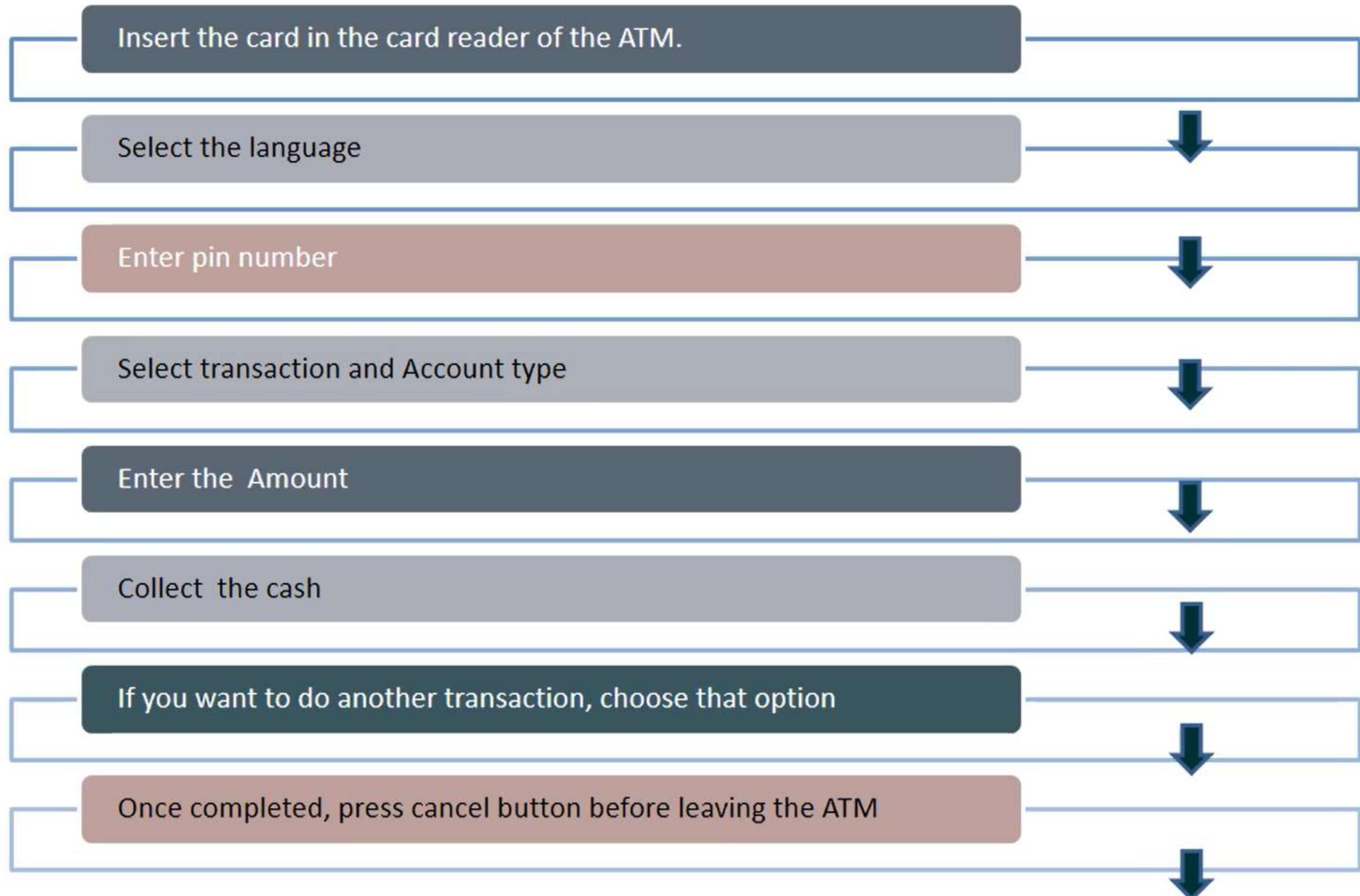


LOGIC DEVELOPMENT





- Here our problem statement is

Withdrawal of cash from ATM

- To solve it, we followed some instructions in a specific order, which is nothing but an ALGORITHM

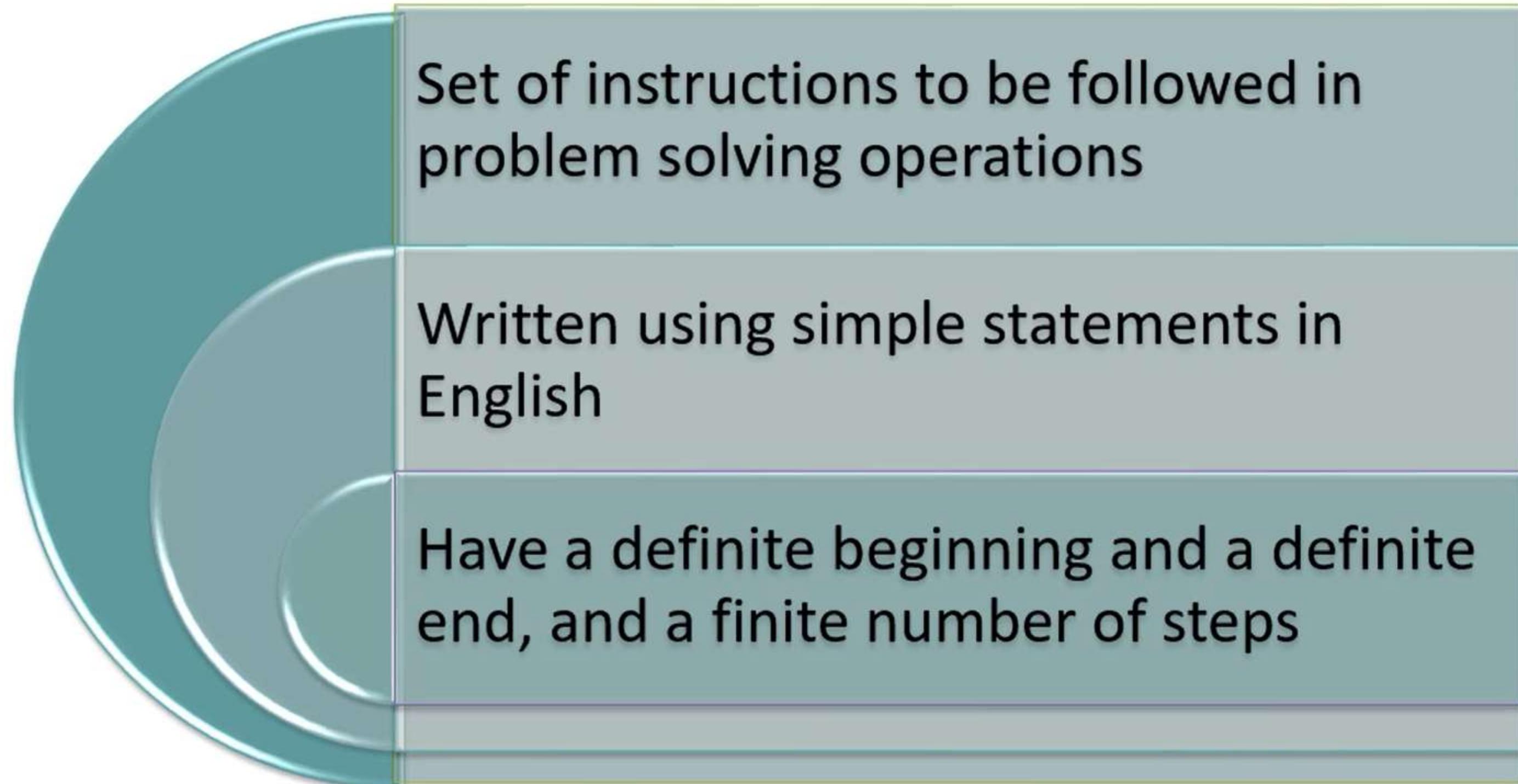


In this module you will learn

- Introduction to Algorithms
- Variables, constants and Expressions
- Operators
- Flowcharts
- Pseudo codes



WHAT IS AN ALGORITHM?



SAMPLE ALGORITHM FOR MAKING TEA

Step by step instructions for making a cup of tea

Get the ingredients - milk, tea leaves, sugar, water.

Boil water in a kettle. When the water boils, add tea leaves, milk.

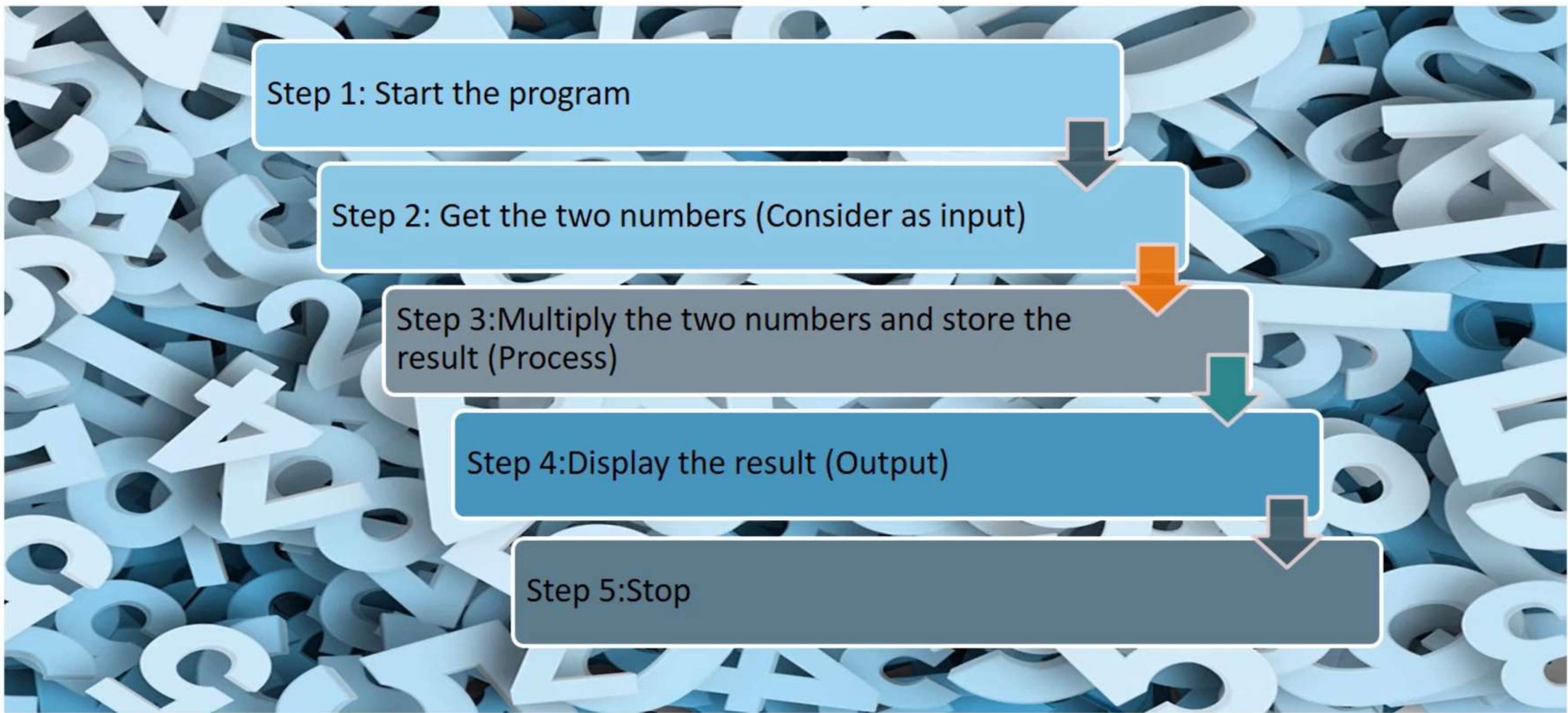
Leave the kettle on the stove for about 2 minutes.

Strain out the leaves; add sugar and stir.

Hot Tea is Ready



ALGORITHM TO MULTIPLY 2 NUMBERS



DEFINING VARIABLES, CONSTANTS & EXPRESSIONS

An algorithm consists of variables, constants and Expressions

VARIABLES

Variable is the name of a memory location, whose value keeps on changing. Variables are containers to store values.



EXPRESSIONS

It is a combination of operand and operators. Operand can be a variable or constant



CONSTANTS

Constant is a fixed value or identifier which cannot be changed. It is opposite to variables.,

BASIC OPERATORS AND OPERATIONS

Operator is a symbol that represents an action or process



There are several classifications of operators and each of them can have one or more operands, a specific data that is to be manipulated.

Basic classification of operators are

- Assignment Operator
- Arithmetic Operators
- Relational Operators
- Logical Operators etc.

BASIC OPERATORS AND OPERATIONS

Operator is a symbol that represents an action or process

There are several classifications of operators and each of them can have one or more operands, a specific data that is to be manipulated.

Basic classification of operators are

- Assignment Operator
- Arithmetic Operators
- Relational Operators
- Logical Operators etc.

