Unit – 1
A Perspective on Testing, Examples

ST
8th Sem, A’
Div 2017 -
Prof. Mouna M. Naravani

Error, Fault, &Failure

- Error
  - Represents mistakes made by people
- Fault
  - Is result of error. May be categorized as
    - Fault of Commission – we enter something into representation that is incorrect
    - Fault of Omission – Designer can make error of omission, the resulting fault is that something is missing that should have been present in the representation
- Failure
  - Occurs when fault executes.
- Incident
  - Behavior of fault. An incident is the symptom(s) associated with a failure that alerts user to the occurrence of a failure

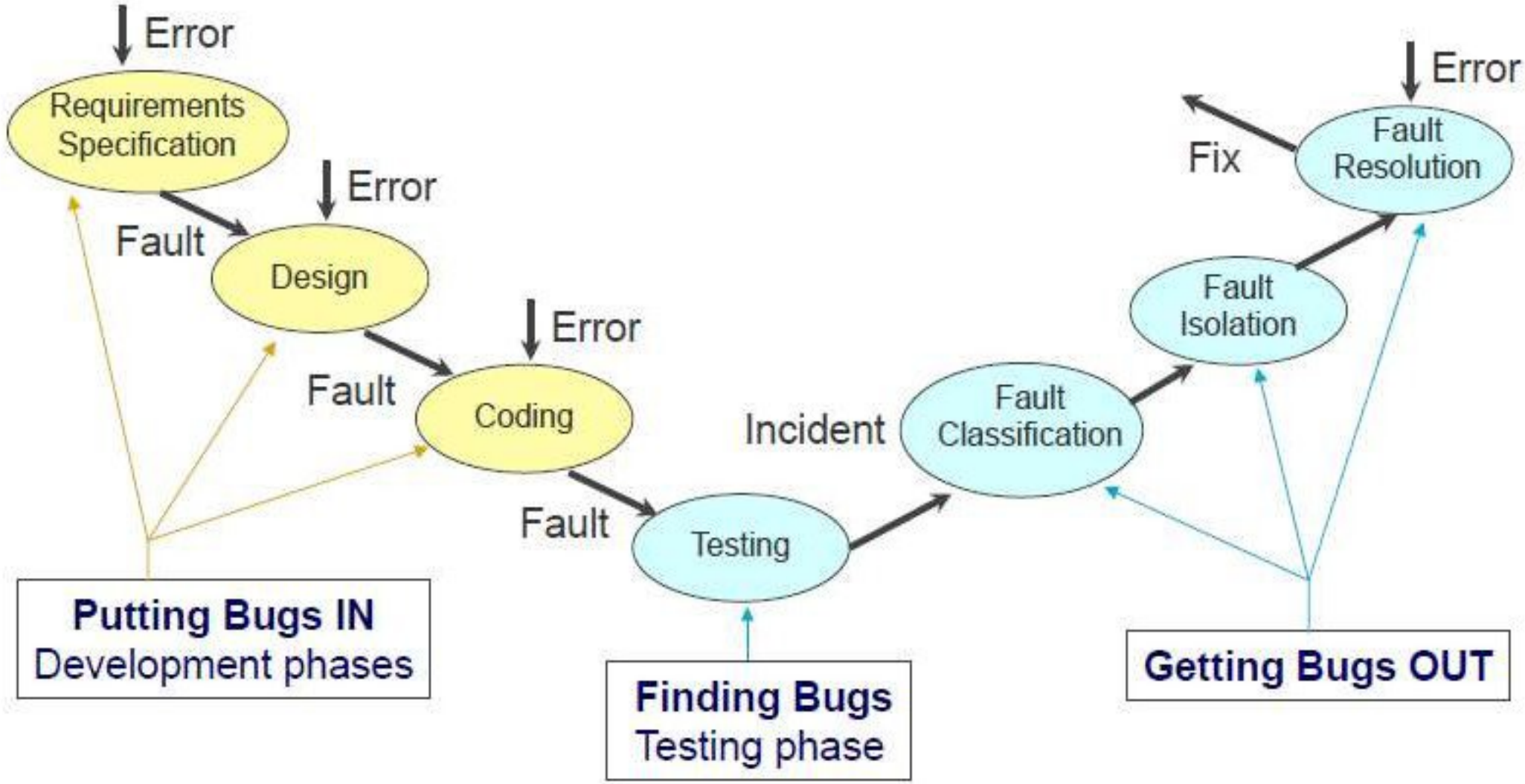
Test

Associated with program behaviour. It carries set of input and list of expected output

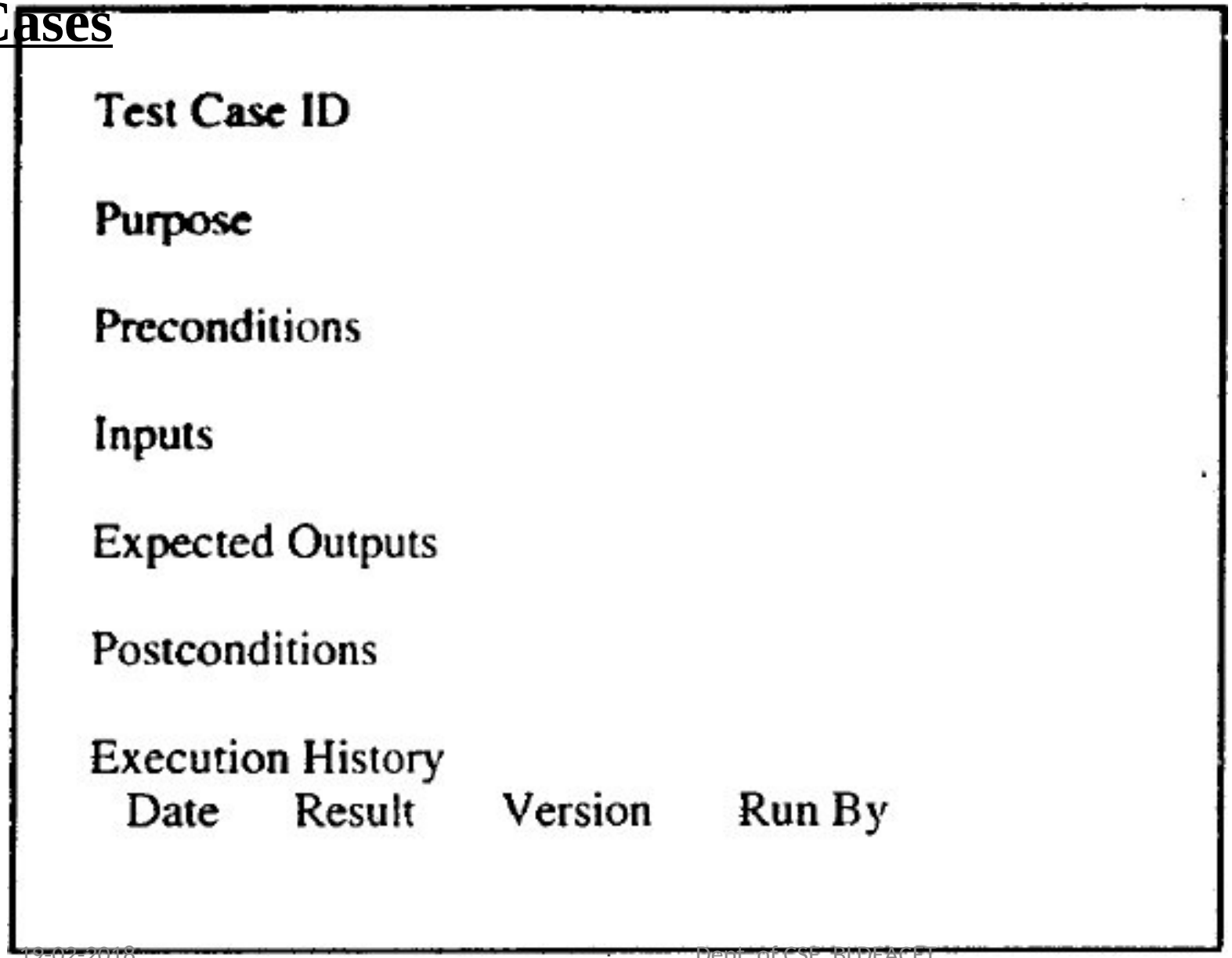
Test case

Test case has an identity and is associated with a program behaviour. A test case also has set of inputs and expected outputs.

A Testing Life Cycle



Test Cases


Test Case ID
Purpose
Preconditions
Inputs
Expected Outputs
Postconditions
Execution History
Date
Result
Version
Run By

Pre-condition: Any prerequisite that must be fulfilled before execution of this test case. List all the pre-conditions in order to execute this test case successfully.

Inputs: What are the inputs for this test case. You can provide different data sets with exact values to be used as an input.

