1 and c2)or c3)

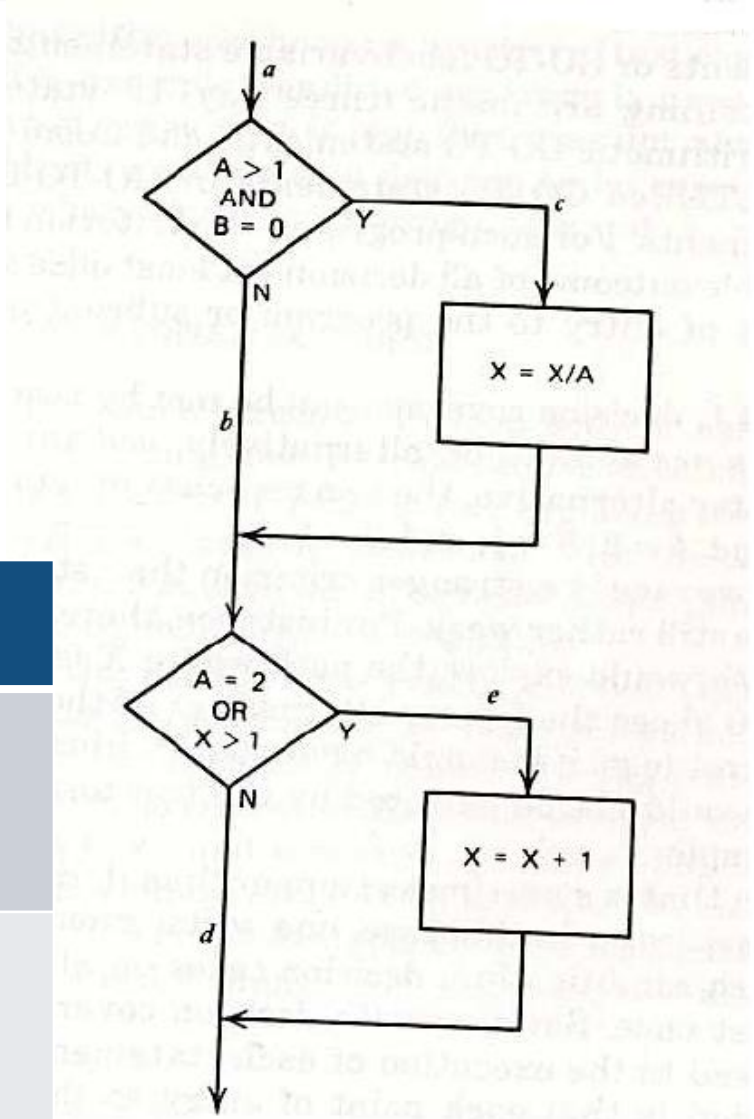
- Each of c1, c2, and c3 are exercised at least once,

i.e. given true and false values require 2^3 test cases.

Condition Coverage

- Each condition in a decision takes on all possible outcomes at least once
- Conditions: A>1, B=0, A=2, X>1

Test ID	Input	Output	Pass/Fail
WBT 8 (CC)	A=5 B=0 X=4 (for true)	ace	Pass
WBT 9(CC)	A=1 B=1 X=1 (for false)	abd	Pass