ASSIGNMENT OPERATOR

Used to assign a value to the variable

Assignment Symbol (\leftarrow or $=$)

- Example 1:

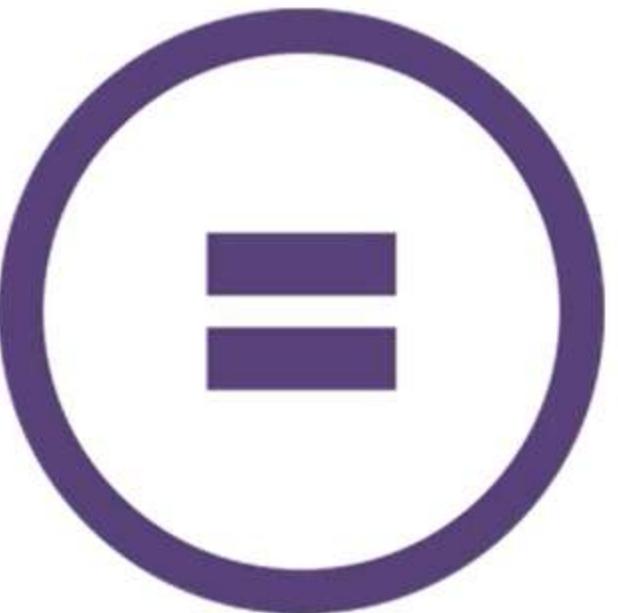
HEIGHT \leftarrow 5 (or) HEIGHT = 5

Assigns value 5 to the variable HEIGHT,
statement is

- Example 2:

C=A+B

The sum value of variable A and variable B is
assigned to the variable C.



ARITHMETIC OPERATORS

Used to perform arithmetic operations like addition, subtraction, multiplication, division etc.

Some of the basic arithmetic operators and its examples

Operator	Meaning	Example
+	Addition	$A + B$
-	Subtraction	$A - B$
*	Multiplication	$A * B$
/	Division	A / B
$^$	Power(Exponentiation)	A^3 for A power 3
$\%$	Remainder (Modulo)	$A \% B$

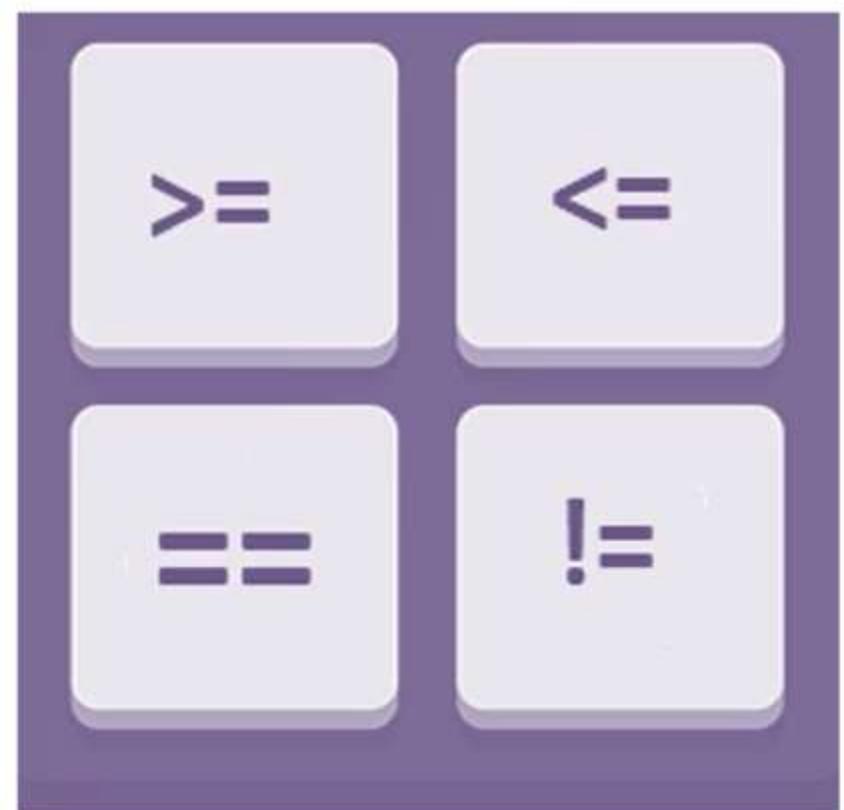


RELATIONAL OPERATORS

Used to evaluate the value in the variables

Some of the basic relational operators and its examples

Operator	Meaning	Example
<	Less than	A < B
<=	Less than or equal to	A <= B
==	Equal to (Comparison)	A == B
!=	Not Equal to	A != B
>	Greater than	A > B
>=	Greater than or equal to	A >= B

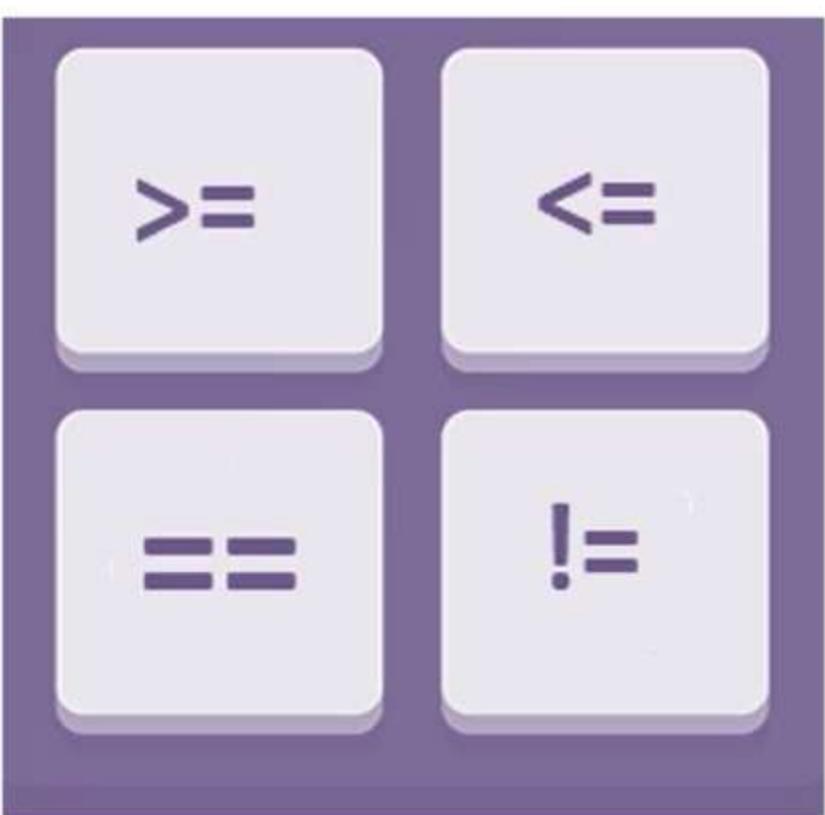


RELATIONAL OPERATORS

Used to evaluate the value in the variables

Some of the basic relational operators and its examples

Operator	Meaning	Example
<	Less than	A < B
<=	Less than or equal to	A <= B
==	Equal to (Comparison)	A == B
!=	Not Equal to	A != B
>	Greater than	A > B
>=	Greater than or equal to	A >= B

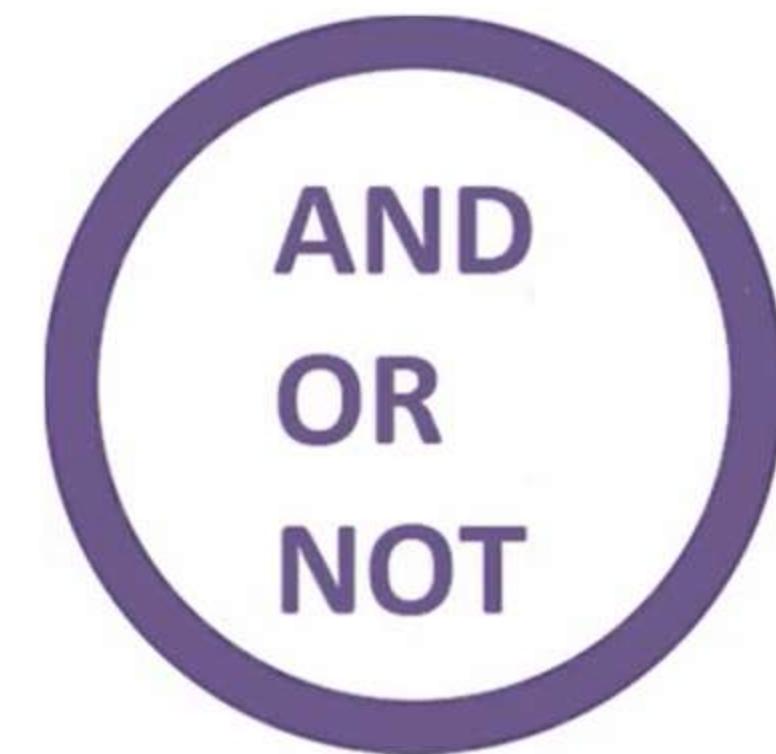


LOGICAL OPERATORS

Used to perform logical operations in the given expressions.

Some of the basic logical operators and its examples

Operator	Example	Meaning
AND	$A < B$ AND $B < C$	Result is True if both $A < B$ and $B < C$ are true else false
OR	$A < B$ OR $B < C$	Result is True if either $A < B$ or $B < C$ are true else false
NOT	NOT ($A > B$)	Result is True if $A > B$ is false else true



BEST PRACTICES TO BE KNOWN

Some important rules to be followed, when using a variable

A variable name must define the exact explanation of its content. Use meaningful variable names by relating them with the purpose of the program and what it's for.

- Avoid variable names starting with numbers & symbols, except underscore(_).
- Avoid using reserved keywords like if, else
- Single variable name should not have space in between



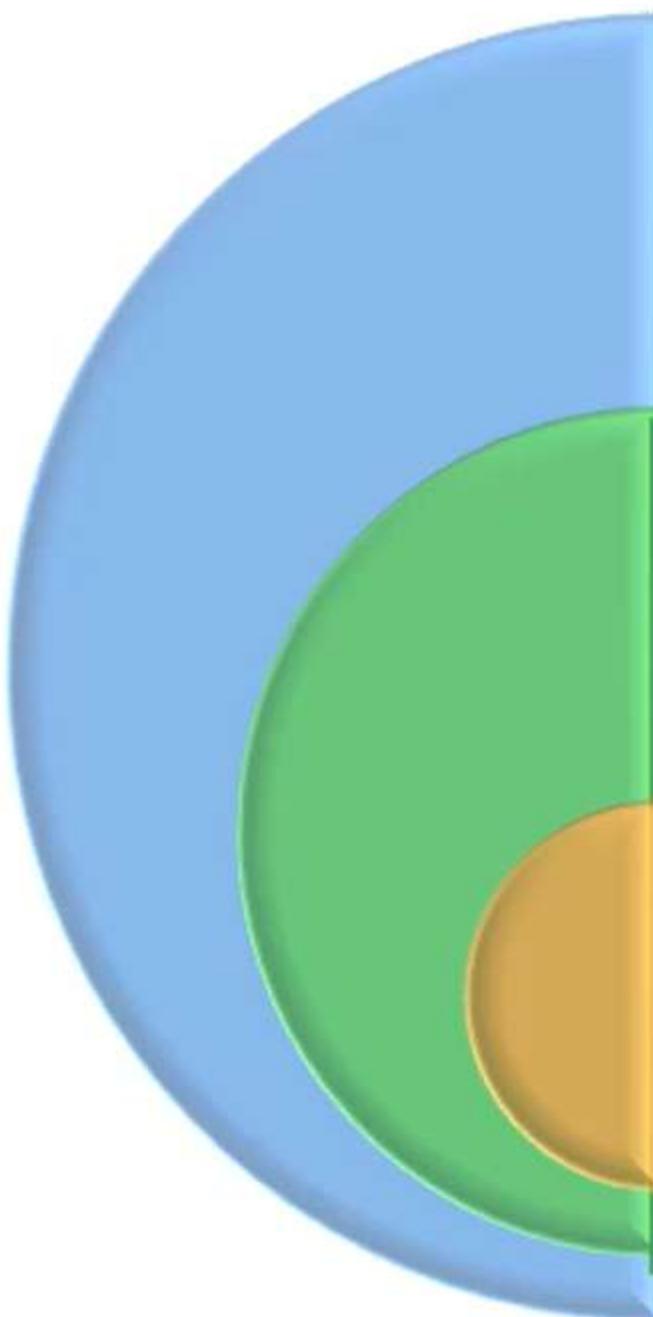
Example:
`age`
`height`
`roll_no`
`reg_number`