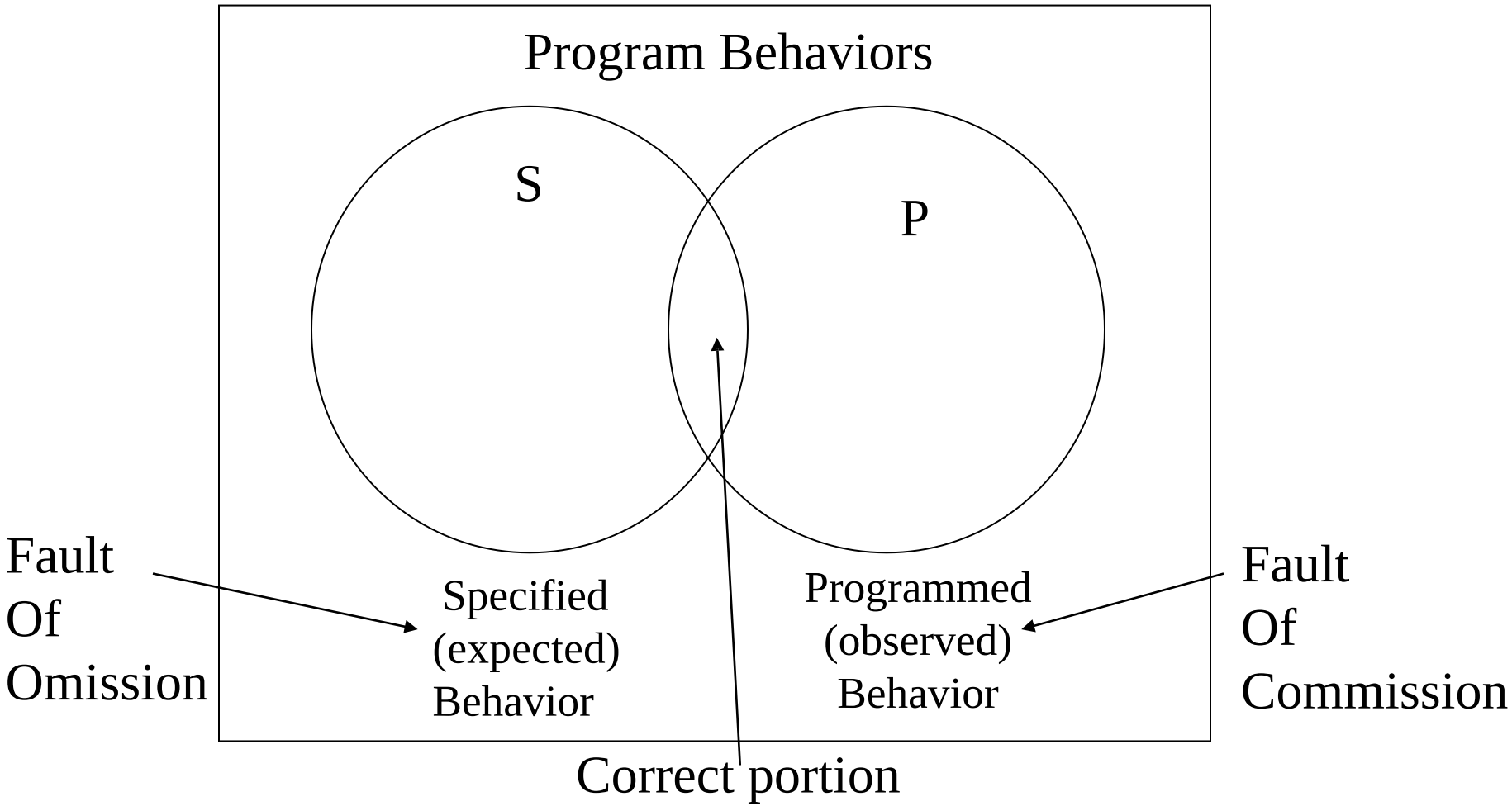
Expected Outputs: What should be the system output after test execution? Describe the expected result in detail including message/error that should be displayed on the screen.

Post-condition: What should be the state of the system after executing this test case?

Insights from a Venn Diagram

- Testing is fundamentally concerned with
  - Behavioural view
  - Structural view
- Structural view focuses on ***"What it is"***
- Behavioural view focuses on ***"What it does"***
- Difficulty for testers: base documents are usually written by and for developers, therefore, emphasis is on structural instead of behavioural.

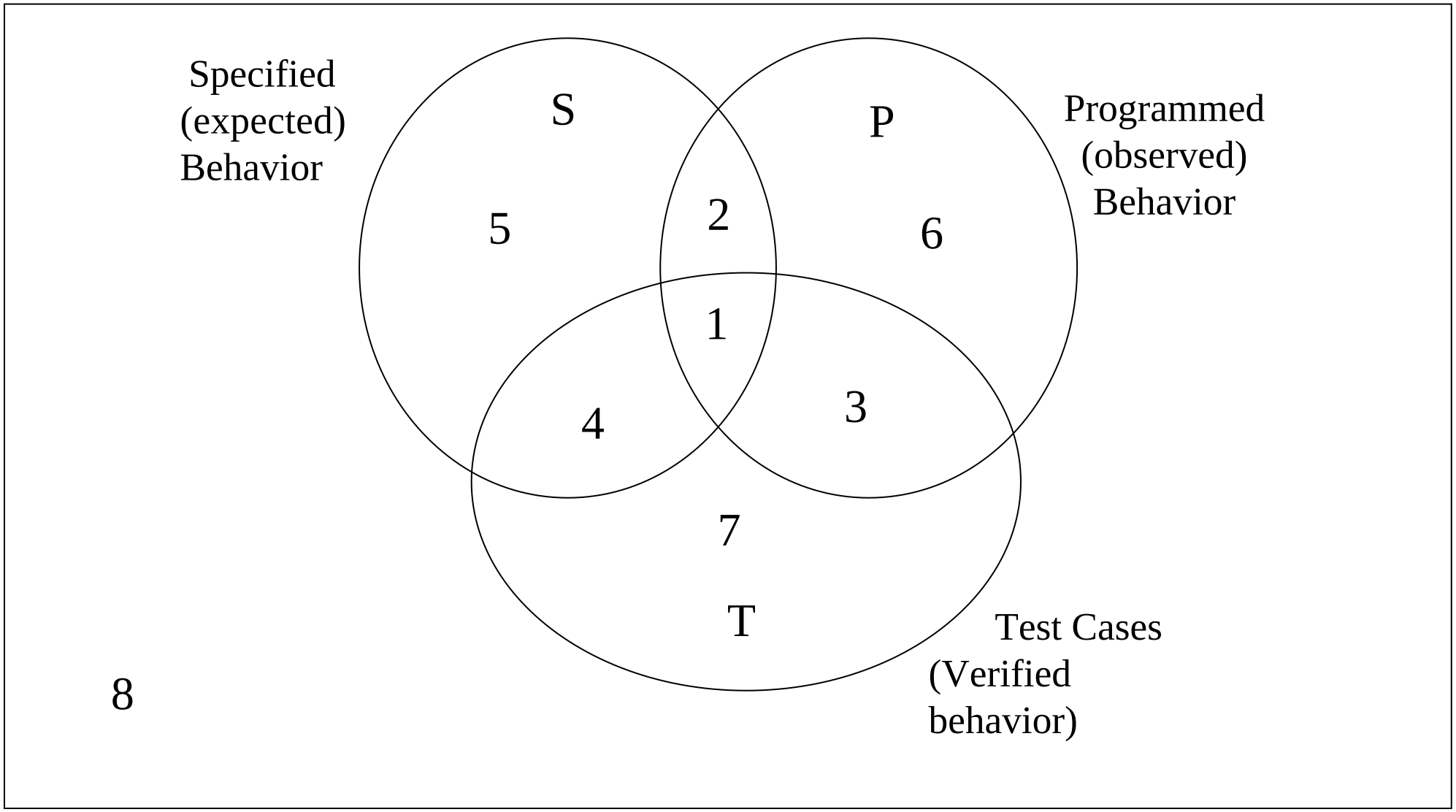
Fig: Specified and Implemented Program behaviours



- Universe : Program Behaviours
- Set S: Specified behaviours (expected)
- Set P: Behaviours actually Programmed (observed)
- Problems that confront a tester:
  - What if certain specified behaviours have not been programmed?
    - These are **faults of omission**.
  - What if certain programmed (implemented) behaviours have not been specified?
    - These are **faults of commission**.

- The intersection of S & P: is the ***"correct"*** portion, that is, behaviours that are both specified and implemented.
- Good Testing: determining the extent of program behaviour that is both specified and
implemented.

Fig: Specified, implemented and tested behaviors



- 2, 5
  - Specified behavior that are not tested
- 1, 4
  - Specified behavior that are tested
- 3, 7
  - Test cases corresponding to unspecified behavior

- 2, 6
  - Programmed behavior that are not tested
- 1, 3
  - Programmed behavior that are tested
- 4, 7
  - Test cases corresponding to un-programmed behaviors

Inferences…

- If there are specified behaviors for which there are no test cases, the testing is incomplete
- If there are test cases that correspond to unspecified behaviors
  - Either such test cases are unwarranted
  - Specification is deficient (also implies that testers should participate in specification and design reviews)

Some possibilities for testing……

What can a tester do to make the region where these sets all intersect (region 1) as large as possible?

How the test cases in set T are identified?

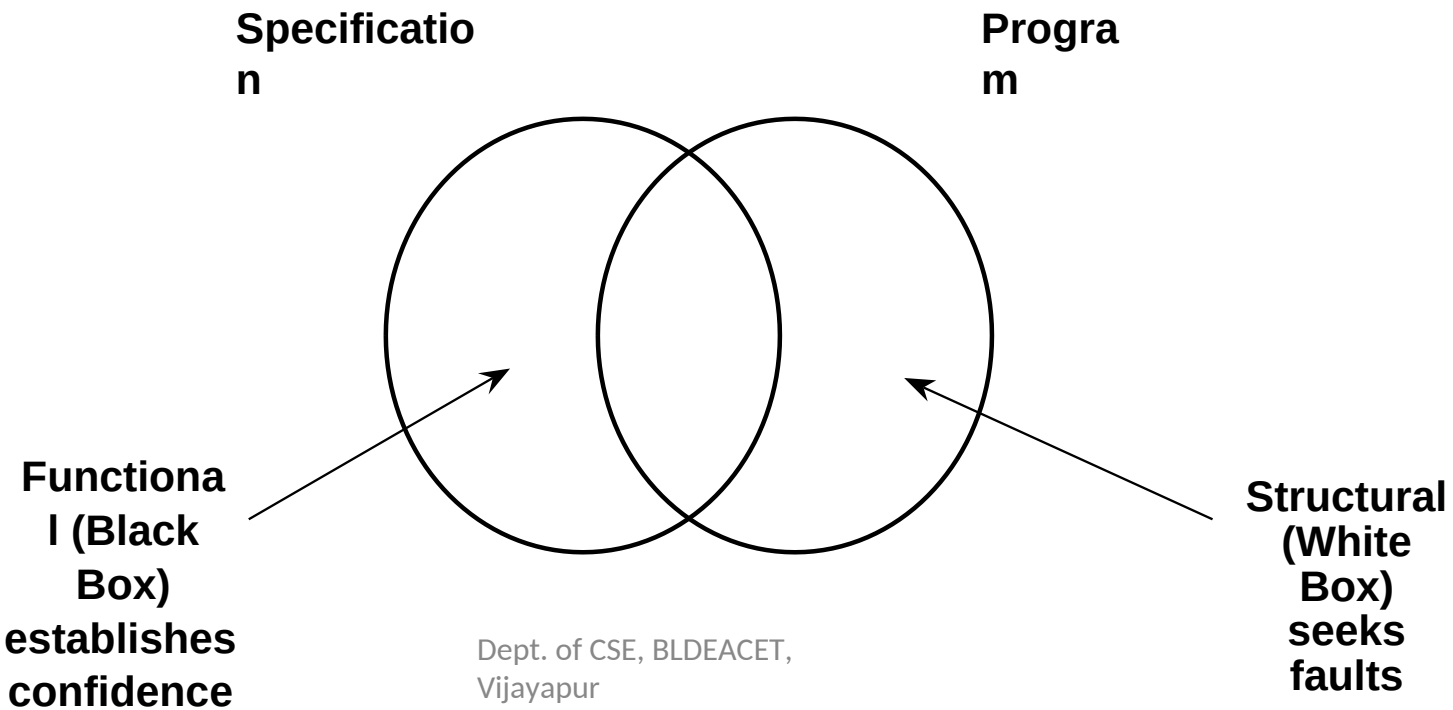
By, Testing Methods.