Condition Coverage

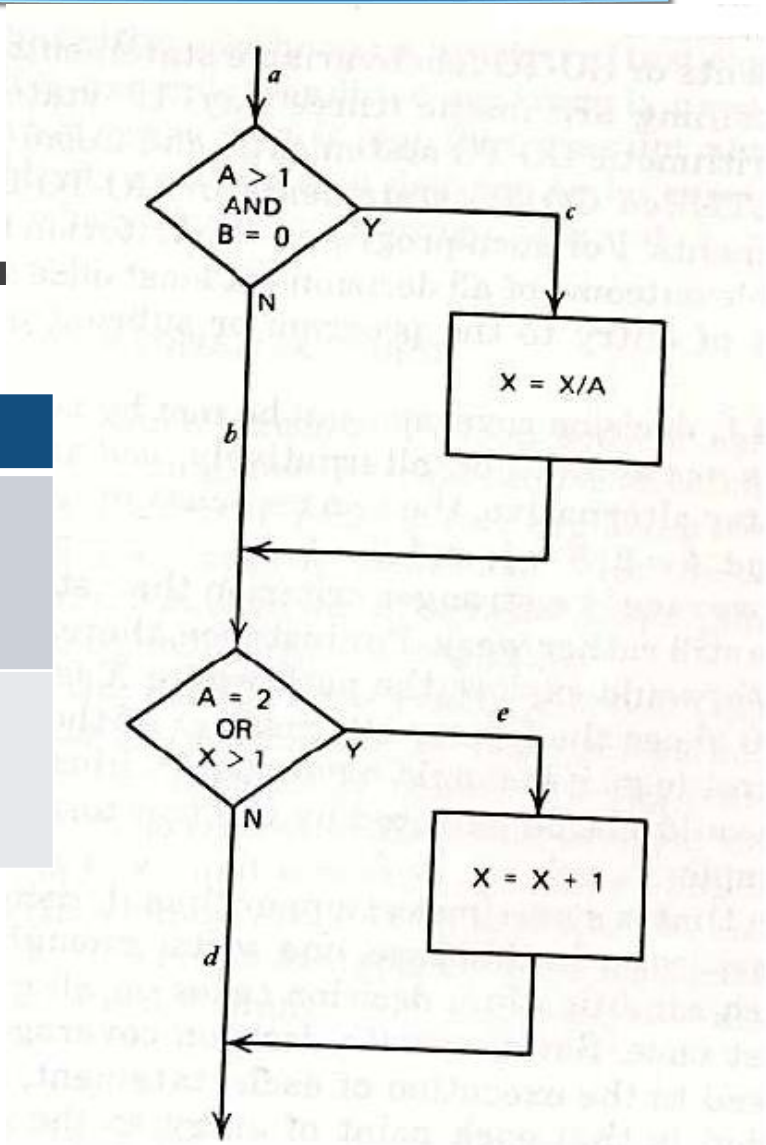
- **Each condition in a decision takes possible outcomes at least once, and**
- **Each decision takes on all possible ou at least once**

Test ID	Input	Output	Pass/Fail
WBT 10(CC)	A=2 B=0 X=4 (for true)	ace	Pass
WBT 9(CC)	A=1 B=1 X=1 (for false)	abd	Pass

- **What about these?**

A=1, B=0, and X=3 (abe)