Meaningful variable
names

Example:
`123`
`Xyz`
`#x1`
`User name`

Bad practice of naming
variables

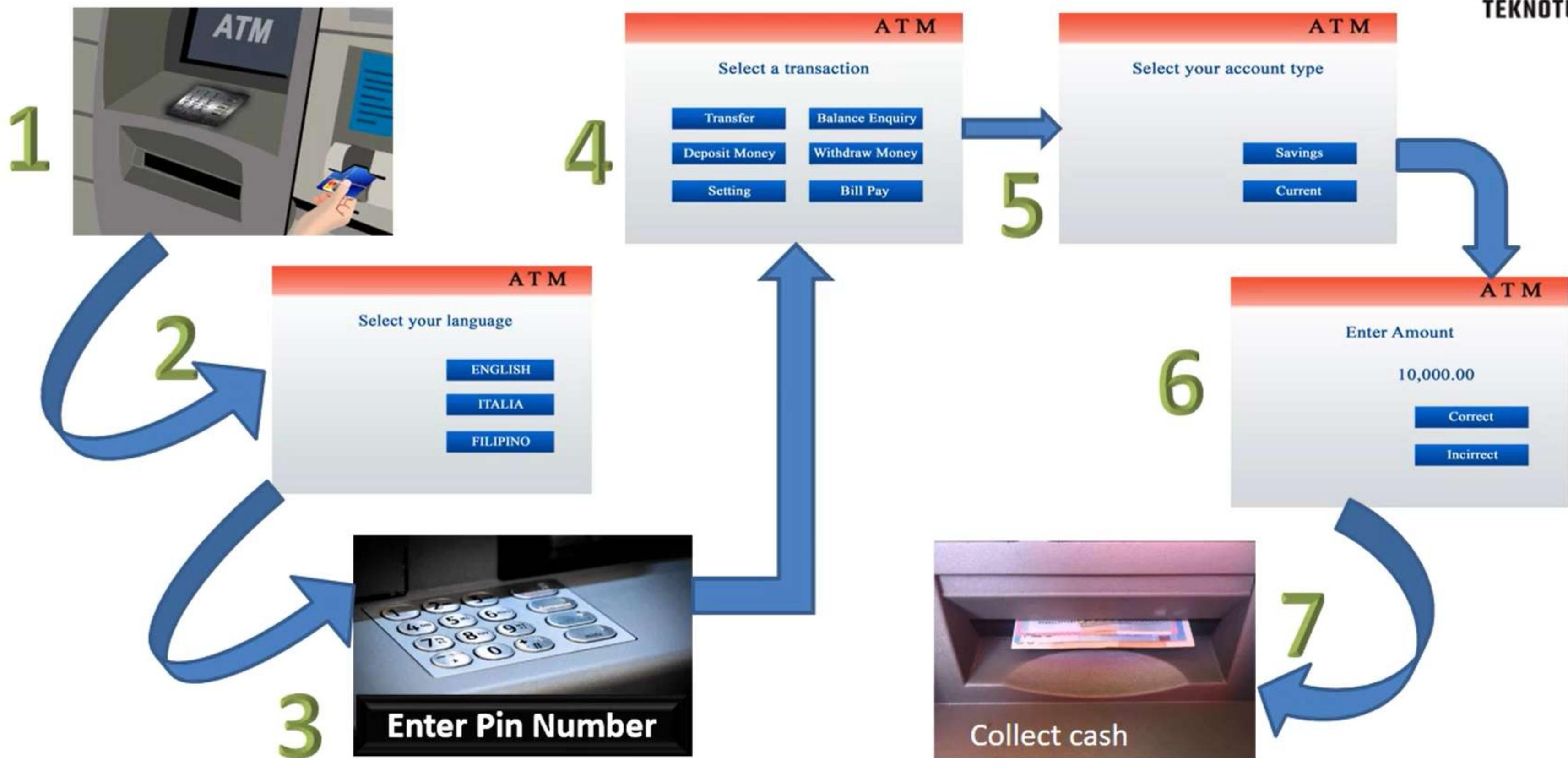
WHAT IS A FLOWCHART?



A diagrammatic representation of an algorithm, workflow or process

Lines and arrows show the sequence of steps and relationship
among them

It is a Business Power Tool



SYMBOLS IN FLOWCHART



START/END



CONNECTOR



INPUT/OUTPUT



FLOW DIRECTION

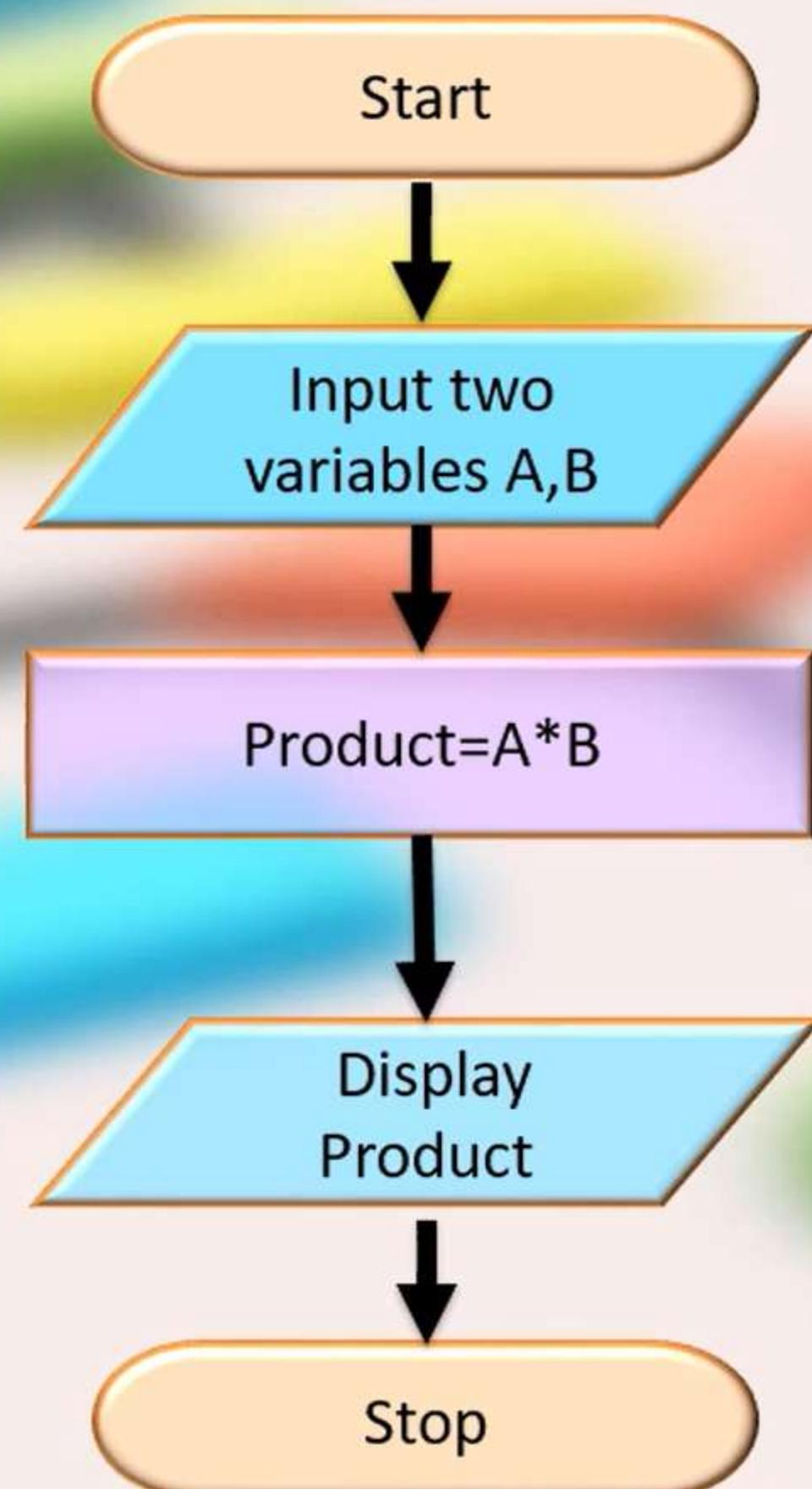


PROCESS



DECISION

SAMPLE FLOWCHART TO MULTIPLY 2 NUMBERS



PSEUDOCODE



It is an Informal way of describing algorithms



Does Not require any strict programming language syntax



It is Easily readable and modular form

SAMPLE PSEUDOCODE FOR MAKING TEA

Pseudocode representation for making a cup of tea

BEGIN

GET water, tea leaves, sugar

PLUG IN kettle

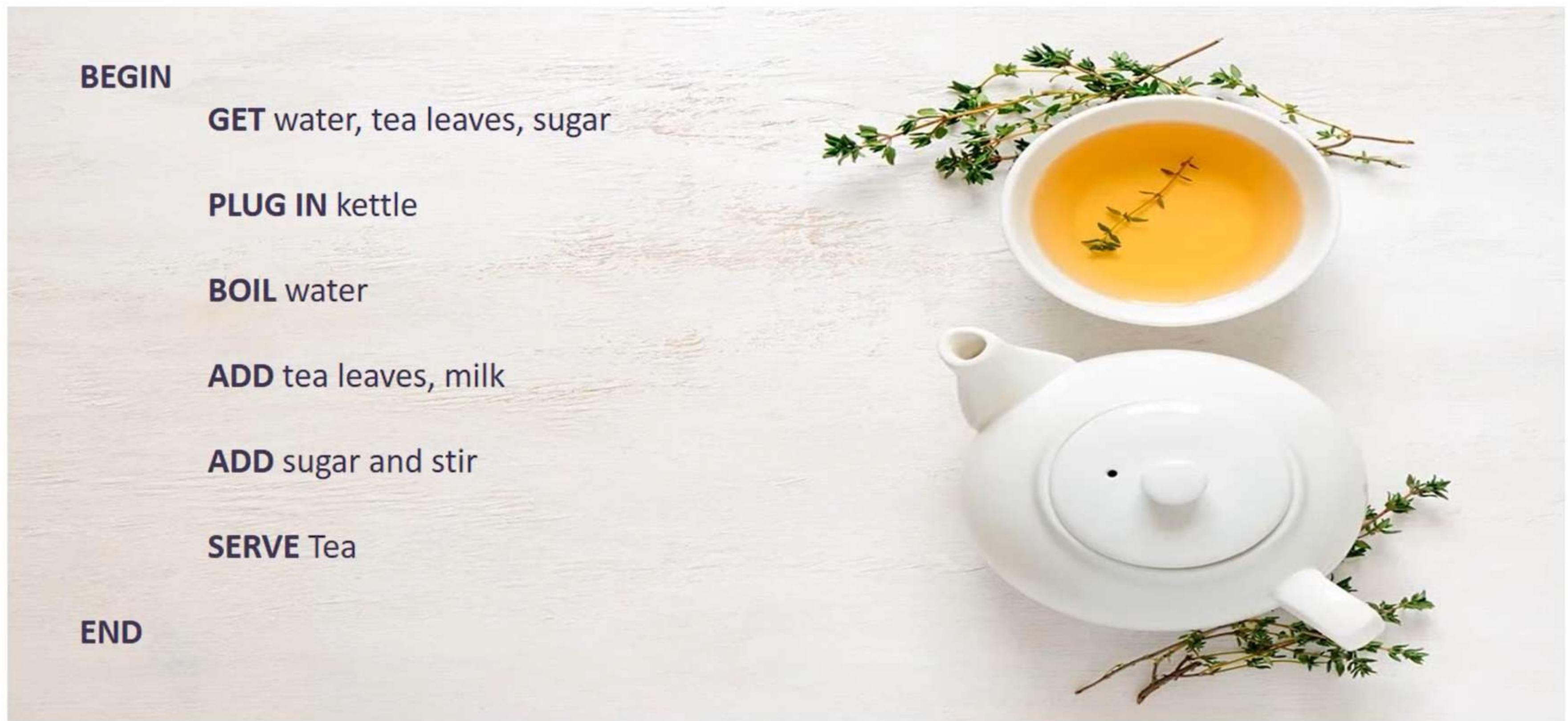
BOIL water

ADD tea leaves, milk

ADD sugar and stir

SERVE Tea

END



PSEUDOCODE TO MULTIPLY TWO NUMBERS



Best- Practice:
Usage of indentation
improves readability &
conveys a better structure

PSEUDOCODE TO MULTIPLY TWO NUMBERS



```
BEGIN  
DECLARE variables number1, number2,Product  
READ variables number1, number2  
SET Product=number1*number2  
PRINT Product  
END
```

Best- Practice:
Usage of indentation
improves readability &
conveys a better structure

Summary

- Introduction to algorithm
- Variables, constants and expressions
- Operators
- Flowcharts and Pseudo codes