Identifying Test Cases

Two approaches:

- Functional (Black box) inspects specified behavior
- Structural (White box) inspects programmed behavior

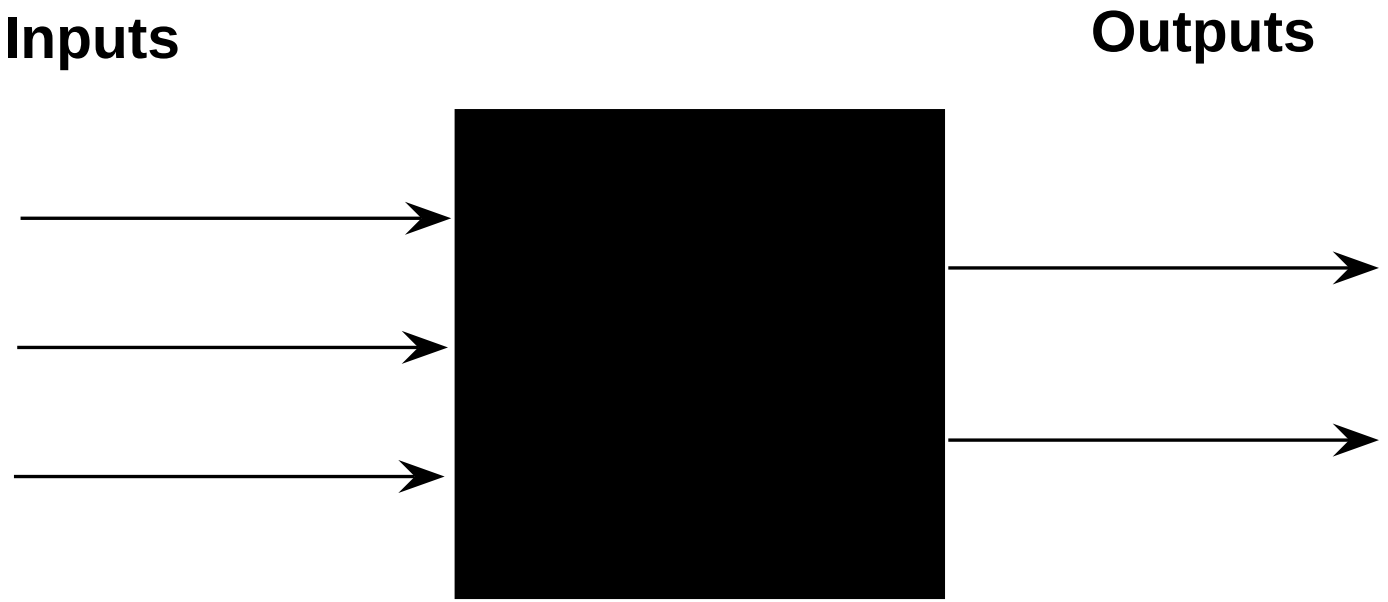
Basic Approaches



Functional Testing

- Functional testing is based on the view that any program can be considered to be a function that maps values from its input domain to values in its output range.
- This leads to "**Black Box Testing**", in which the implementation of a black box is not known, and the function of the black box is understood completely in terms of its input and output.
- Example: Automobiles

Black Box



Function is understood only in terms of it's inputs and outputs, with no knowledge of its implementation.

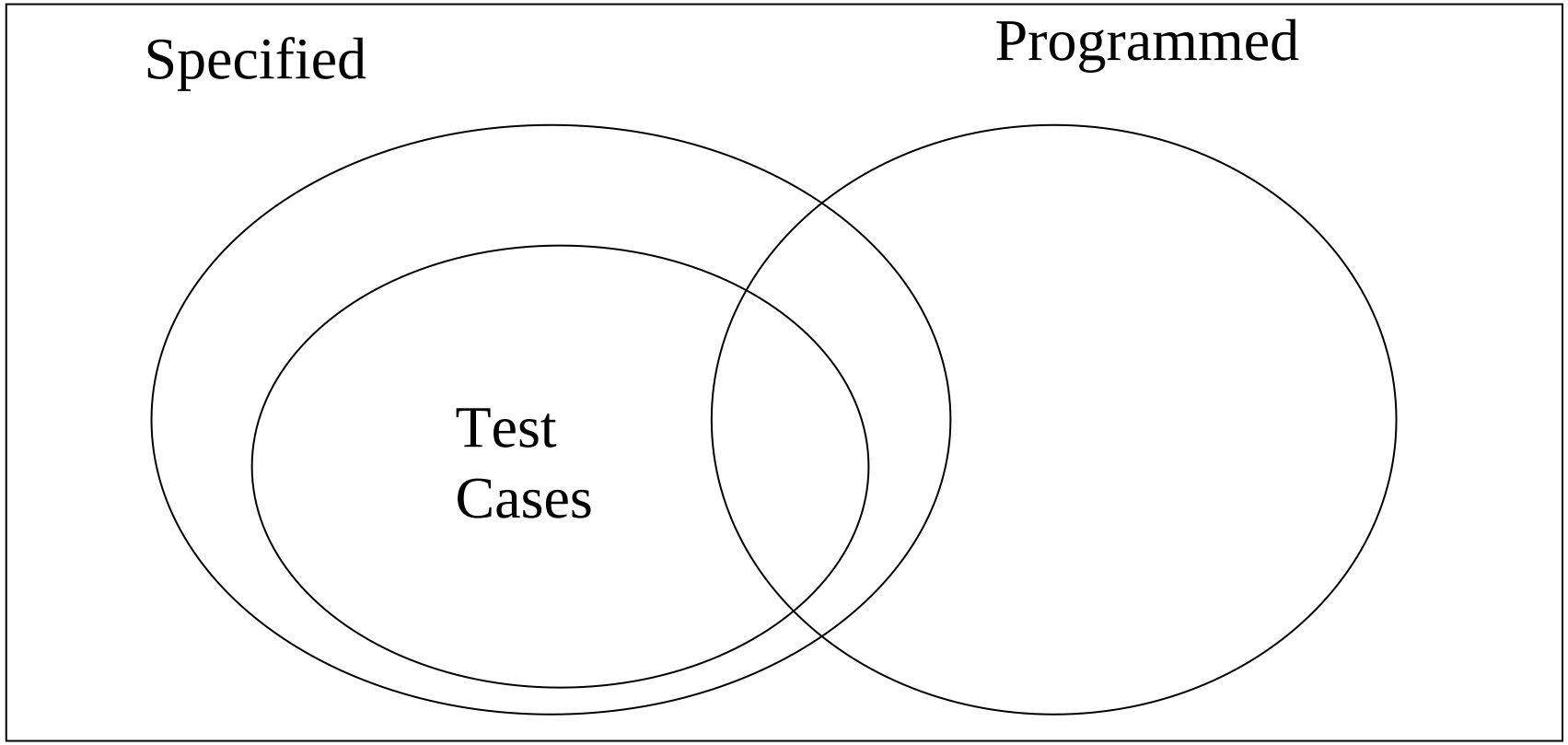
Functional test cases have 2 advantages:

1. They are independent of how the software is implemented, so if the implementation changes, the test cases are still useful.
2. Test case development can occur in parallel with the implementation, thereby reducing the overall project development time.

Disadvantages:

1. Redundancies may exist among test cases
2. Possibility of gaps of untested software

Functional Test cases

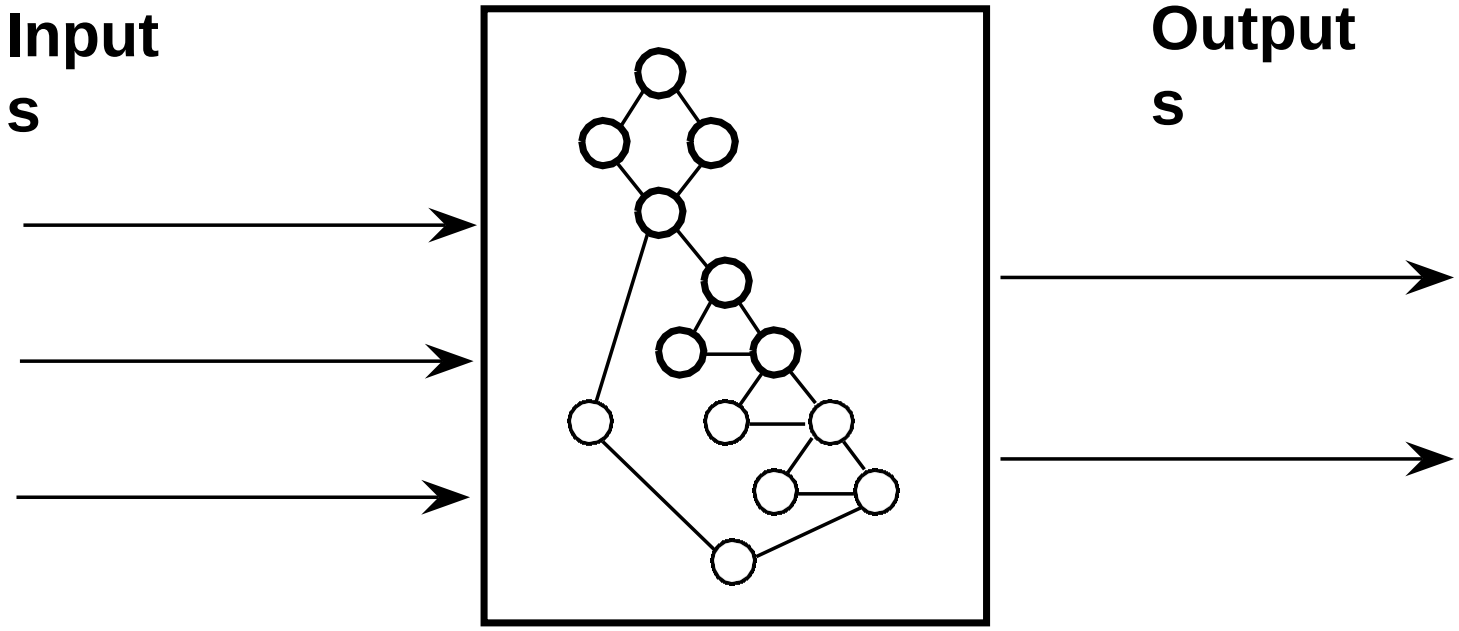


Because functional testing is based on specified behaviours, test cases are present within the set of specified behaviours.