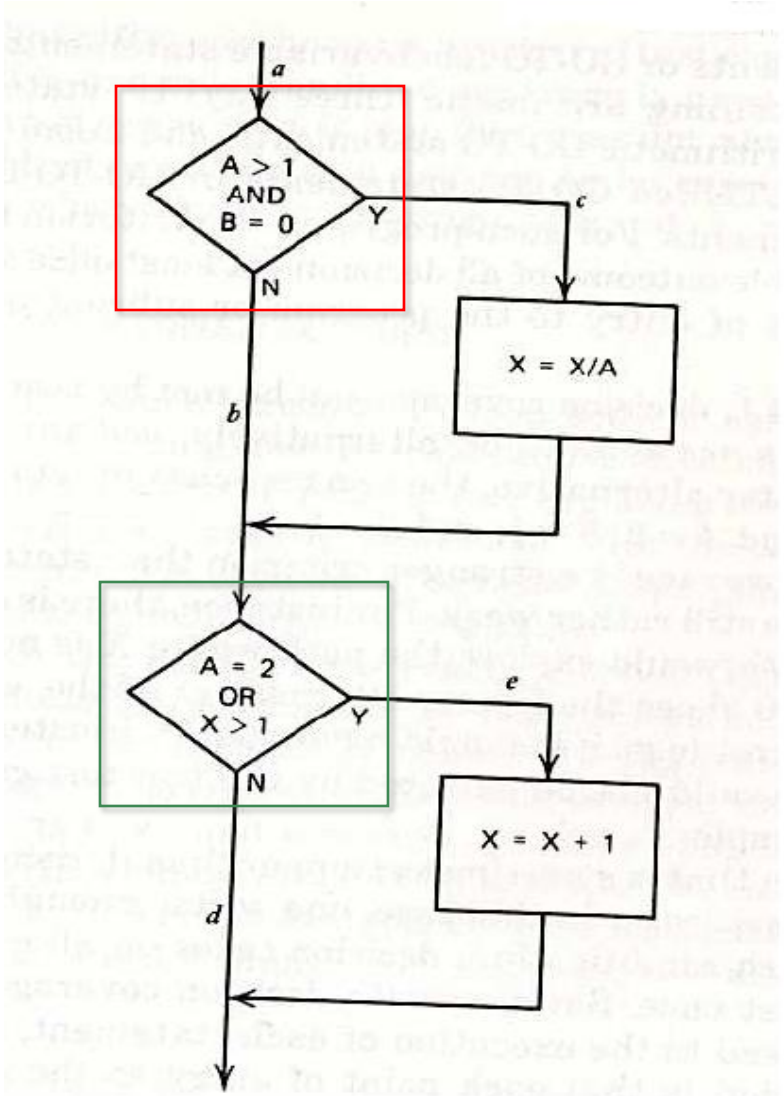
A=2, B=1, and X=1 (abe)



Multiple Condition Coverage

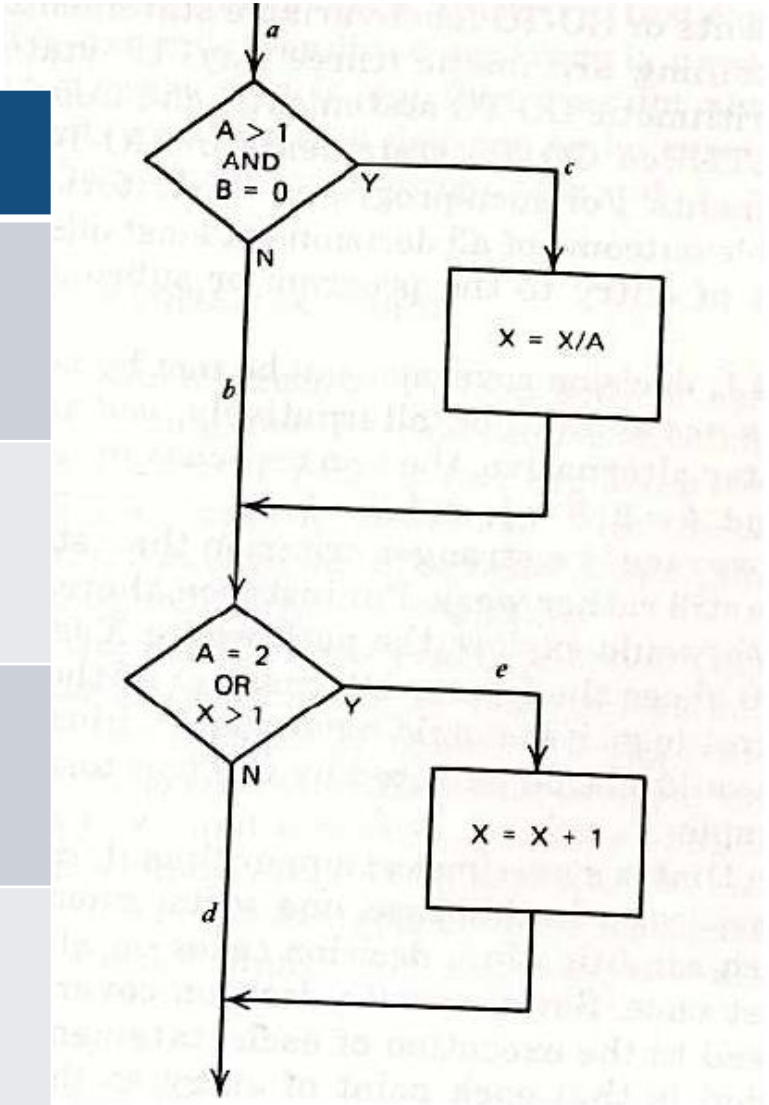
- **Exercise all possible combinations of condition outcomes in each decision**
- **Conditions:**