LOGIC DEVELOPMENT



In this module you will learn

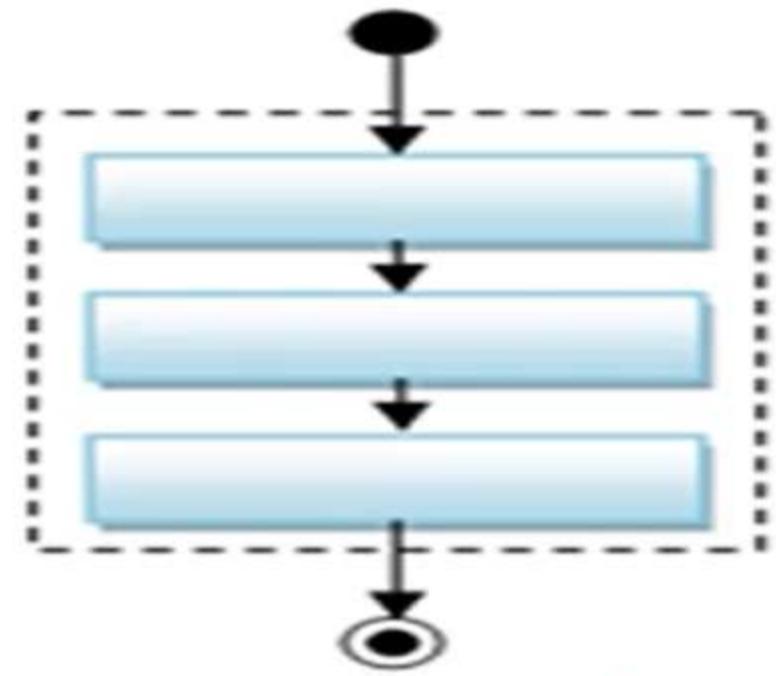
- The flow of program
- Introduction to selection statements
- Categories of selection statements
- How to construct algorithm, flowchart and pseudo codes
for selection statements



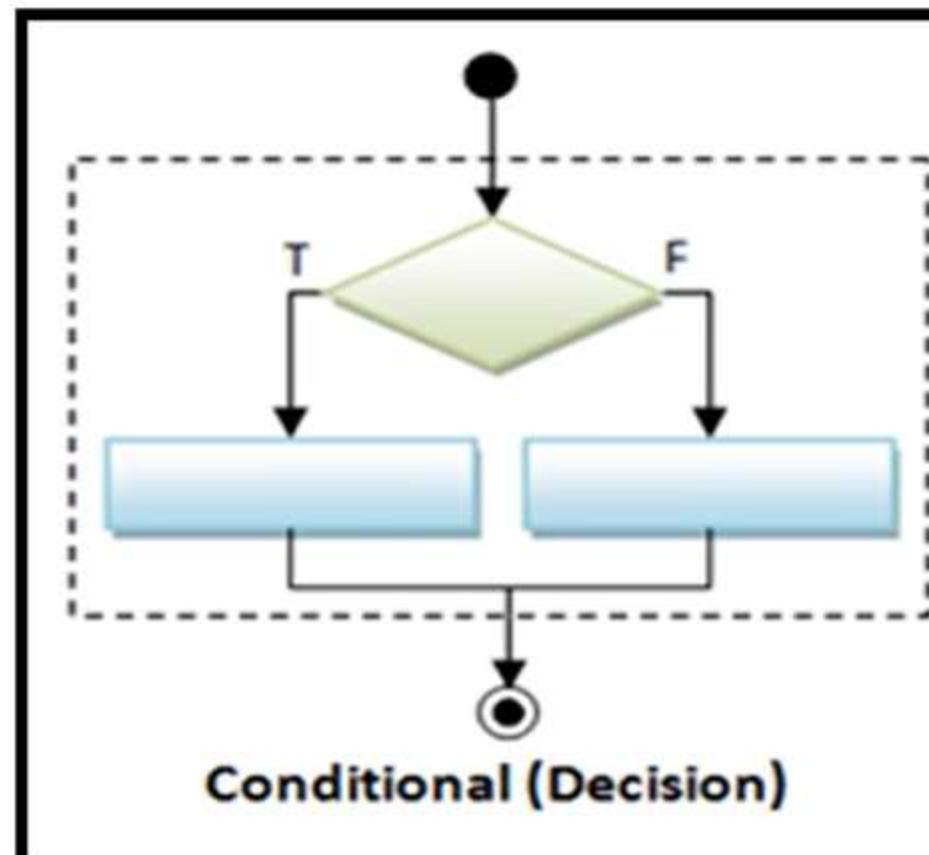
FLOW OF A PROGRAM

There's an order in which the computer executes the statements in a program.

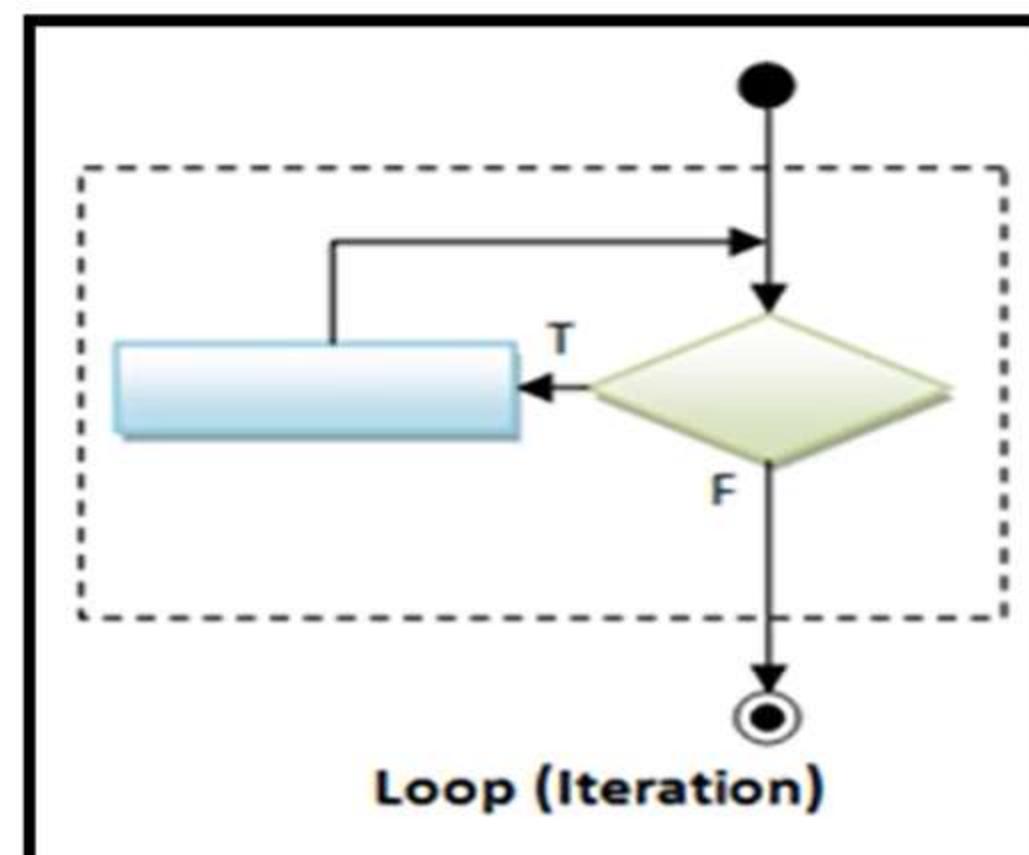
The orders even be combined to deal with a given problem.



Sequential

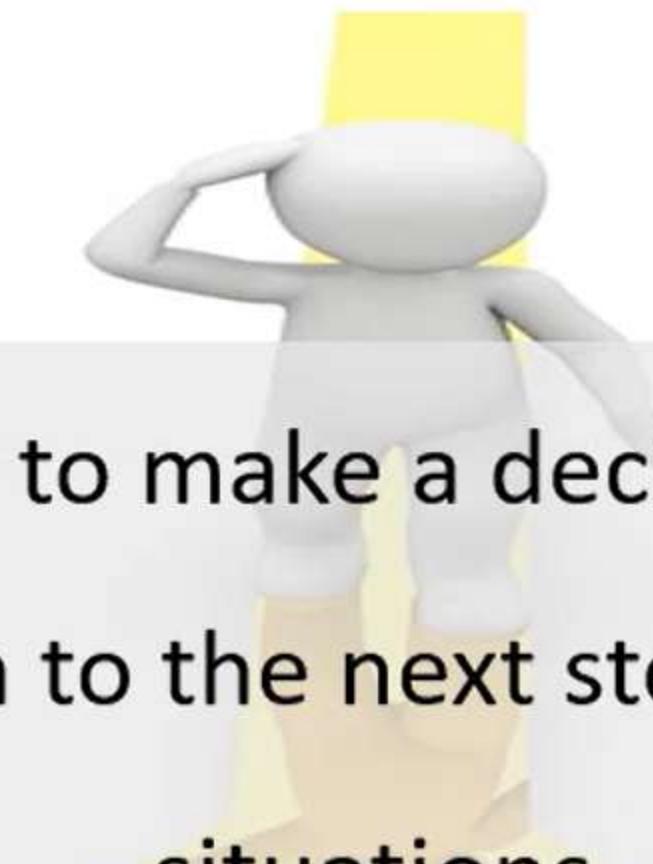


Conditional (Decision)



Loop (Iteration)

SELECTION OR CONDITIONAL STATEMENTS

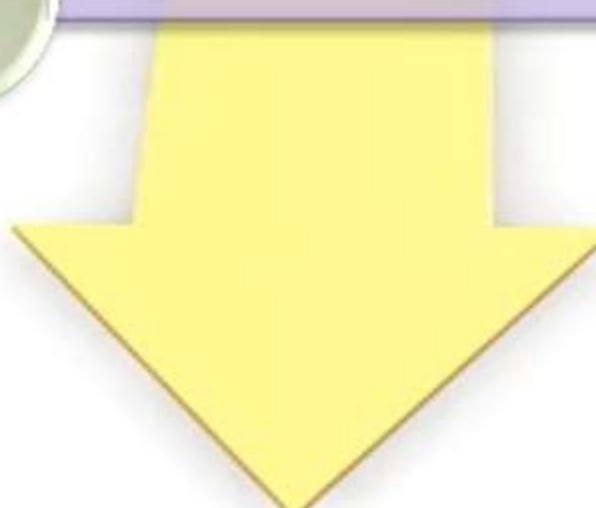


It might be necessary to make a decision before arriving at a conclusion or to go on to the next step of processing in many situations.

SELECTION OR CONDITIONAL STATEMENTS

**Selection
statements can
be categorized
as**

- Simple If:
- If-else
- Else If - ladder
- Nested-if

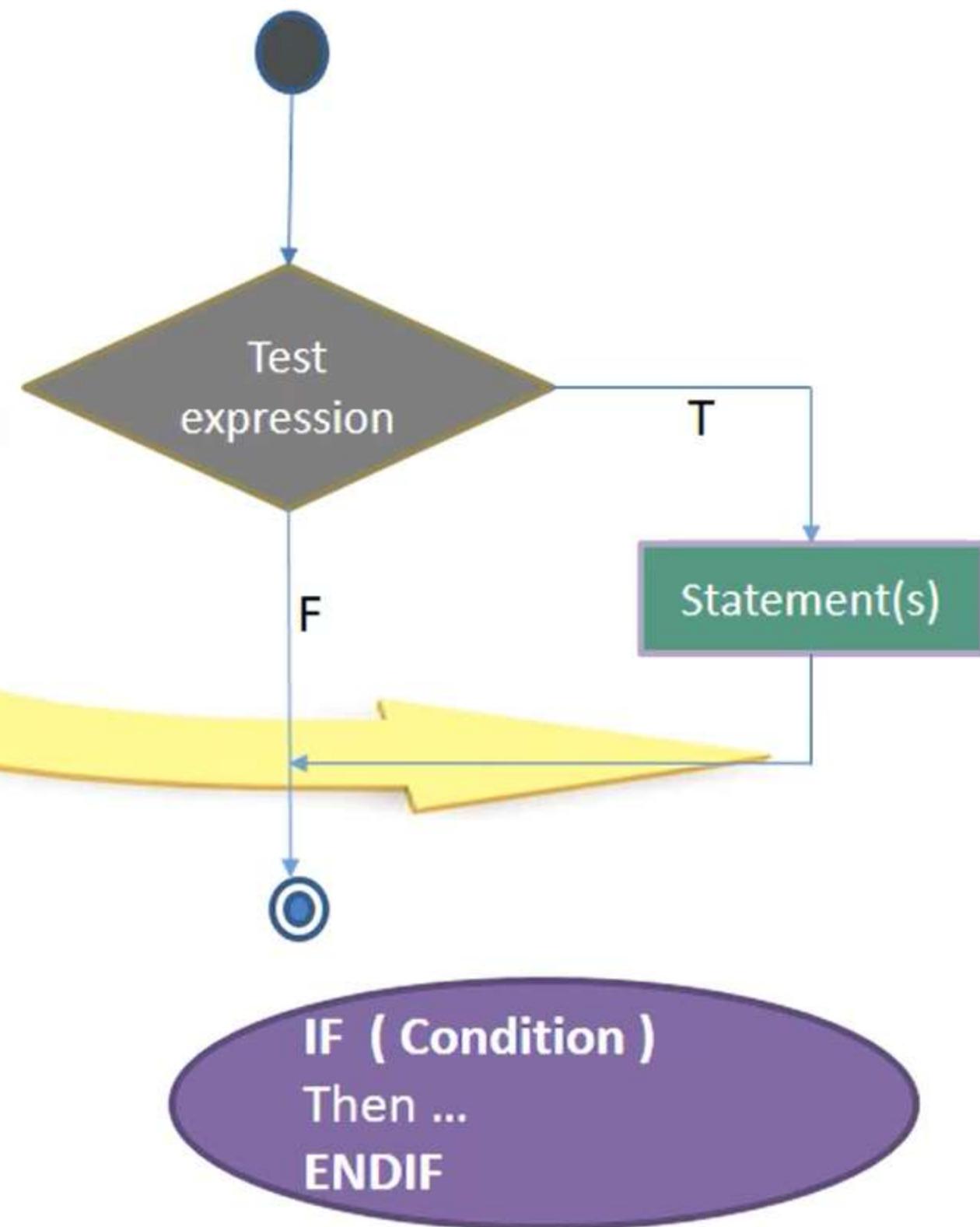


SELECTION OR CONDITIONAL STATEMENTS-SIMPLE IF

Decides the sequence of execution of instructions based on the condition enclosed in simple if statement

Used for branching when a single condition is to be checked

- If the condition is true, the statements inside the if block are executed and moves sequentially to the next statement
- If the condition is false, the true block statements are skipped and the next statement after simple if statement gets executed



ALGORITHM FOR SIMPLE-IF

Sample algorithm to print the square of a number if it is less than 6.