Structural Testing

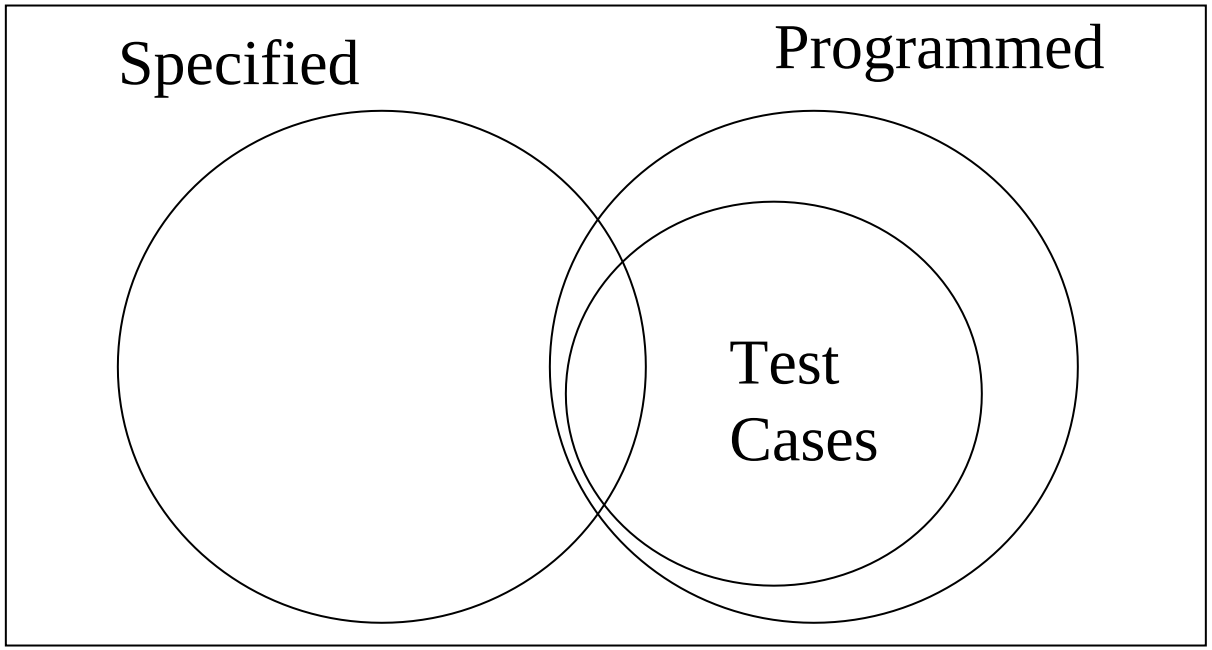
- Also called as "**White Box or Clear Box**" Testing.
- Here implementation is known and used to identify test cases.
- The ability to "see inside" the black box allows the tester to identify test cases based on how the function is actually implemented.
- The tester can rigorously describe what exactly is tested.
- Test coverage metrics provide a way to explicitly state the extent to which a software item has been tested, and this in turn makes testing management more meaningful.

White (Clear) Box



Function is understood only in terms of its implementation.

Structural Test cases



Because structural testing is based on programmed behaviours, test cases are present within the set of programmed behaviours.

Advantages of White Box Testing:

- Forces test developer to reason carefully about implementation.
- Reveals errors in "hidden" code.
- Spots the Dead Code or other issues with respect to best programming practices.

Disadvantages of White Box Testing:

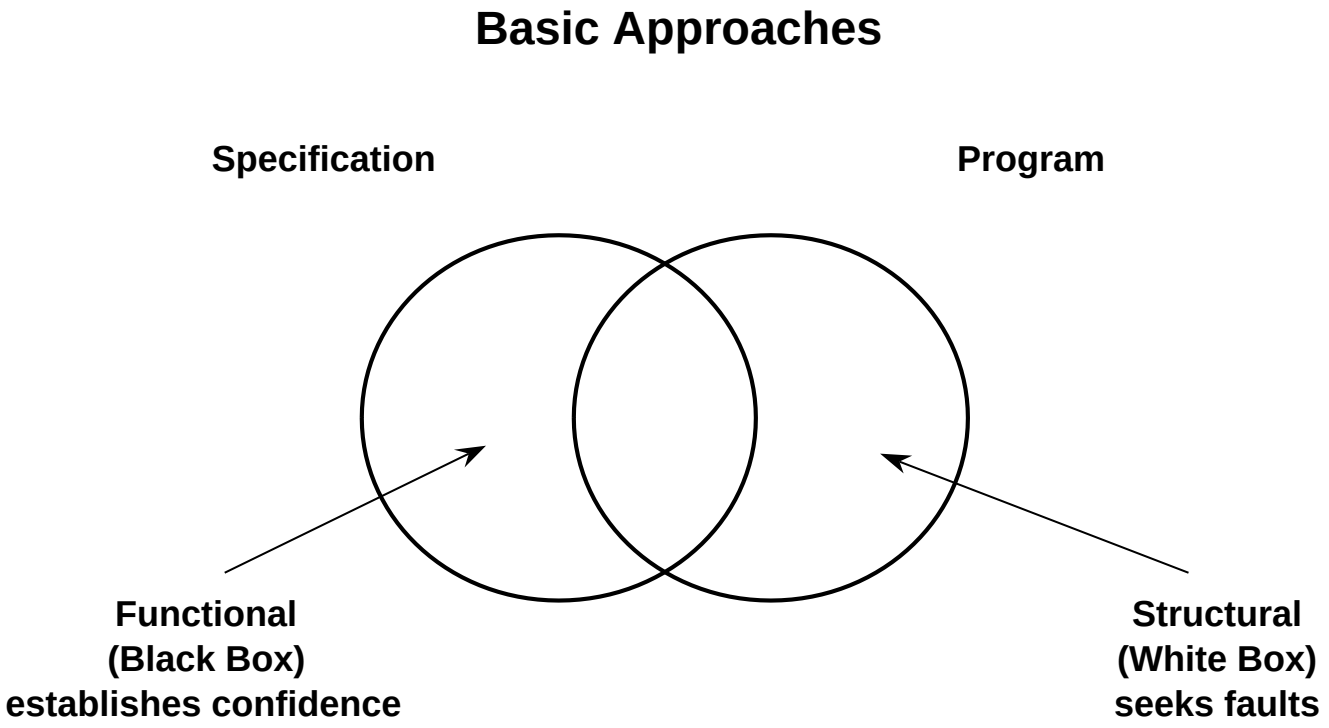
- Expensive as one has to spend both time and money to perform white box testing.
- Every possibility that few lines of code are missed accidentally.
- In-depth knowledge about the programming language is necessary to perform white box testing.

The Functional versus Structural Debate

- Which is better?
- Referring to Structural testing, Robert Poston writes,
  "This tool has been wasting testers time since it does not support good software testing practice and should not be in the testers toolkit".
- In the defence of structural testing, Edward Miller writes,
  "A structural test, if attained 85% or better level, tends to identify twice the number of defects that would have been found functional testing".

- The goal of both approaches is to identify test cases.
- Functional testing uses only specification to identify test cases
- Structural testing uses program source code(implementation) to identify test cases
- Consider Program behaviour:
  - If all specified behaviours have not been implemented, structural test cases will never be able to recognize this.
  - If the program implements behaviour that have not been specified, this will never be revealed by functional test cases.
- Quick Answer: **both approaches are needed**.

- Combination of both will provide the confidence of functional testing and the measurement of structured testing.



- When functional test cases are executed in combination with structural test cases, both problems of functional testing can be recognized and resolved.

Difference between Black Box and White Box Testing

Black Box Testing	White Box Testing
Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
This type of testing is carried out by testers.	Generally, this type of testing is carried out by software developers.
Implementation Knowledge is not required to carry out Black Box Testing.	Implementation Knowledge is required to carry out White Box Testing.
Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.
Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.
Black box testing means functional test or external testing.	White box testing means structural test or interior testing.

Error and Fault Taxonomies

- **Process**: refers to how we do something
- **Product**: is the end result of a process
- **Software Quality Assurance(SQA)**: It tries to improve the product by improving the process.
- SQA is more concerned with reducing errors in the development process.
- Testing is concerned with discovering faults in a product.
- Faults can be classified in several ways:
  - Faults by severity
  - I/O Faults
  - Logic Faults
  - Computation Faults
  - Interface Faults
  - Data Faults

1. Mild	Misspelled word
2. Moderate	Misleading or redundant information
3. Annoying	Truncated names, bill for $0.00
4. Disturbing	Some transaction(s) not processed
5. Serious	Lose a transaction
6. Very serious	Incorrect transaction execution
7. Extreme	Frequent "very serious" errors
8. Intolerable	Database corruption
9. Catastrophic	System shutdown
10. Infectious	Shutdown that spreads to others

Faults classified by severity.