A > 1, B = 0	A = 2, X > 1
A > 1, B <> 0	A = 2, X <= 1
A <= 1, B = 0	A <> 2, X > 1
A <=1, B <> 0	A <> 2, X <= 1



Multiple Condition Coverage

Test ID	Input	Output	Pass/Fail
WBT 11 (CC)	A=2 B=0 X=4	ace	Pass
WBT 12(CC)	A=2 B=1 X=1	abe	Pass
WBT 13(CC)	A=1 B=0 X=2	abe	Pass
WBT 14(CC)	A=1 B=1 X=1	abd	Pass



Path Coverage

Path Testing

- Path testing is an approach of testing where you ensure that every path through a program has been executed at least once.
- We must execute these paths at least once in order to test the program thoroughly.
- We design test cases according to the identified paths.

Path Testing Approach

❑Steps

1. Draw a control flow graph.
2. Determine the cyclomatic complexity.
3. Find a set of paths.
4. Generate test cases for each path.

Cyclomatic Complexity

- Invented by Thomas McCabe (1974) to measure the complexity of a program's conditional logic
- Provides a quantitative measure of the logical complexity of a program
- Provides an upper bound for the number of tests that must be conducted to ensure all statements have been executed at least once

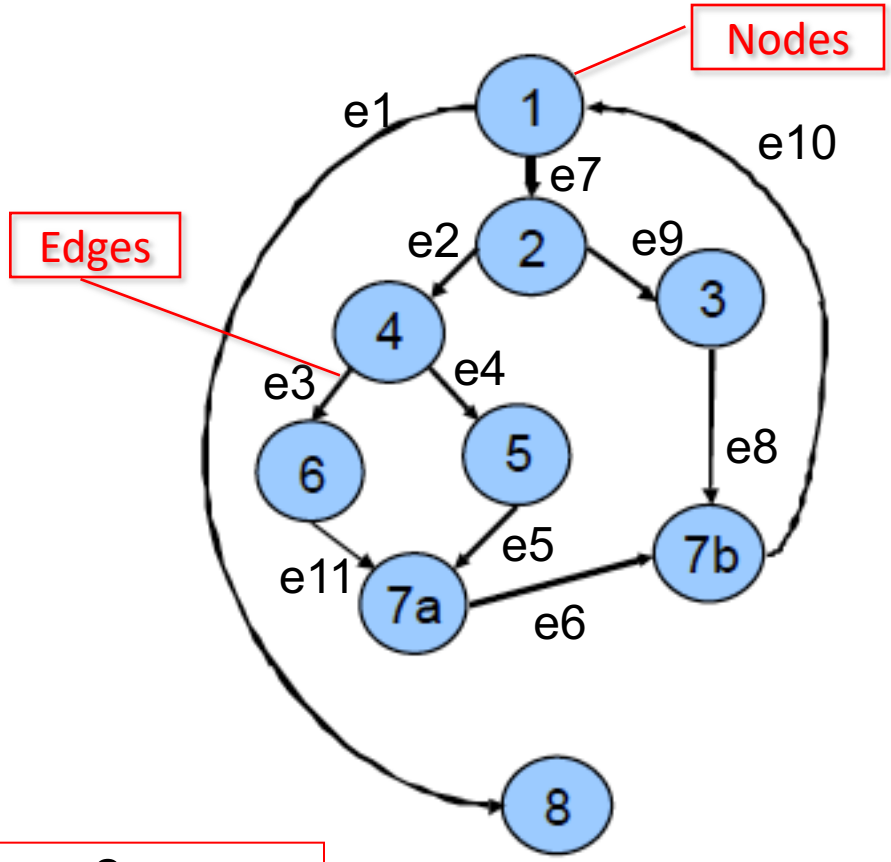
Cyclomatic Complexity

- **Cyclomatic Complexity**
  - V(G) = E – N + 2,
    - V refers to the cyclomatic number in graph
    - where E is the number of edges
    - and N is the number of nodes in graph G

- ***Though the McCabe's metric does not directly identify the linearly independent paths, but it informs approximately how many paths to look for***.

Example # 01

```
1.      do while records remain
            read record;
2.          if record field 1 = 0
3.              then process record;
                    store in buffer;
                    increment counter;
4.          elsif record field 2 = 0
5.              then reset record;
6.              else process record;
                    store in file;
7a.         endif;
        endif;
7b.     enddo;
8.  end;
```



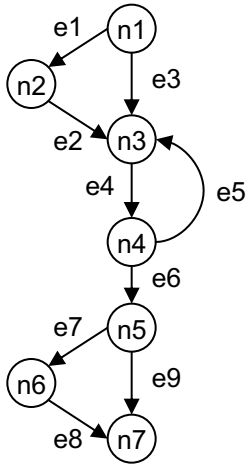
$$V(G) = e - n + 2$$

$$= 11 - 9 + 2$$

$$= 4$$

Example # 02

```
if expression1
then
     statement2
end if

do
     statement3
  while expr4
end do

if expression5
then
     statement6
end if
statement7
```



Paths:

P1 = e1, e2, e4, e6, e7, e8