 STEP 1:
Start the process

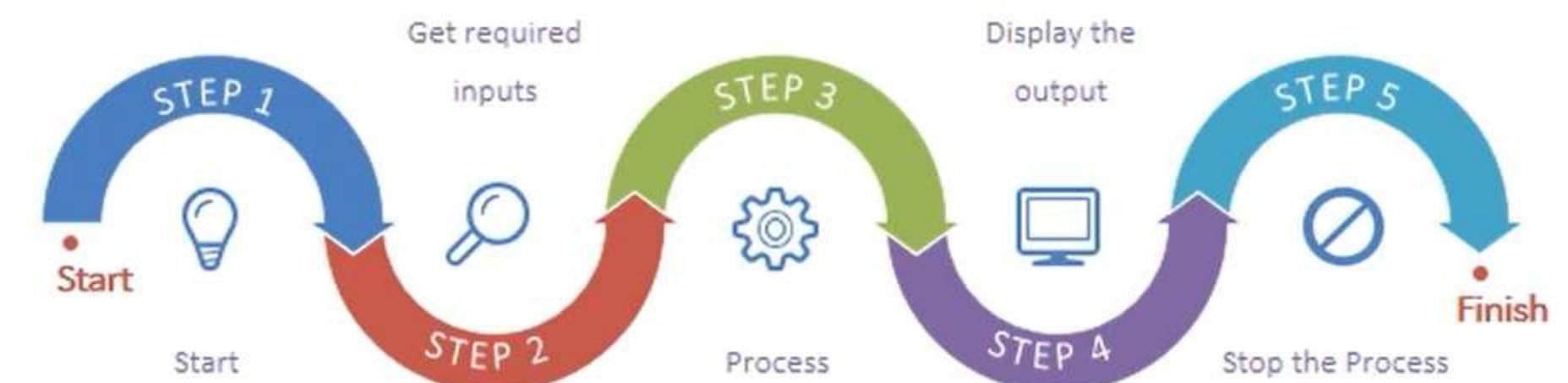
 STEP 2:
Get the number

 STEP 3:
Check whether number is less than 6, if YES GOTO step 4,
if NO GOTO step 6

 STEP 4:
Squares the number and store the result in variable
square

 STEP 5:
Display the square value

 STEP 6:
Stop the process

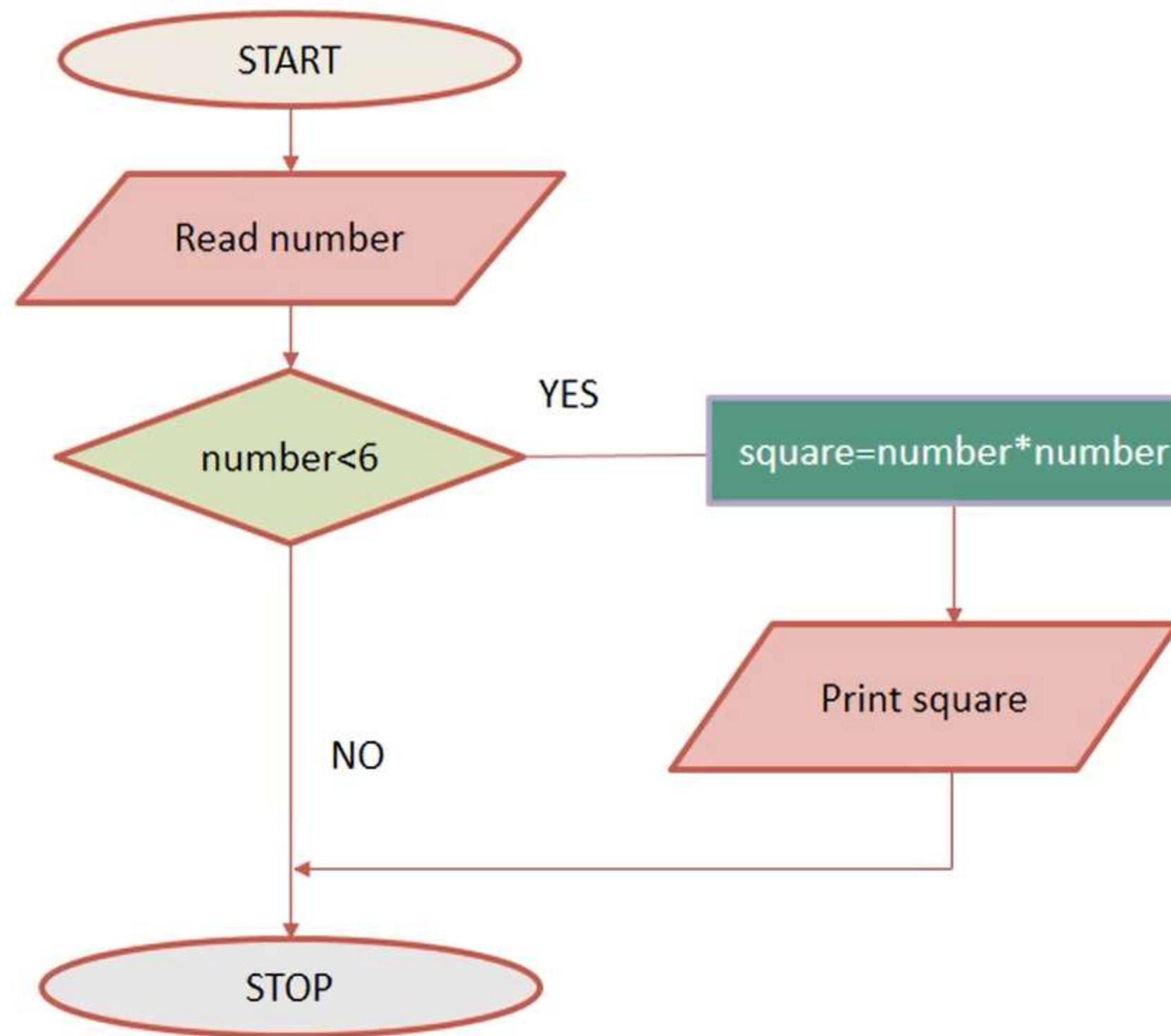


Example algorithm for
Simple if logic

GOTO statement
transfers control of
execution to
another statement

FLOWCHART FOR SIMPLE-IF

Basic Flowchart representation to print the square of a number if it is less than 6



(Start the process)

(Let the variable be a number and the value for num=5)

(This step checks whether 5 is less than 6. As 5 is less than 6, it squares the number ($5*5$)

(Displays 25)

(Ends the process)

PSEUDOCODE FOR SIMPLE-IF

Pseudocode representation to print the square of a number if it is less than 6

BEGIN



DECLARE variables number, square



READ number



IF number<6



THEN

SET square ← number * number

PRINT square

ENDIF



END

SIMPLE IF –DRY RUN

Manual execution of steps in an algorithm is called as dry run. Sample dry run for a simple if.

Sample number be 5

Step no	number	number<6	square ← number * number	output
1	5			
2		Y	25	
3				25

SIMPLE IF –DRY RUN

Manual execution of steps in an algorithm is called dry run. Sample dry run for a simple if.

Sample number be 5

Step no	number	number<6	square \leftarrow number * number	output
1	5			
2		Y	25	
3				25

Sample number be 15

Step no	number	number<6	square \leftarrow number * number	output
1	15			
2		N	-	
3				-



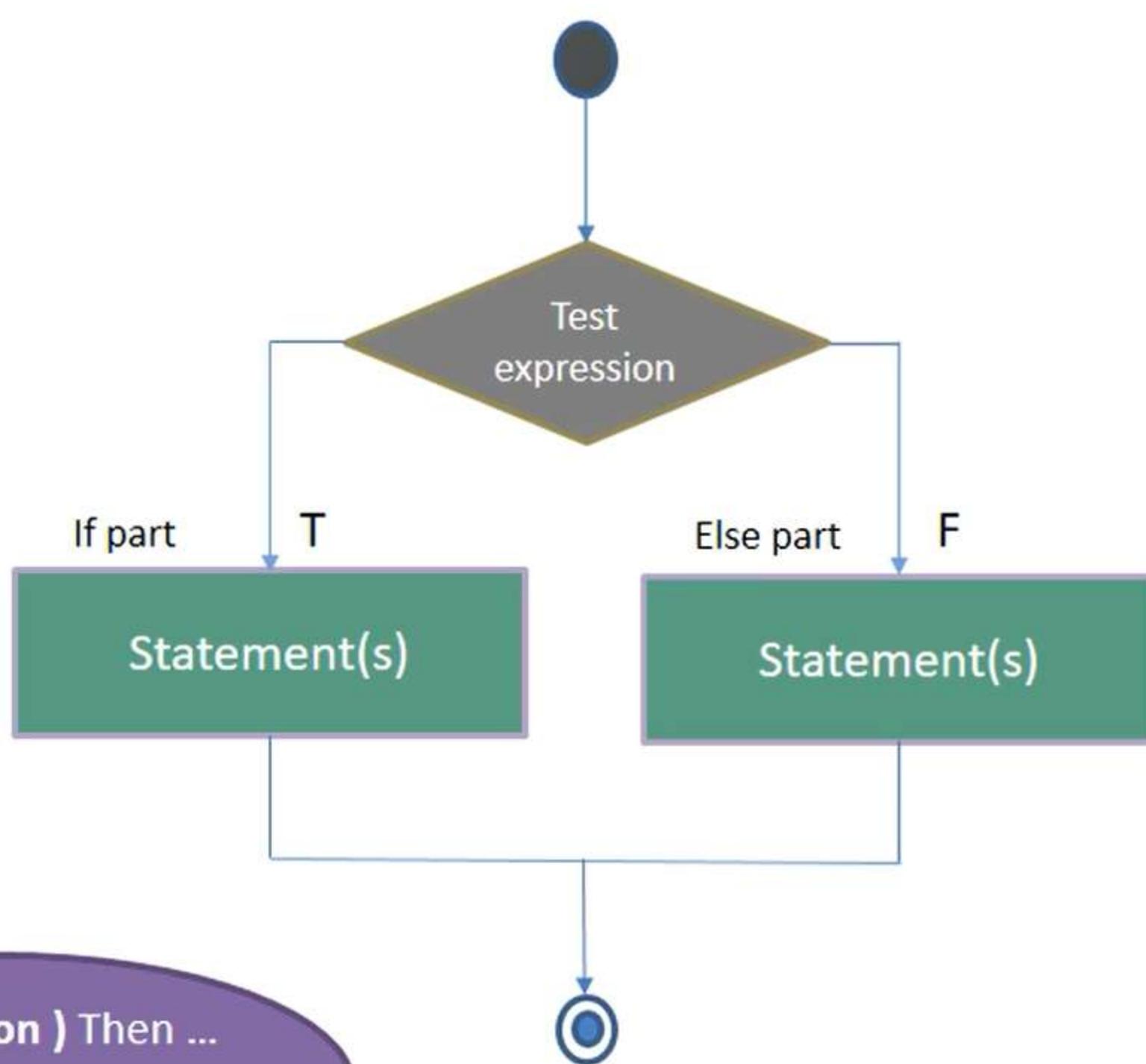
Condition becomes false and hence no output

SELECTION STATEMENT: IF-ELSE

Decides on the execution of one of the two statements, depending upon the test expression.