Table 1.1 Input/Output Faults

Type	*Instances*
Input	Correct input not accepted
	Incorrect input accepted
	Description wrong or missing
	Parameters wrong or missing
Output	Wrong format
	Wrong result
	Correct result at wrong time (too early, too late)
	Incomplete or missing result
	Spurious result
	Spelling/grammar
	Cosmetic

Table 1.2 Logic Faults

Missing case(s)
Duplicate case(s)
Extreme condition neglected
Misinterpretation
Missing condition
Extraneous condition(s)
Test of wrong variable
Incorrect loop iteration
Wrong operator (e.g., < instead of ≤)

Table 1.3 Computation Faults

Incorrect algorithm
Missing computation
Incorrect operand
Incorrect operation
Parenthesis error
Insufficient precision (round-off, truncation)
Wrong built-in function

Table 1.4 Interface Faults

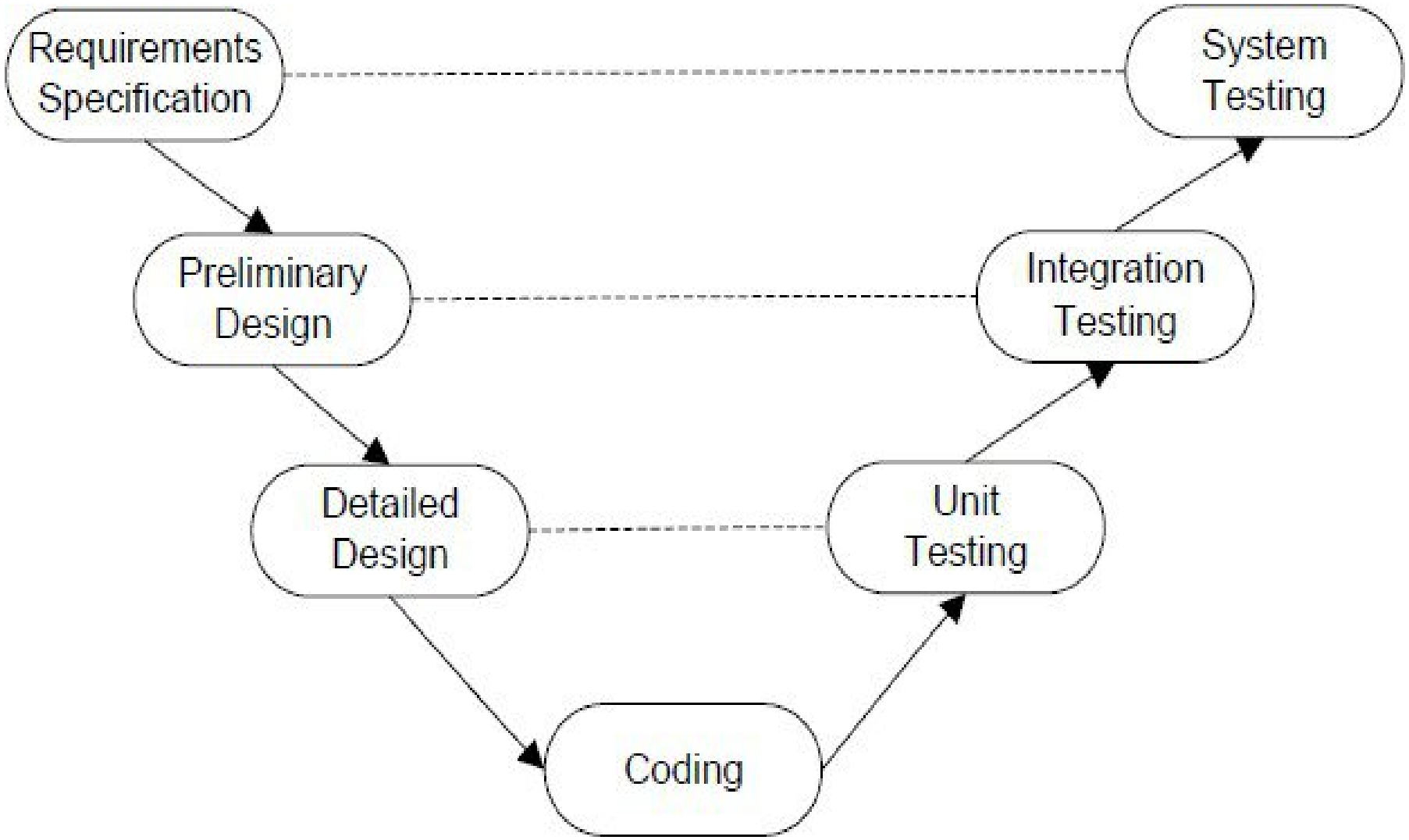
Incorrect interrupt handling
I/O timing
Call to wrong procedure
Call to nonexistent procedure
Parameter mismatch (type, number)
Incompatible types
Superfluous inclusion

Table 1.5 Data Faults

Incorrect initialization
Incorrect storage/access
Wrong flag/index value
Incorrect packing/unpacking
Wrong variable used
Wrong data reference
Scaling or units error
Incorrect data dimension
Incorrect subscript
Incorrect type
Incorrect data scope
Sensor data out of limits
Off by one
Inconsistent data

Levels of Testing

- Levels of testing is similar to waterfall model of software development life cycle.
- Functional testing is most appropriate at the system level.
- Structural testing is most appropriate at the unit level.


Requirements Specification
System Testing
Preliminary Design
Integration Testing
Detailed Design
Unit Testing
Coding

Level	Summary
Unit Testing	A level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
Integration Testing	A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
System Testing	A level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.

Examples

- The Triangle Problem
- The NextDate Function
- The Commission Problem
- The SATM System
- The Currency Converter
- Saturn Windshield Wiper Controller

Generalized Pseudocode

Pseudocode provides a “language neutral” way to express program source code.

Language Element	*Generalized Pseudocode Construct*
Comment	' <text>
Data structure declaration	Type <type name> <list of field descriptions> End <type name>
Data declaration	Dim <variable> As <type>
Assignment statement	<variable> = <expression>
Input	Input (<variable list>)
Output	Output (<variable list>)
Condition	<expression> <relational operator> <expression>
Compound condition	<Condition> <logical connective> <Condition>
Sequence	statements in sequential order

Simple selection	If <condition> Then
	<then clause>
	EndIf

Selection	If <condition> Then <then clause> Else <else clause> EndIf
Multiple selection	Case <variable> Of Case 1: <predicate> <Case clause> ... Case *n*: <predicate> <Case clause> EndCase
Counter-controlled repetition	For <counter> = <start> To <end> <loop body> EndFor
Pretest repetition	While <condition> <loop body> EndWhile

Posttest repetition	Do <loop body> Until <condition>
Procedure definition (similarly for functions and o–o methods)	<procedure name> (Input: <list of variables>; Output: <list of variables>) <body> End <procedure name>
Interunit communication	Call <procedure name> (<list of variables>; <list of variables>)
Class/Object definition	<name> (<attribute list>; <method list>, <body>) End <name>
Interunit communication	msg <destination object name>.<method name> (<list of variables>)
Object creation	Instantiate <class name>.<object name> (list of attribute values)
Object destruction	Delete <class name>.<object name>
Program	Program <program name> <unit list> End<program name>

Triangle Problem

Simple version: The triangle program accepts three integers, a, b, and c, as input. These are taken to be sides of a triangle. The output of the program is the type of triangle determined by the three sides: Equilateral, Isosceles, Scalene, or Not A Triangle.

Improved version: "Simple version" plus better definition of inputs:

The integers a, b, and c must satisfy the following conditions:

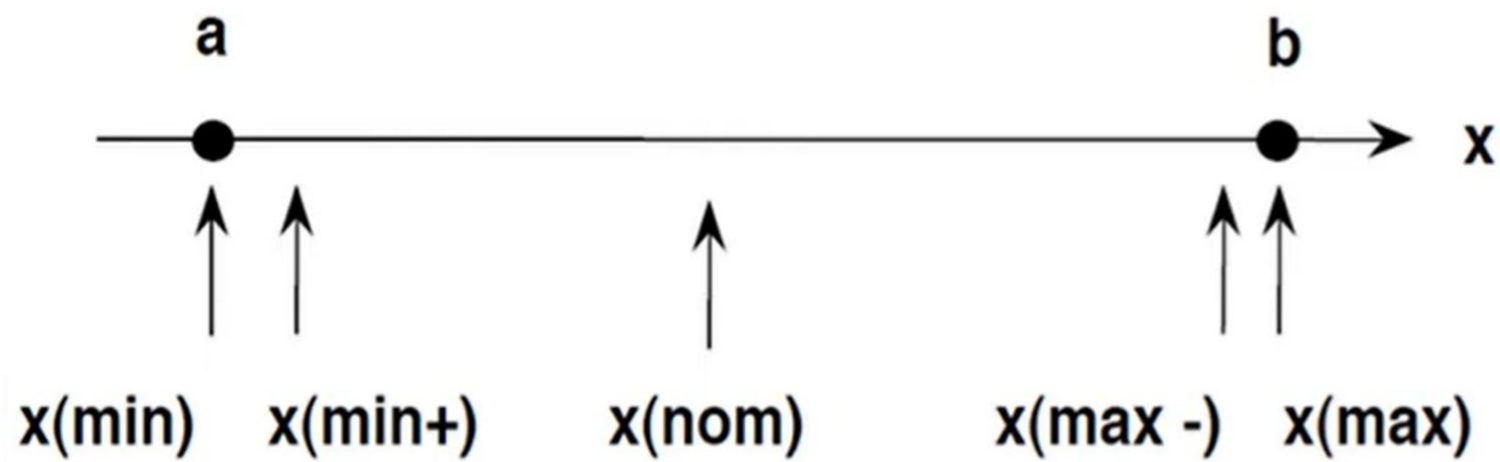
c1.	$1 \le a \le 200$	c4.	$a < b + c$
c2.	$1 \le b \le 200$	c5.	$b < a + c$
c3.	$1 \le c \le 200$	c6.	$c < a + b$

Triangle Problem Discussion

Final Version: "Improved version" plus better definition of outputs:
If an input value fails any of conditions c1, c2, or c3, the program notes this with an output message, for example, "Value of b is not in the range of permitted values." If values of a, b, and c satisfy conditions c1, c2, and c3, one of four mutually exclusive outputs is given:

1. If all three sides are equal, the program output is Equilateral.
2. If exactly one pair of sides is equal, the program output is Isosceles.
3. If no pair of sides is equal, the program output is Scalene.
4. If any of conditions c4, c5, and c6 is not met, the program output is NotATriangle.

Input variable values for BVT



Test Case Name :Boundary Value Analysis for triangle problem

Experiment Number : 01

Test Data : Enter the 3 Integer Value(a , b And c)

Pre-condition : $1 \leq a \leq 10$, $1 \leq b \leq 10$ and $1 \leq c \leq 10$ and $a < b + c$, $b < a + c$ and $c < a + b$

Brief Description : Check whether given value for a Equilateral, Isosceles , Scalene triangle or can't form a triangle.