P2 = e1, e2, e4, e5, e4, e6, e7, e8

P3 = e3, e4, e6, e7, e8, e10

P4 = e6, e7, e8, e10, e3, e4

P5 = e1, e2, e4, e6, e9, e10

P6 = e4, e5

P7 = e3, e4, e6, e9, e10

P8 = e1, e2, e4, e5, e4, e6, e9, e10

$$V(G) = e - n + 2$$

$$= 9 - 7 + 2$$

$$= 4$$

How Complex Should Code Be?

Complexity Number	Meaning
1-10	Structured and well written code High Testability Cost and Effort is less
10-20	Complex Code Medium Testability Cost and effort is Medium
20-40	Very complex Code Low Testability Cost and Effort are high
>40	Not at all testable Very high Cost and Effort

Higher the **Cyclomatic complexity** number, the more complex the code

Advantages of White Box Testing

- As tester has knowledge of the source code it is easy to find errors.
- **White box testing helps us to identify memory leaks.**
  - When we allocate memory using malloc( ) in C, we should explicitly release that memory also.
  - If this is not done then over time, there would be no memory available for allocating memory on requests.
- **Performance analysis**
  - Code coverage tests can identify the areas of a code that are executed most frequently.
  - Extra efforts can be then made to check these sections of code.

Advantages of White Box Testing

- **White box testing is useful in identifying bottlenecks in resource usage.**
  - For example, if particular resource like RAM or ROM or even network is perceived as a bottleneck then code can help identify where the bottlenecks are and point towards possible solutions.
- **White box testing can help identify security holes in dynamically generated code.**
  - For example in case of Java, some intermediate code may also be generated.
  - Testing this intermediate code requires code knowledge.
  - White box testing only does this.

Advantages of White Box Testing

- Due to testers knowledge of code , maximum coverage is attained during test scenario.

- Testers can then see if the program diverges from its intended goal.

Disadvantages of White Box Testing

- As skilled tester are performing tests, costs is increased.
- As it is very difficult to look into every corner of the code, some code will go unchecked.
- White box testing does not account for errors caused by omission.