Two way branching statement works based on condition

- If the condition is true, the statements inside the if block are executed.
- If the condition is false, if block statements are skipped, and the statements inside else block gets executed



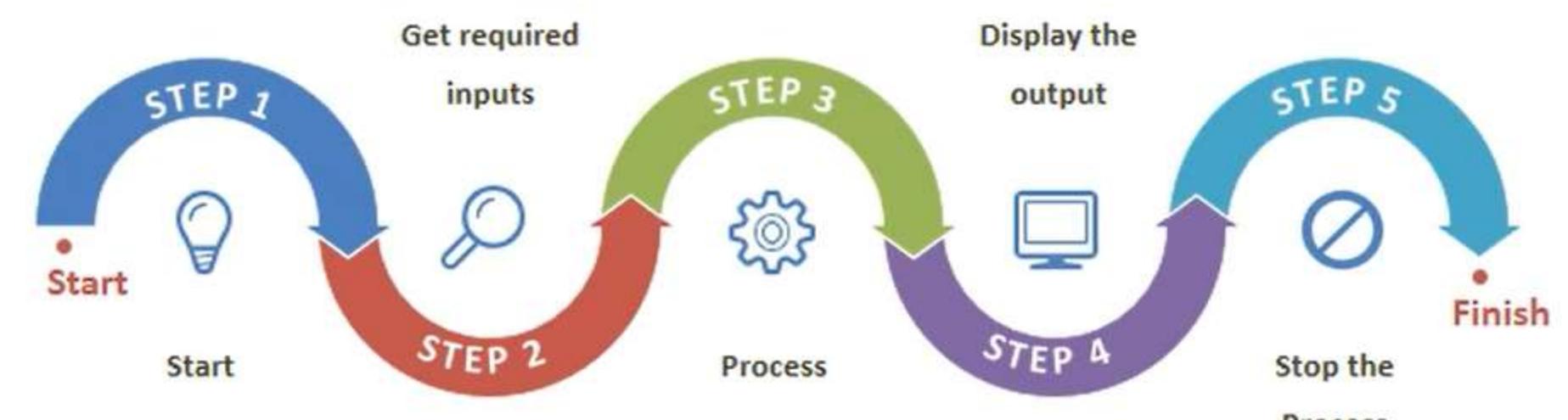
```
graph TD; Start(( )) --> Test{Test expression}; Test -- T --> If[Statement(s)]; Test -- F --> Else[Statement(s)]; If --> End(( )); Else --> End;
```

IF (Condition) Then ...
ELSE
ENDIF

ALGORITHM FOR IF-ELSE

Sample algorithm to check whether a number is even or odd

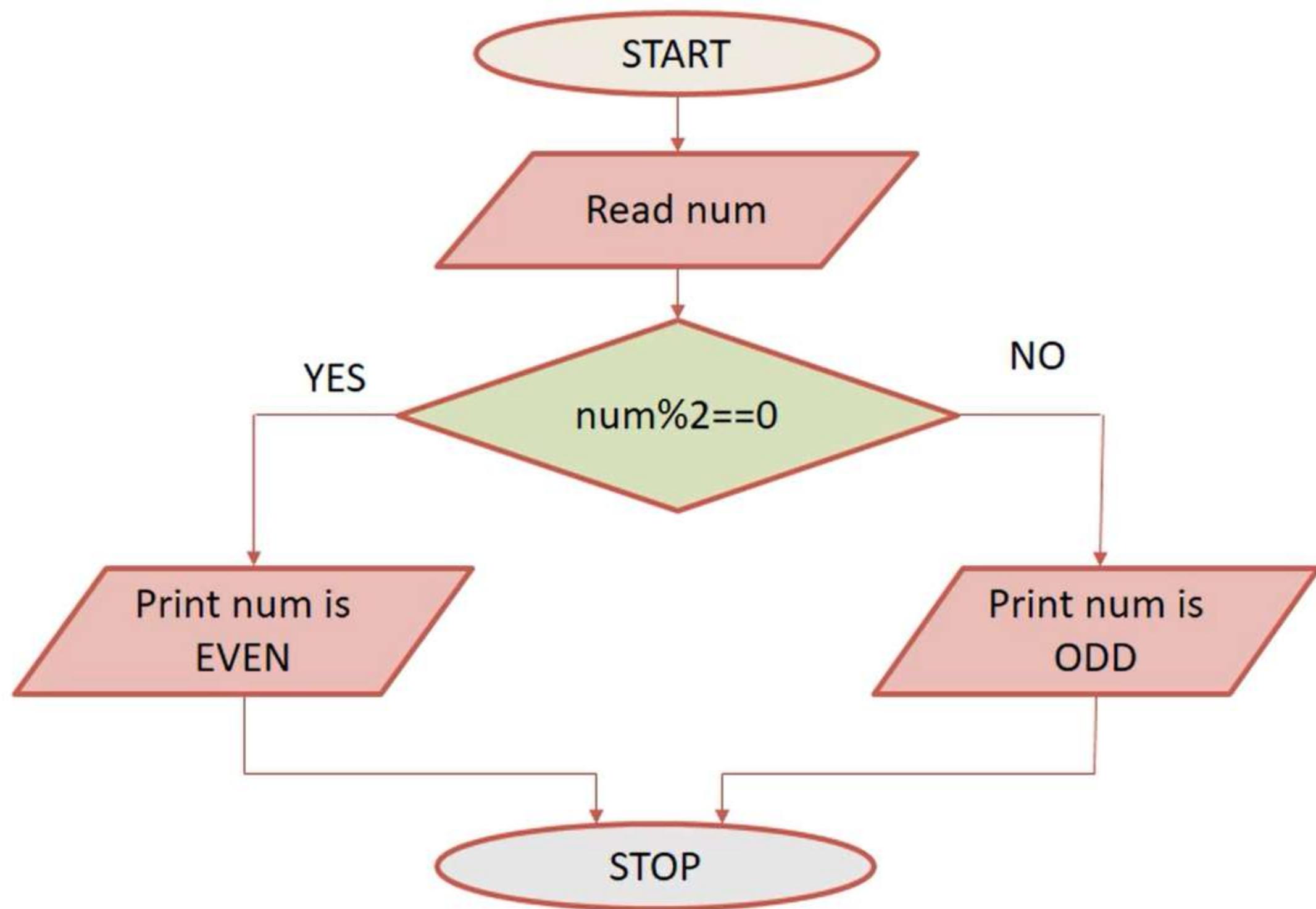
-  **STEP 1:**
Start the process
-  **STEP 2:**
Get number
-  **STEP 3:**
Check whether number mod 2 is equal to 0,if YES GOTO
step 4, if NO GOTO step 5
-  **STEP 4:**
Display EVEN. GOTO step 6.
-  **STEP 5:**
Display ODD
-  **STEP 6:**
Stop the process



Example algorithm for
If/else logic

FLOWCHART FOR IF-ELSE

Basic Flowchart representation to check if a number is even or odd



(Start the process)

(Let the variable be num and the value for num=5)

(5 is divided by 2 and the remainder is obtained.
This step checks whether the remainder value is
equal to 0 or not)

(Displays the result as ODD)

(Ends the process)

PSEUDOCODE FOR IF-ELSE

Pseudocode representation to check if a number is even or odd

BEGIN

DECLARE variables number , result

READ number

SET result ← number%2

IF result==0

THEN

PRINT number is EVEN

ELSE

PRINT number is ODD

ENDIF

END



IF-ELSE DRY RUN

Sample number be 5

Step no	Number	Result \leftarrow Number%2	Result	Result==0?	OUTPUT
1	5				
2		1	1		
3				N	
4					ODD



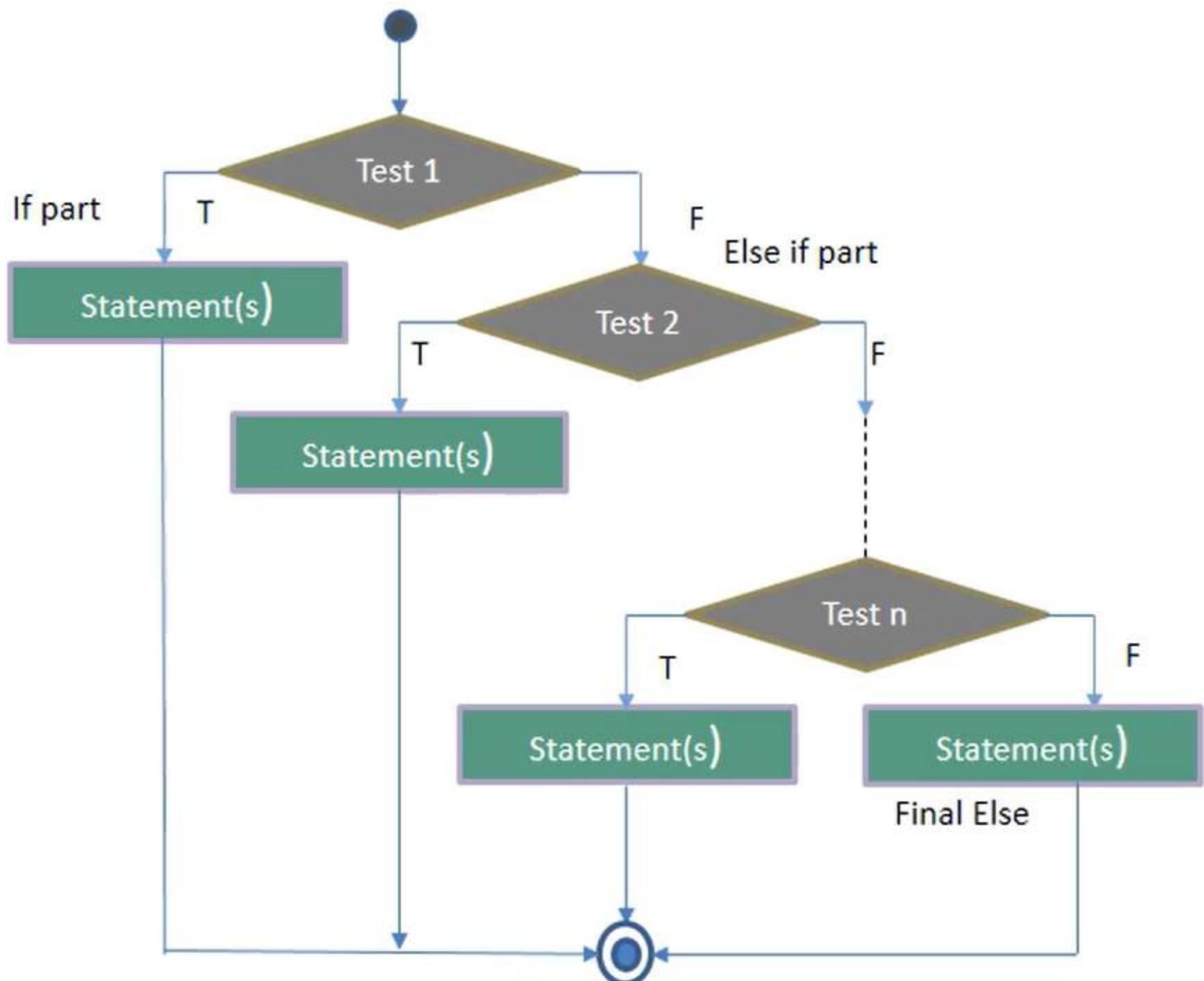
Condition becomes false and hence prints the number as ODD

SELECTION STATEMENT: ELSE-IF LADDER

Decides the execution of statements, when multiple conditions are to be checked

Used for branching when multiple test expressions are to be checked

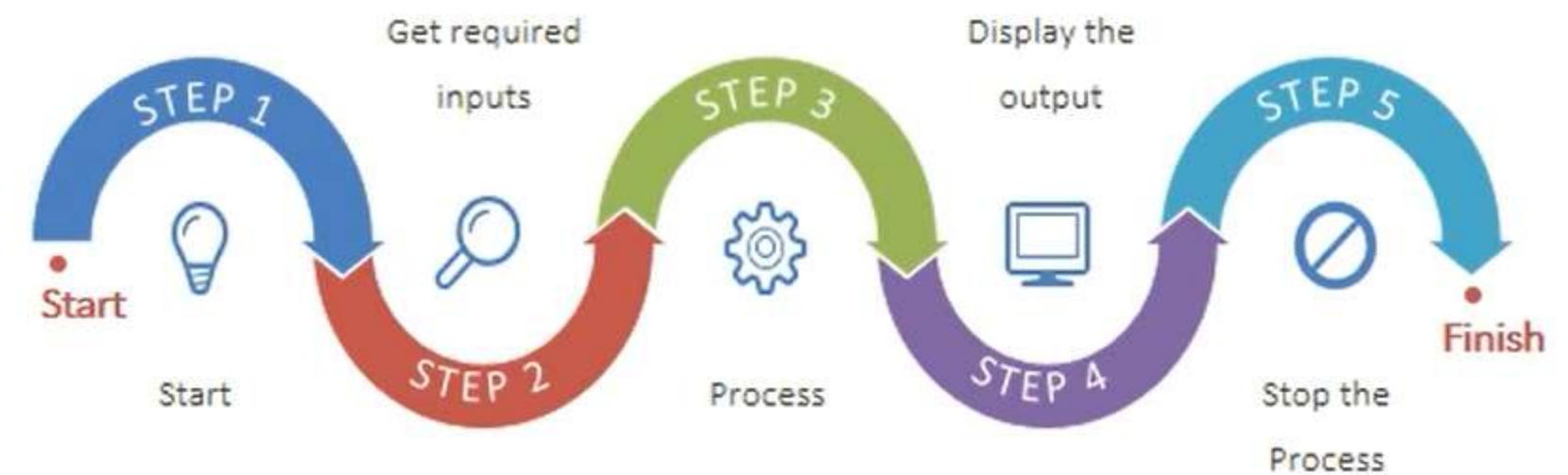
- Conditions are checked one by one. If any of the conditions is evaluated to be true, that block alone gets executed and the other block statements are ignored.
- If all the conditions are evaluated to false, the default else part gets executed