Case Id	Description	Input Data			Expected Output	Actual Output
		a	b	c		
1	Enter the min value for a , b and c	1	1	1	Should display the message Equilateral triangle	
2	Enter the min value for 2 items and min +1 for any one item1	1	1	2	Message should be displayed can't form a Triangle	
3	Enter the min value for 2 items and min +1 for any one item1	1	2	1	Message should be displayed can't form a triangle	
4	Enter the min value for 2 items and min +1 for any one item1	2	1	1	Message should be displayed can't form a triangle	
5	Enter the normal value for 2 items and 1 item is min value	5	5	1	Should display the message Isosceles triangle	
6	Enter the normal value for 2 items and 1 item is min value	5	1	5	Should display the message Isosceles triangle	
7	Enter the normal value for 2 items and 1 item is min value	1	5	5	Should display the message Isosceles triangle	
8	Enter the normal Value for a, b and c	5	5	5	Should display the message Equilateral triangle	

9	Enter the normal value for 2 items and 1 item is max value	5	5	10	Should display the message Not a triangle	
10	Enter the normal value for 2 items and 1 item is max value	5	10	5	Should display the message Not a triangle	
11	Enter the normal value for 2 items and 1 item is max value	10	5	5	Should display the message Not a triangle	
12	Enter the max value for 2 items and max - 1 for any one item	10	10	9	Should display the message Isosceles triangle	
13	Enter the max value for 2 items and max - 1 for any one item	10	9	10	Should display the message Isosceles triangle	
14	Enter the max value for 2 items and max - 1 for any one item	9	10	10	Should display the message Isosceles triangle	
15	Enter the max value for a, b and c	10	10	10	Should display the message Equilateral triangle	

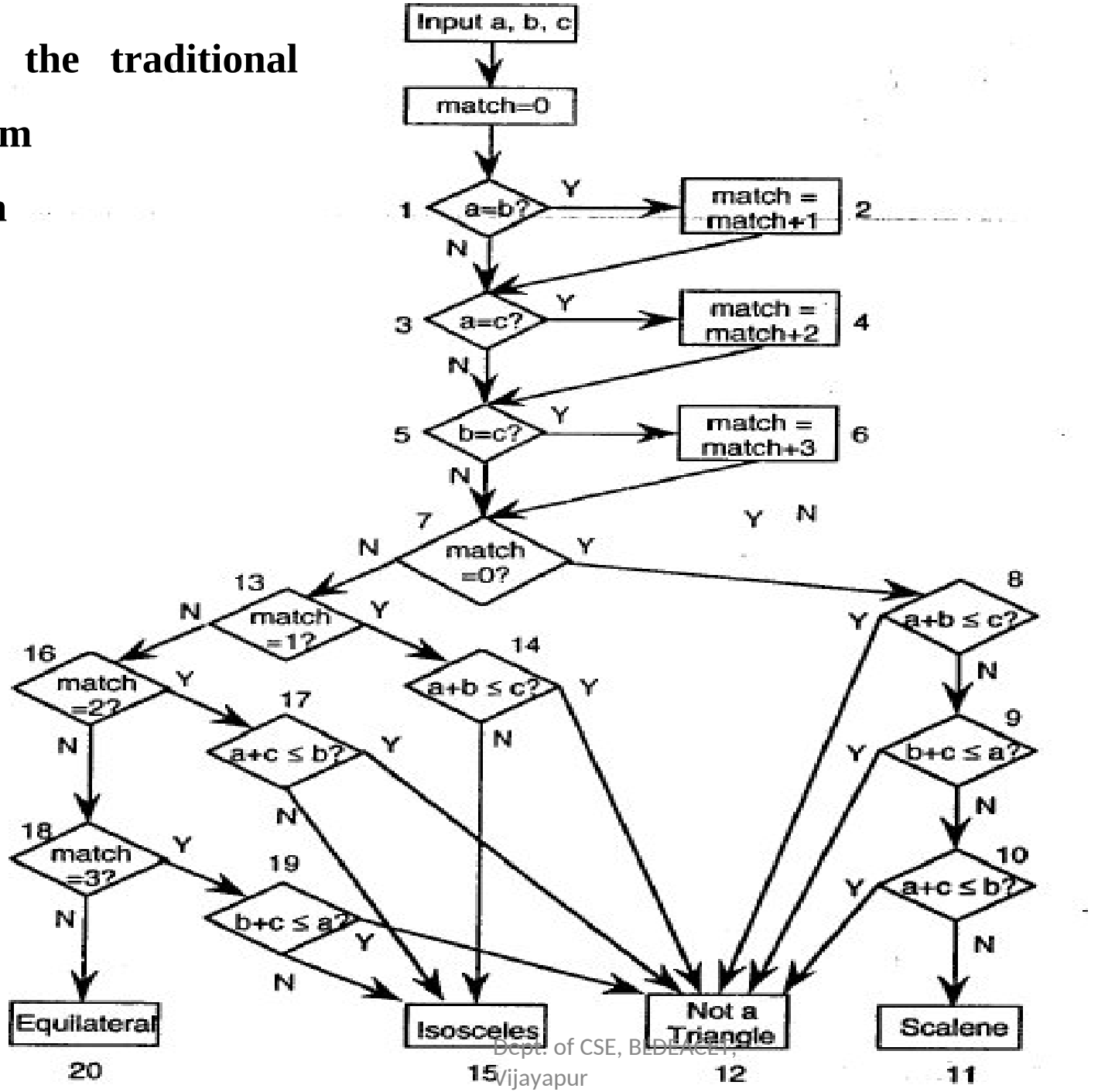
Weak Normal and Strong Normal Equivalence class Testing						
Case ID	Description	Input Data			Expected Output	Actual Output
		a	b	c		
WN1	Enter the equal value for a , b and c	5	5	5	Should display the message Equilateral triangle	
WN2	Enter the equal value for a and b and different for c	2	2	3	Should display the message Isosceles triangle	
WN3	Enter the diff. value for a , b and c	3	4	5	Should display the message Scalene triangle	
WN4	Enter the value for a , b and c such that it can not form a triangle	1	2	3	Message should be displayed can't form a triangle	

Weak Robust Equivalence Class Testing

Case ID	Description	Input Data			Expected output	Actual output
		a	b	c		
WR1	Enter one invalid input and two valid value for a , b and c	-1	5	5	Should display value of a is not in the range of permitted values	
WR2	Enter one invalid input and two valid value for a , b and c	5	-1	5	Should display value of b is not in the range of permitted values	
WR3	Enter one invalid input and two valid value for a , b and c	5	5	-1	Should display value of c is not in the range of permitted values	
WR4	Enter one invalid input and two valid value for a , b and c	11	5	5	Should display value of a is not in the range of permitted values	
WR5	Enter one invalid input and two valid value for a , b and c	5	11	5	Should display value of b is not in the range of permitted values	
WR6	Enter one invalid input and two valid value for a , b and c	5	5	11	Should display value of c is not in the range of permitted values	

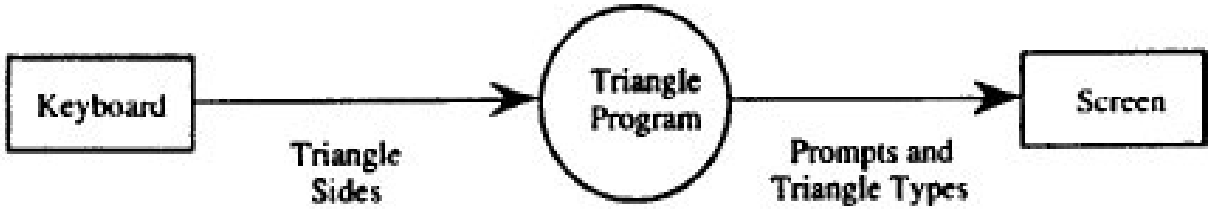
Strong Robust Equivalence Class Testing						
Case ID	Description	Input Data			Expected output	Actual output
		a	b	c		
SR1	Enter one invalid input and two valid value for a , b and c	-1	5	5	Should display value of a is not in the range of permitted values	
SR2	Enter one invalid input and two valid value for a , b and c	5	-1	5	Should display value of b is not in the range of permitted values	
SR3	Enter one invalid input and two valid value for a , b and c	5	5	-1	Should display value of c is not in the range of permitted values	
SR4	Enter two invalid input and two valid value for a , b and c	-1	-1	5	Should display value of a and b is not in the range of permitted values	
SR5	Enter two invalid input and two valid value for a , b and c	5	-1	-1	Should display value of b and c is not in the range of permitted values	
SR6	Enter two invalid input and two valid value for a , b and c	-1	5	-1	Should display value of a and c is not in the range of permitted values	
SR7	Enter all invalid inputs	-1	-1	-1	Should display value of a, b and c is not in the range of permitted values	

Flowchart for the traditional triangle program implementation



- **Traditional Implementation**

  Fortran-like style

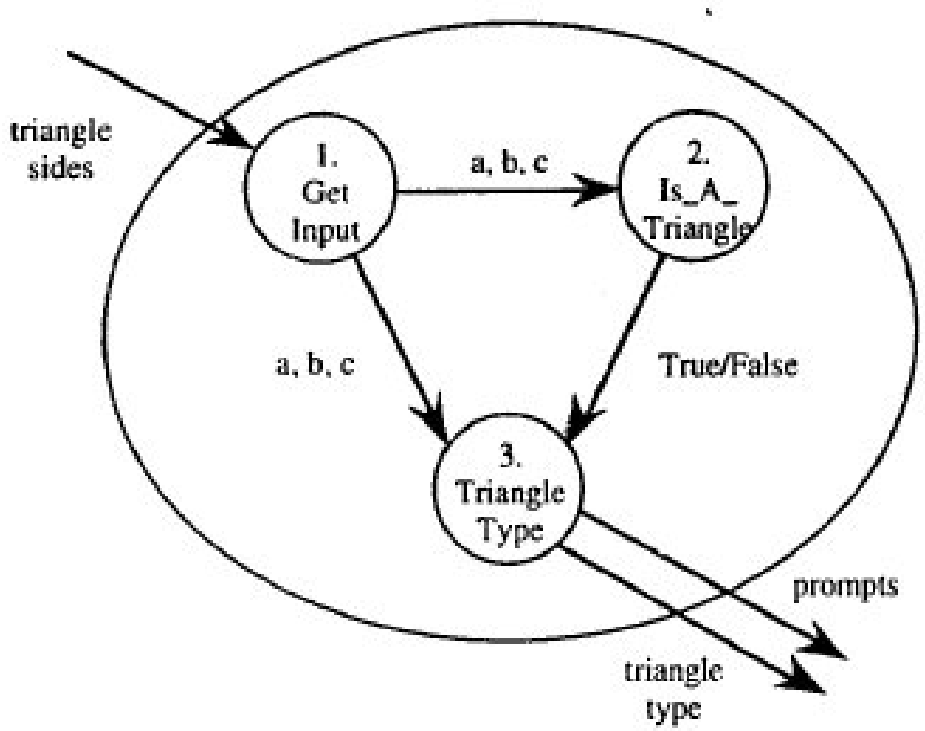


- **Structured Implementation**

  DFD – Data Flow Diagram

Structures Programming

For Simple version and Improved version



Dataflow diagram for a structured triangle program implementation.

Problems persist!

What output is expected for the input set (2, 2, 5)?

- **Isosceles because a = b?**
- **NotATriangle because c > a+b?**

NextDate

NextDate is a function of three variables: month, date, and year. It returns the date of the day after the input date. The month, date, and year variables have integer values subject to these conditions:

c1. $1 \leq month \leq 12$
c2. $1 \leq day \leq 31$
c3. $1812 \leq year \leq 2012$

If any of conditions c1, c2, or c3 fails, NextDate produces an output indicating the corresponding variable has an out-of-range value — for example, “Value of month not in the range 1..12”. Because numerous invalid day–month–year combinations exist, NextDate collapses these into one message: “Invalid Input Date.”

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
c1: month in	M1	M1	M1	M1	M1	M2	M2	M2	M2	M2
c2: day in	D1	D2	D3	D4	D5	D1	D2	D3	D4	D5
c3: year in	—	—	—	—	—	—	—	—	—	—
a1: impossible					X					
a2: increment day	X	X	X			X	X	X	X	
a3: reset day				X						X
a4: increment month				X						X
a5: reset month										
a6: increment year										

The Commission Problem

A rifle salesperson in the former Arizona Territory sold rifle locks, stocks, and barrels made by a gunsmith in Missouri. Locks cost $45, stocks cost $30, and barrels cost $25. The salesperson had to sell at least one complete rifle per month, and production limits were such that the most the salesperson could sell in a month was 70 locks, 80 stocks, and 90 barrels. After each town visit, the salesperson sent a telegram to the Missouri gunsmith with the number of locks, stocks, and barrels sold in that town. At the end of a month, the salesperson sent a very short telegram showing –1 locks sold. The gunsmith then knew the sales for the month were complete and computed the salesperson's commission as follows: 10% on sales up to (and including) $1000, 15% on the next $800, and 20% on any sales in excess of $1800. The commission program produced a monthly sales report that gave the total number of locks, stocks, and barrels sold, the salesperson's total dollar sales, and, finally, the commission.

Conditions	Condition entries (Rules)					
C1: 1≤locks≤70?	F	T	T	T	T	T
C2: 1≤stocks≤80?	--	F	T	T	T	T
C3: 1≤barrels≤90?	--	--	F	T	T	T
C4:sales>1800?	--	--	--	T	F	F
C5:sales>1000?	--	--	--	--	T	F
C6:sales≤1000?	--	--	--	--	--	T
Actions	Action entries					
A1:com1=0.10*sales						X
A2:com2=com1+0.15*(sales-1000)					X	
A3:com3=com2+0.20*(sales-1800)				X		
A4:out of range	X	X	X			

Test Case Name : Boundary Value for Commission Problem

Experiment Number : 2

Test data : price Rs for lock - 45.0 , stock - 30.0 and barrel - 25.0

sales = total lock * lock price + total stock * stock price + total barrel * barrel price

commission : 10% up to sales Rs 1000 , 15 % of the next Rs 800 and 20 % on any sales in excess of 1800

Pre-condition : lock = -1 to exit and 1< =lock < = 70 , 1<=stock <=80 and 1<=barrel<=90

Brief Description : The salesperson had to sell at least one complete rifle per month.

Case	Locks	Stocks	Barrels	Sales	Comm	Comment
1	1	1	1	100	10	output minimum
2	1	1	2	125	12.5	output minimum +
3	1	2	1	130	13	output minimum +
4	2	1	1	145	14.5	output minimum +
5	5	5	5	500	50	midpoint
6	10	10	9	975	97.5	border point –
7	10	9	10	970	97	border point –
8	9	10	10	955	95.5	border point –
9	10	10	10	1000	100	border point
10	10	10	11	1025	103.75	border point +
11	10	11	10	1030	104.5	border point +
12	11	10	10	1045	106.75	border point +
13	14	14	14	1400	160	midpoint
14	18	18	17	1775	216.25	border point –
15	18	17	18	1770	215.5	border point –
16	17	18	18	1755	213.25	border point –
17	18	18	18	1800	220	border point
18	18	18	19	1825	225	border point +
19	18	19	18	1830	226	border point +
20	19	18	18	1845	229	border point +
21	48	48	48	4800	820	midpoint
22	70	80	89	7775	1415	output maximum –
23	70	79	90	7770	1414	output maximum –
24	69	80	90	7755	1411	output maximum –
25	70	80	90	7800	1420	output maximum

Test Cases (Weak Robust)

Test Case	Test Case Description	Lock	Stock	Barrel	Sales	Expected Output	ACTUAL OUTPUT
1	WR1	10	10	10	1000	100	100
2	WR2	-1	40	45	-	Program Terminates	Program Terminates
3	WR3	-2	40	45	-	Value of locks not in range of 1...70	Value of locks not in range of 1...70
4	WR4	71	40	45	-	Value of locks not in range of 1...70	Value of locks not in range of 1...70
P5	WR5	35	-1	45	-	Value of stock not in 1...80	Value of stock not in 1...80
P6	WR6	35	81	45	-	Value of stock not in 1...85	Value of stock not in 1...85
P7	WR7	10	9	10	970	97	97

Test Cases (Strong Robust)

Test Case	Test Case Description	Lock	Stock	Barrel	Sale	Expected Output	Actual Output
1	SR1	-2	40	45	-	Values of locks not 1...70	Values of locks not 1...70
2	SR2	35	-1	45	-	Values of stocks not in 1...80	Values of stocks not in 1...80
3	SR3	35	40	-2	-	Values of barrels not In 1...90	Values of barrels not In 1...90
4	SR4	-2	-1	45	-	Values of locks not	Values of
5	SR5	-2	40	1	-	Values of locks not in 1...70 Values of barrels not in 1...90	Values of locks not in 1...70 Values of barrels not in 1...90
6	SR6	35	-1	-1	-	Values of stocks not in 1...80 Values of barrels not in 1...90	Values of stocks not in 1...80 Values of

The Simple ATM (SATM) System

- Deliberately simple (only 15 screens)
- Very familiar example
- Full description in text

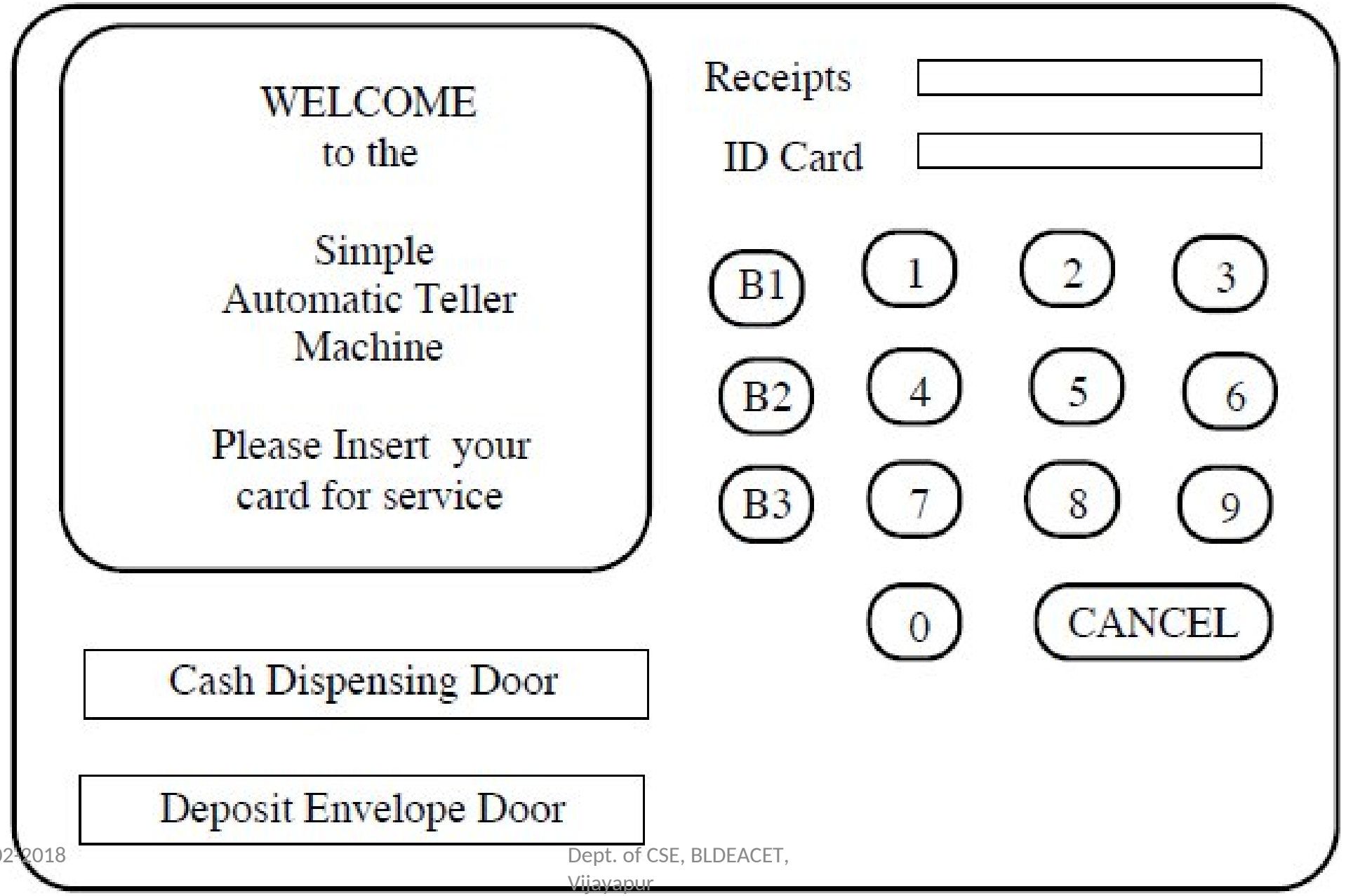
Customers can select any of 3 transaction types:

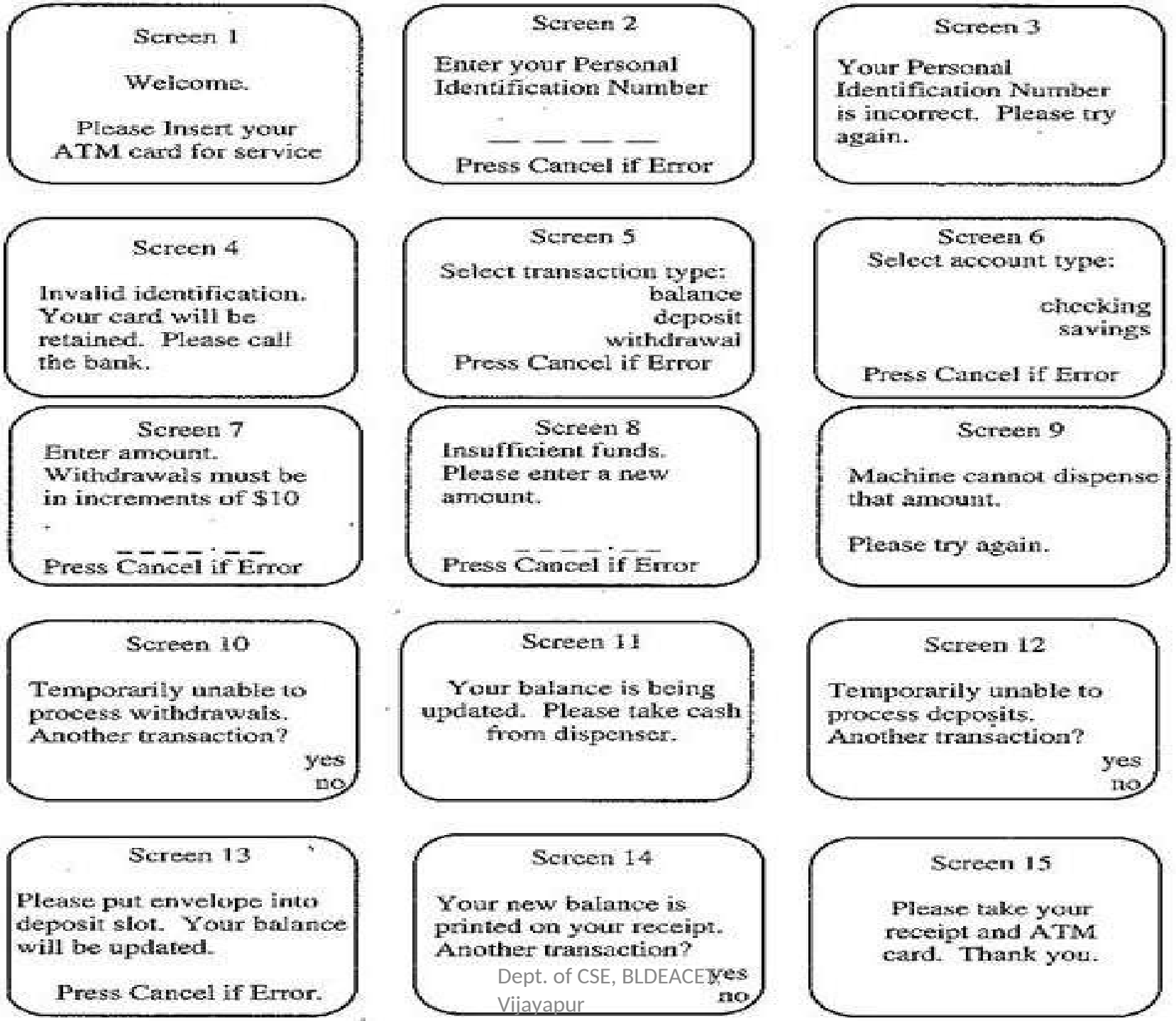
- Deposits
- Withdrawals
- Balance inquires

Transactions can be done on two types of accounts:

- Checking
- Savings

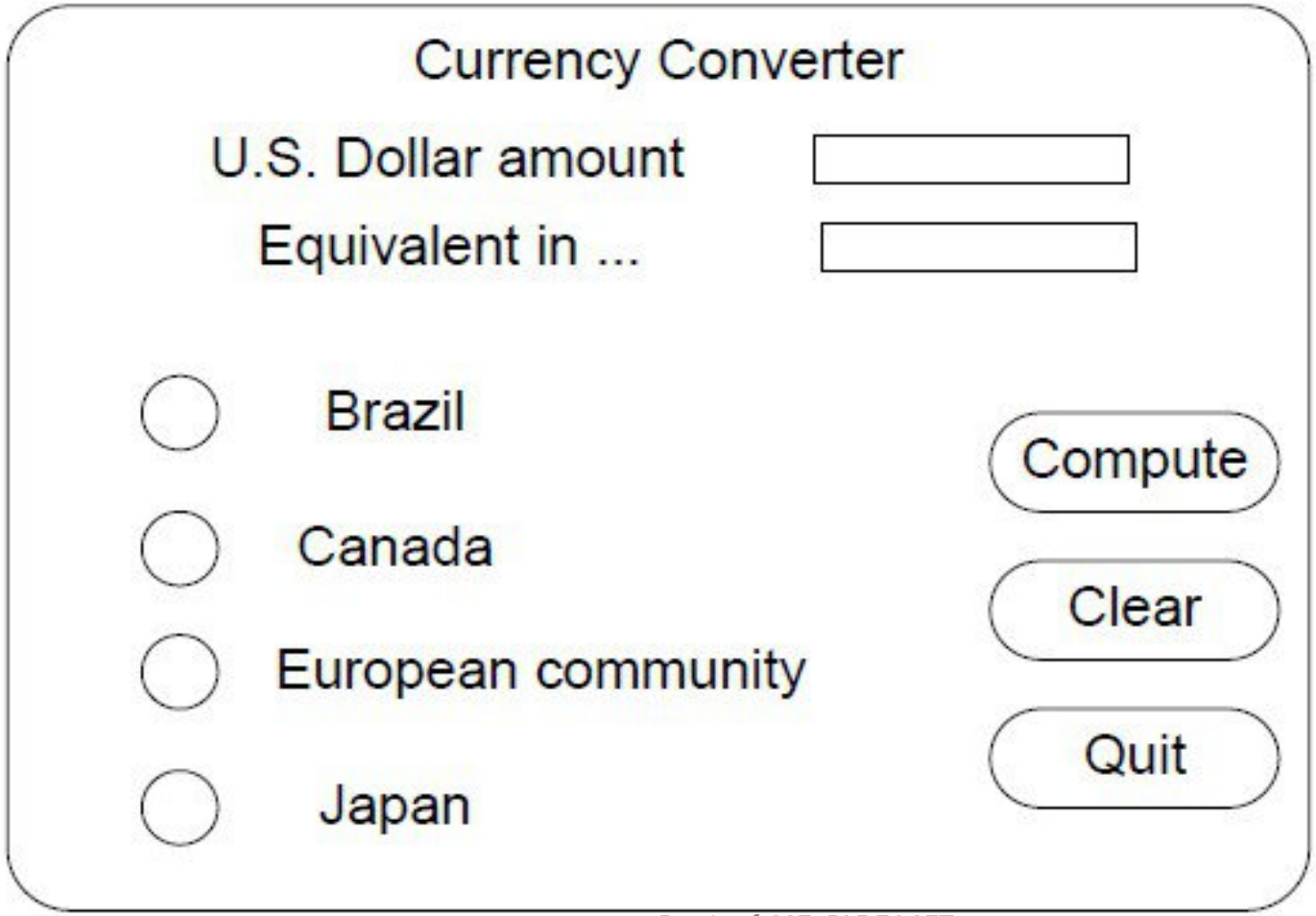
The Simple ATM System




Screen 1
Welcome.
Please Insert your ATM card for service
Screen 2
Enter your Personal Identification Number
Press Cancel if Error
Screen 3
Your Personal Identification Number is incorrect. Please try again.
Screen 4
Invalid identification. Your card will be retained. Please call the bank.
Screen 5
Select transaction type:
balance
deposit
withdrawal
Press Cancel if Error
Screen 6
Select account type:
checking
savings
Press Cancel if Error
Screen 7
Enter amount.
Withdrawals must be in increments of $10
Press Cancel if Error
Screen 8
Insufficient funds.
Please enter a new amount.
Press Cancel if Error
Screen 9
Machine cannot dispense that amount.
Please try again.
Screen 10
Temporarily unable to process withdrawals.
Another transaction?
yes
no
Screen 11
Your balance is being updated. Please take cash from dispenser.
Screen 12
Temporarily unable to process deposits.
Another transaction?
yes
no
Screen 13
Please put envelope into deposit slot. Your balance will be updated.
Press Cancel if Error.
Screen 14
Your new balance is printed on your receipt.
Another transaction?
yes
no
Screen 15
Please take your receipt and ATM card. Thank you.

The Currency Converter

The currency conversion program is another event-driven program that emphasizes code associated with a graphical user interface (GUI).



The Saturn Windshield Wiper Controller

The windshield wiper on some Saturn automobiles is controlled by a lever with a dial. The lever has four positions, OFF, INT (for intermittent), LOW, and HIGH, and the dial has three positions, numbered simply 1, 2, and 3. The dial positions indicate three intermittent speeds, and the dial position is relevant only when the lever is at the INT position. The decision table below shows the windshield wiper speeds (in wipes per minute) for the lever and dial positions.

c1. Lever	OFF	INT	INT	INT	LOW	HIGH
c2. Dial	n/a	1	2	3	n/a	n/a
a1. Wiper speed is…	0	4	6	12	30	60


MIST
OFF
INT
LO
HI
FRONT
INT
OFF
INT
ON
REAR

References

Paul C. Jorgensen: Software Testing, A Craftsman's Approach, 3rd Edition, Auerbach Publications, 2008.

Thank You