ALGORITHM FOR ELSE-IF LADDER

Sample algorithm to generate grade report

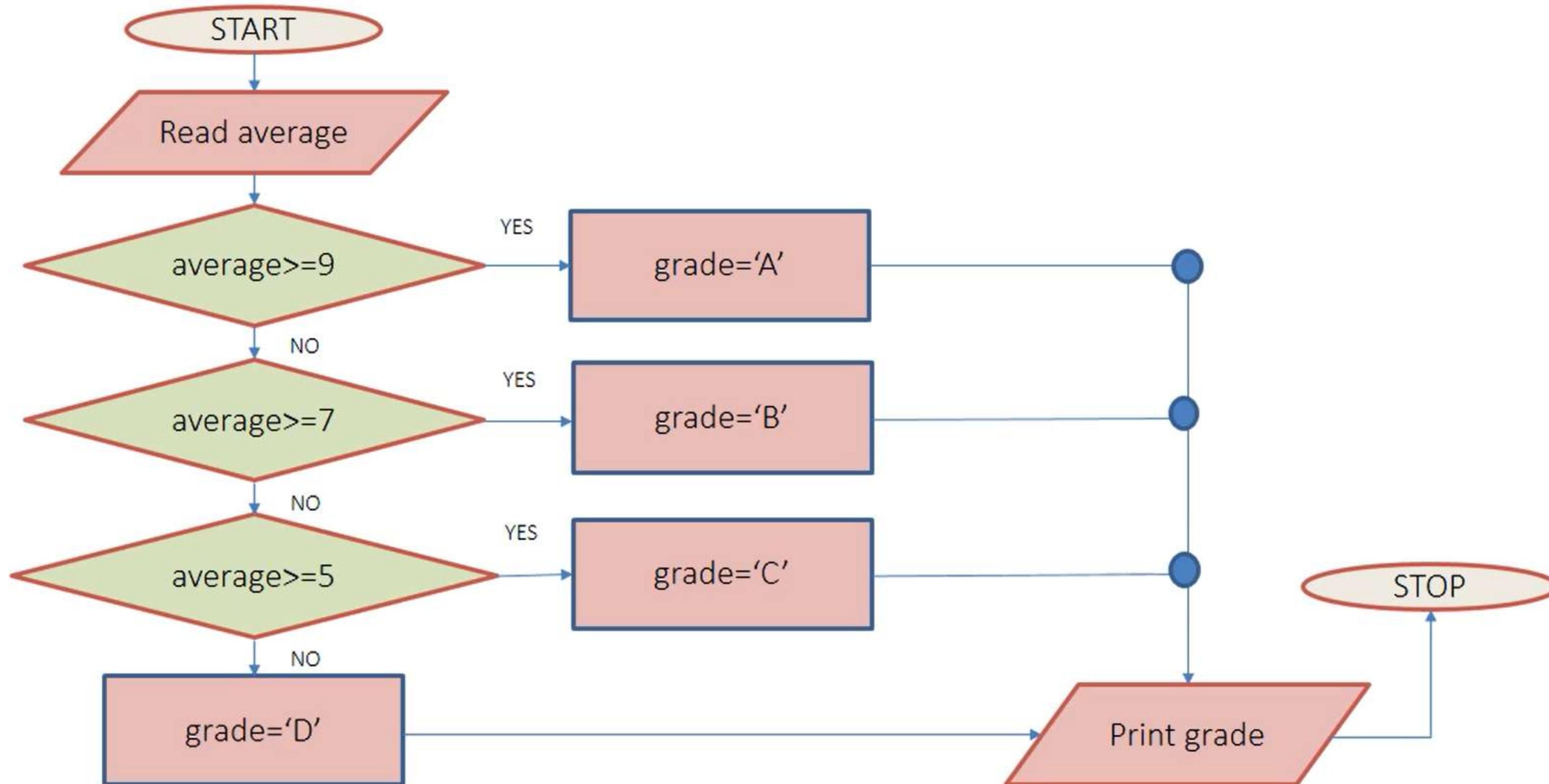
-  STEP 1:
Start the process
-  STEP 2:
Read average
-  STEP 3:
If average is greater than or equal to 9,grade is set as "A' and GOTO step 7
-  STEP 4:
If average is greater than or equal to 7,grade is set as "B' and GOTO step 7
-  STEP 5:
If average is greater than or equal to 5, grade is set as "C' and GOTO step 7
-  STEP 6:
grade is set as 'D'
-  STEP 7:
Print grade
-  STEP 8:
Stop the process



Example algorithm for
Else if ladder logic

FLOWCHART FOR ELSE-IF LADDER

Basic Flowchart to generate grade report



PSEUDOCODE FOR ELSE-IF LADDER

Pseudocode representation to generate grade report

BEGIN

DECLARE variables average , grade

READ average

IF average>=9

SET grade='A'

ELSEIF average>=7

SET grade='B'

ELSEIF average>=5

SET grade='C'

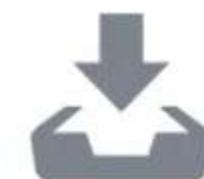
ELSE

SET grade='D'

PRINT grade

ENDIF

END



SELECTION –DRY RUN

Sample average value be 6

Step no	average	average>=9?	average>=7?	average>=5?	grade	output
1	6					
2		N				
3			N			
4				Y	C	
5						C



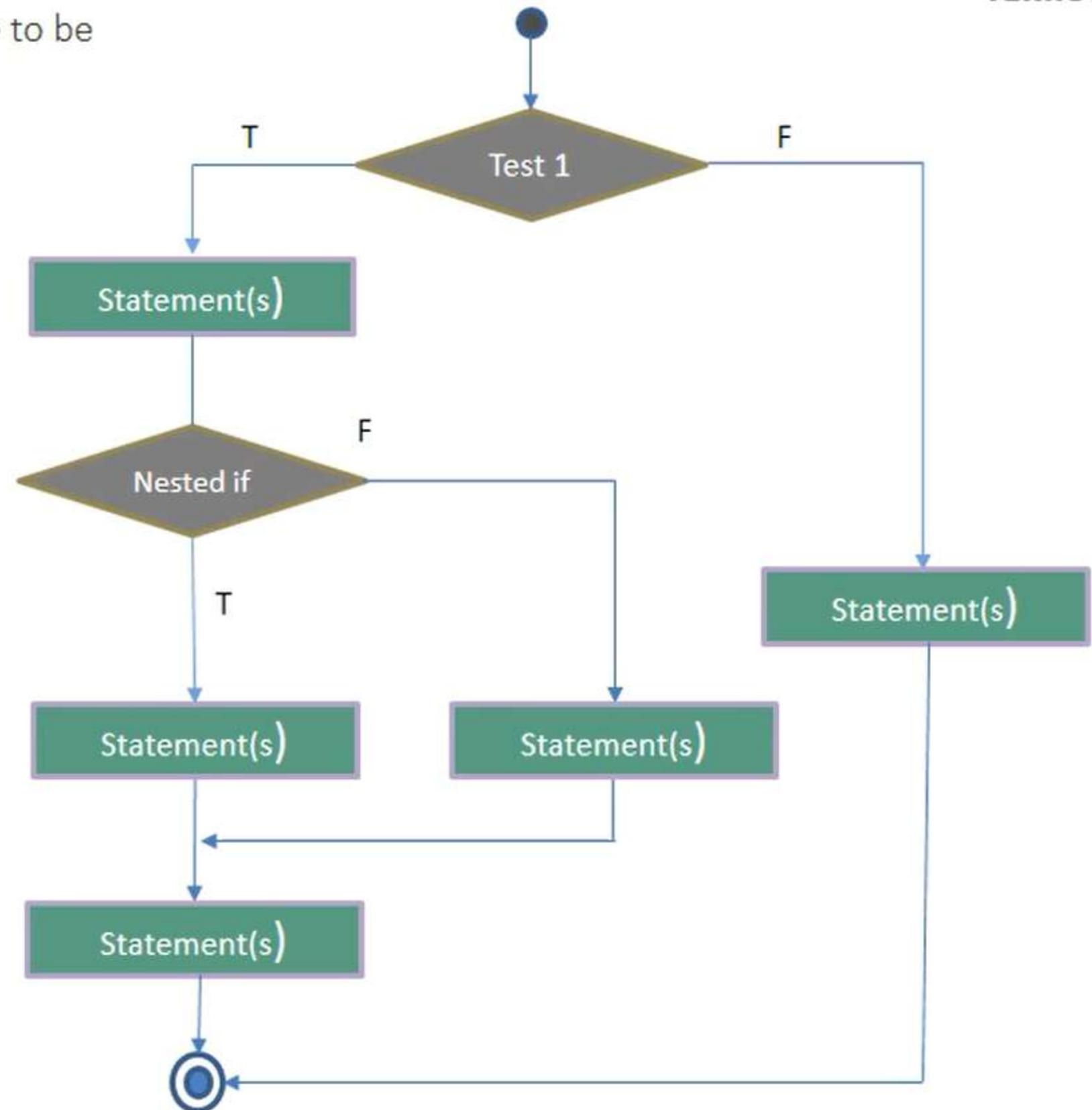
In step 4, condition becomes true and hence grade is set to C and prints the grade as C.

SELECTION STATEMENT: NESTED-IF

Nested if statements are used if there is one or more true conditions are to be checked

Used, if there is a sub condition to be tested inside another condition

- If else statement present inside the body of another “if” or “else”.
- Decisions can be nested to any level



WORKING OF NESTED-IF

Nesting of if-else statements

- Outer condition is checked first.
- If the outer expression is evaluated to be true, the program flow gets inside the block and tests for the inner condition.
 - If the inner condition is true, it executes the block of statements associated with it.
 - If the inner condition is false, it moves to the else part of the inner condition

IF <test-expression 1>

statement(s)

IF <test-expression2>

.....

statement(s)

.....

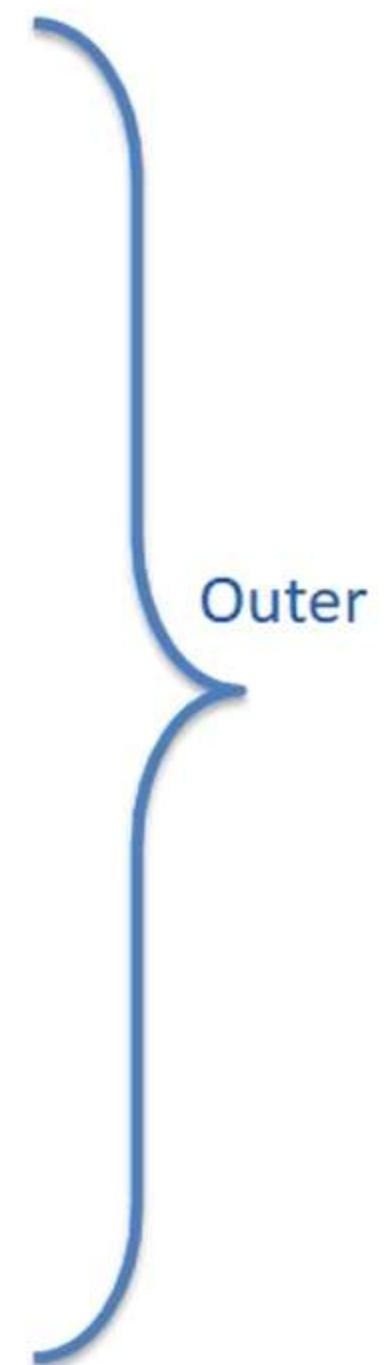
statement(s)

.....

ELSE

ELSE

statement(s)



WORKING OF NESTED-IF

Nesting of if-else statements

- Outer condition is checked first.
- If the outer expression is evaluated to be true, the program flow gets inside the block and tests for the inner condition.
 - If the inner condition is true, it executes the block of statements associated with it.
 - If the inner condition is false, it moves to the else part of the inner condition
- If the outer condition is false, it skips the outer condition part and moves to the else part of the outer condition.

IF <test-expression 1>

statement(s)

IF <test-expression2>

.....

statement(s)

.....

ELSE

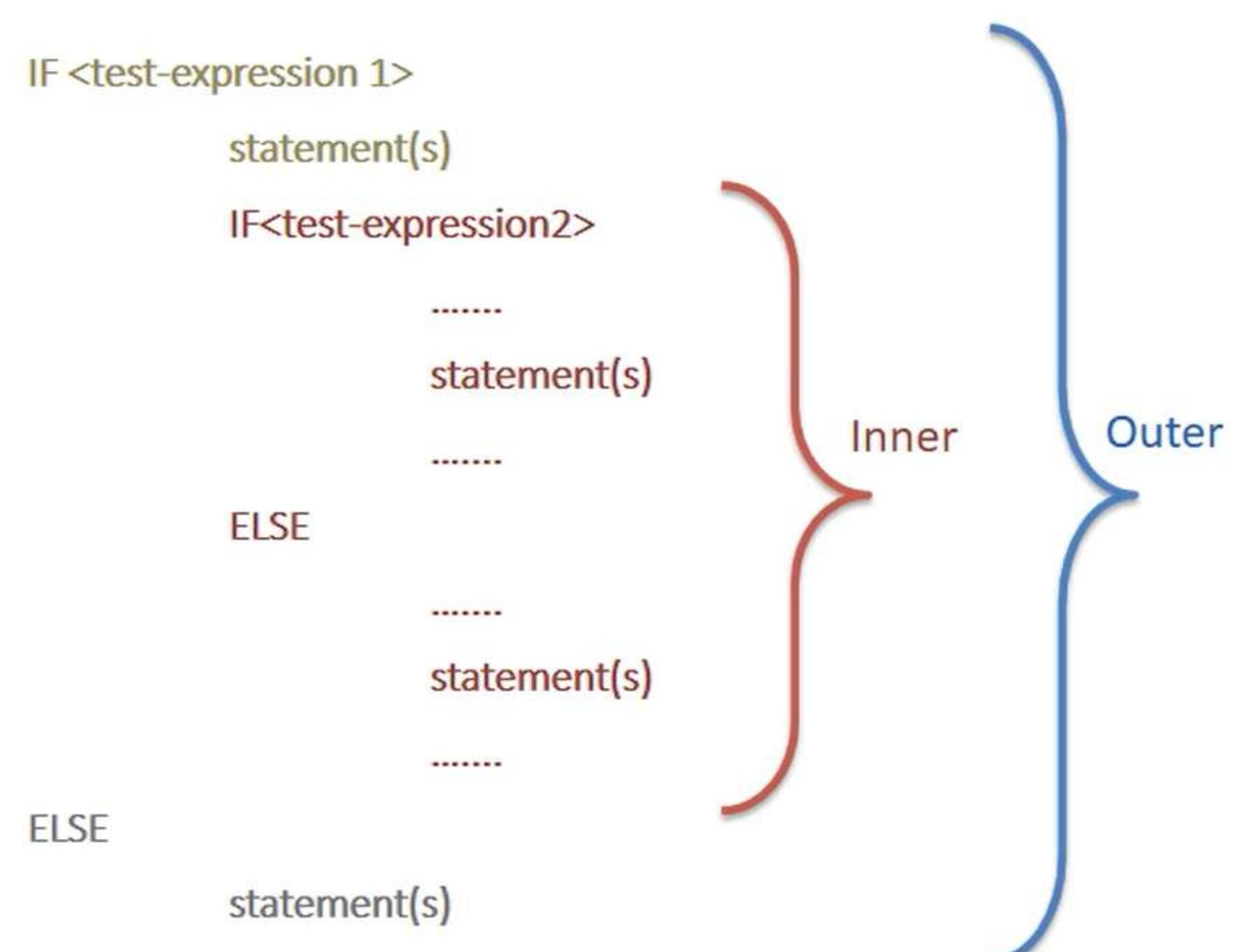
.....

statement(s)

.....

ELSE

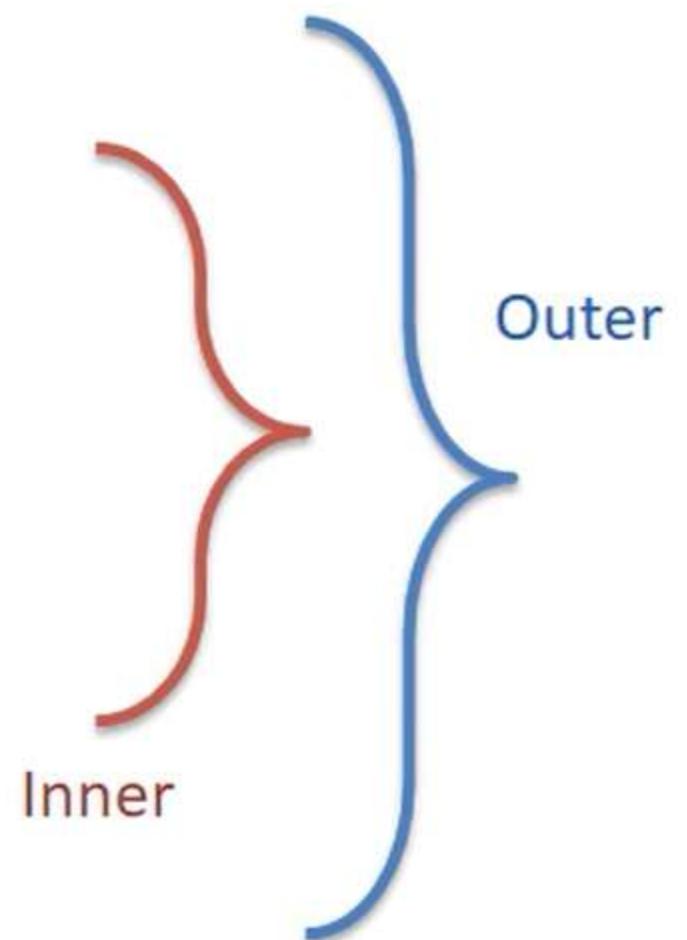
statement(s)



NESTED-IF -EXAMPLE

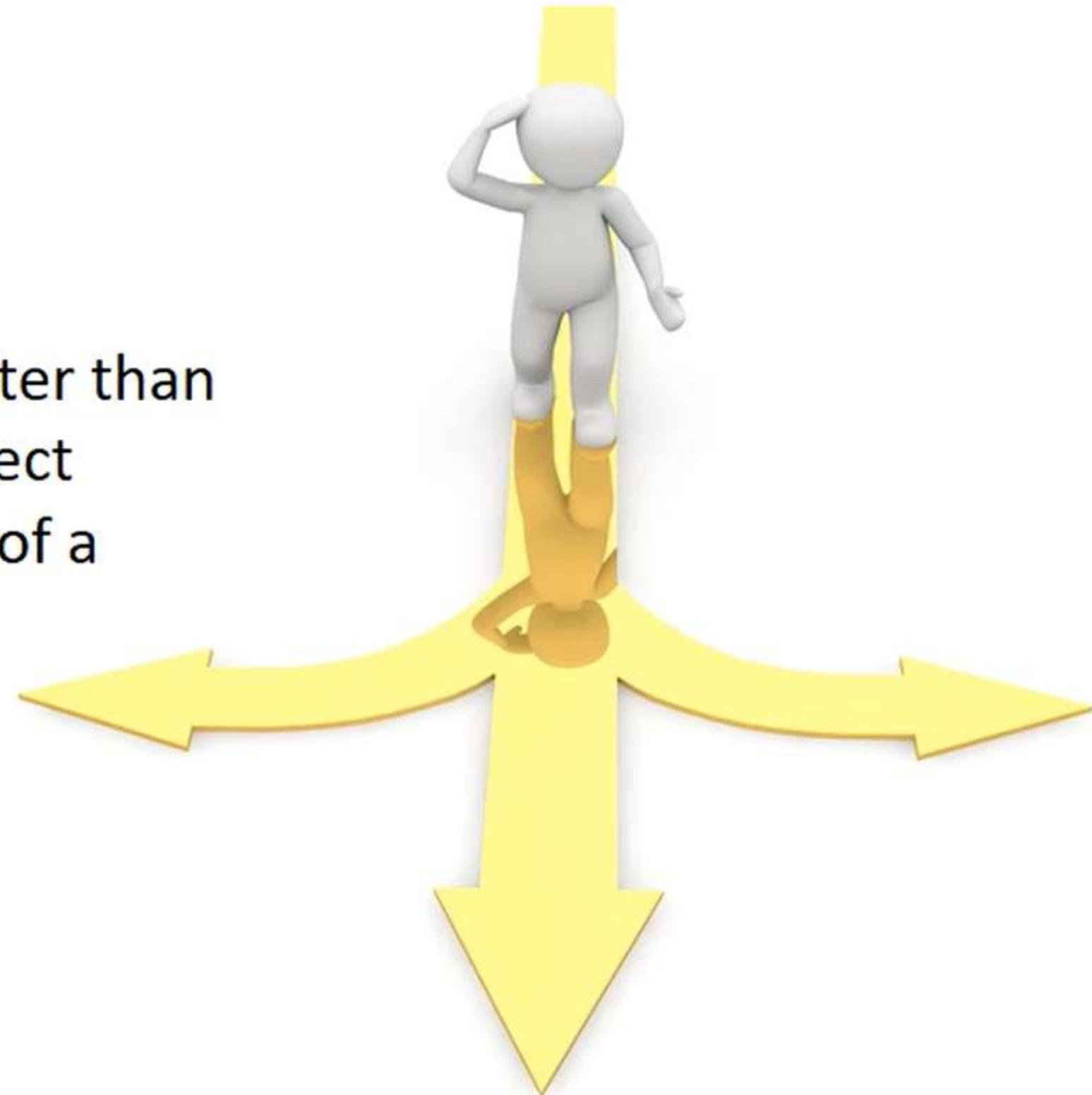
Sample Pseudo code to find the Profit and Loss

```
BEGIN
DECLARE variables cost_price, selling_price
INPUT cost_price, selling_price
IF (cost_price>0) AND (selling_price>0) THEN
    IF cost_price > selling_price THEN
        PRINT "You have Loss"
    ELSE IF cost_price < selling_price THEN
        PRINT "You have Profit"
    ELSE
        PRINT "No profit, No loss"
    END IF
ELSE
    PRINT "Invalid Price"
END IF
END
```



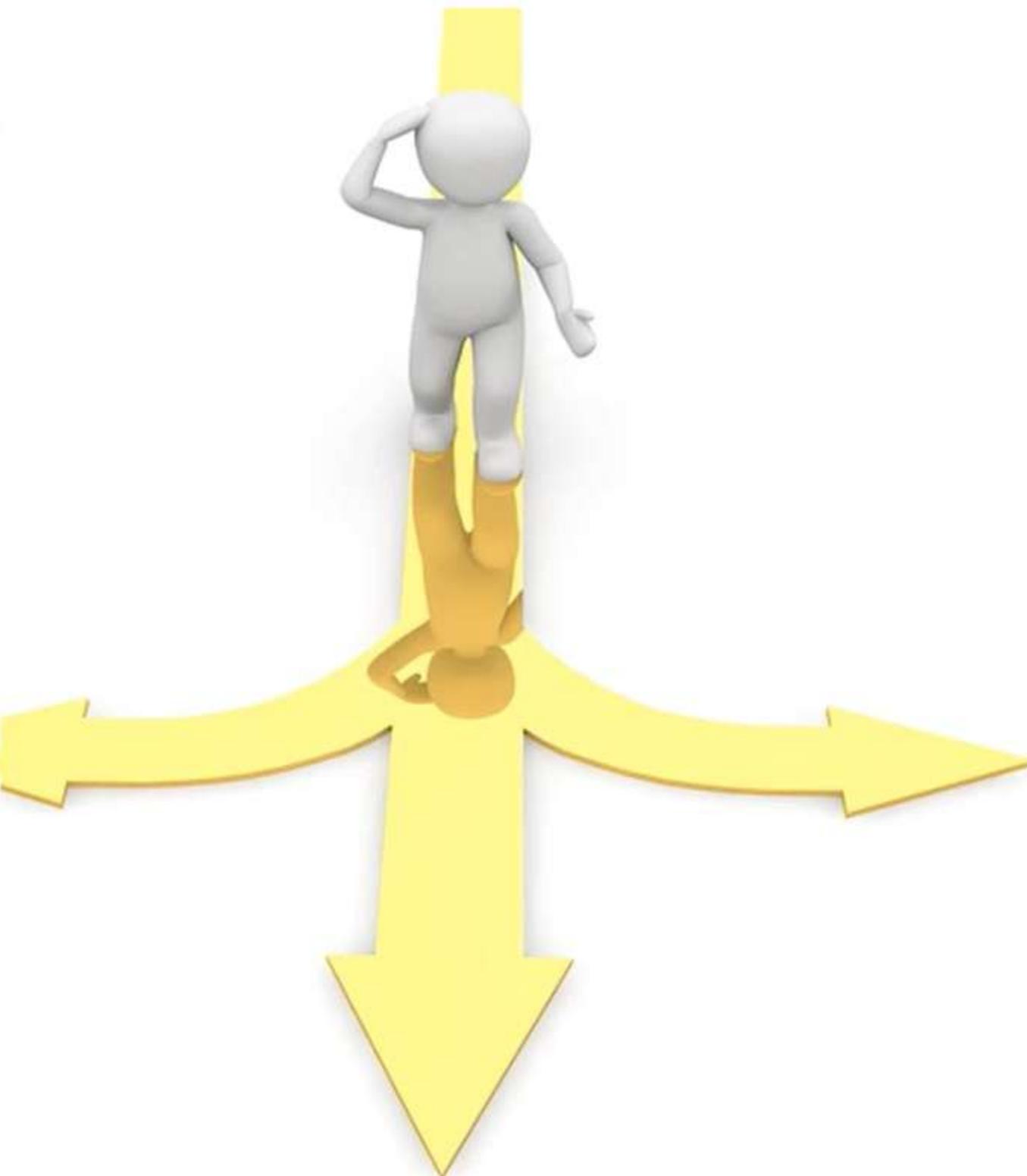
Time To think

Let's say, the eligible age to vote is greater than or equal to 18. What would be the correct pseudo code to check for the eligibility of a person to vote??



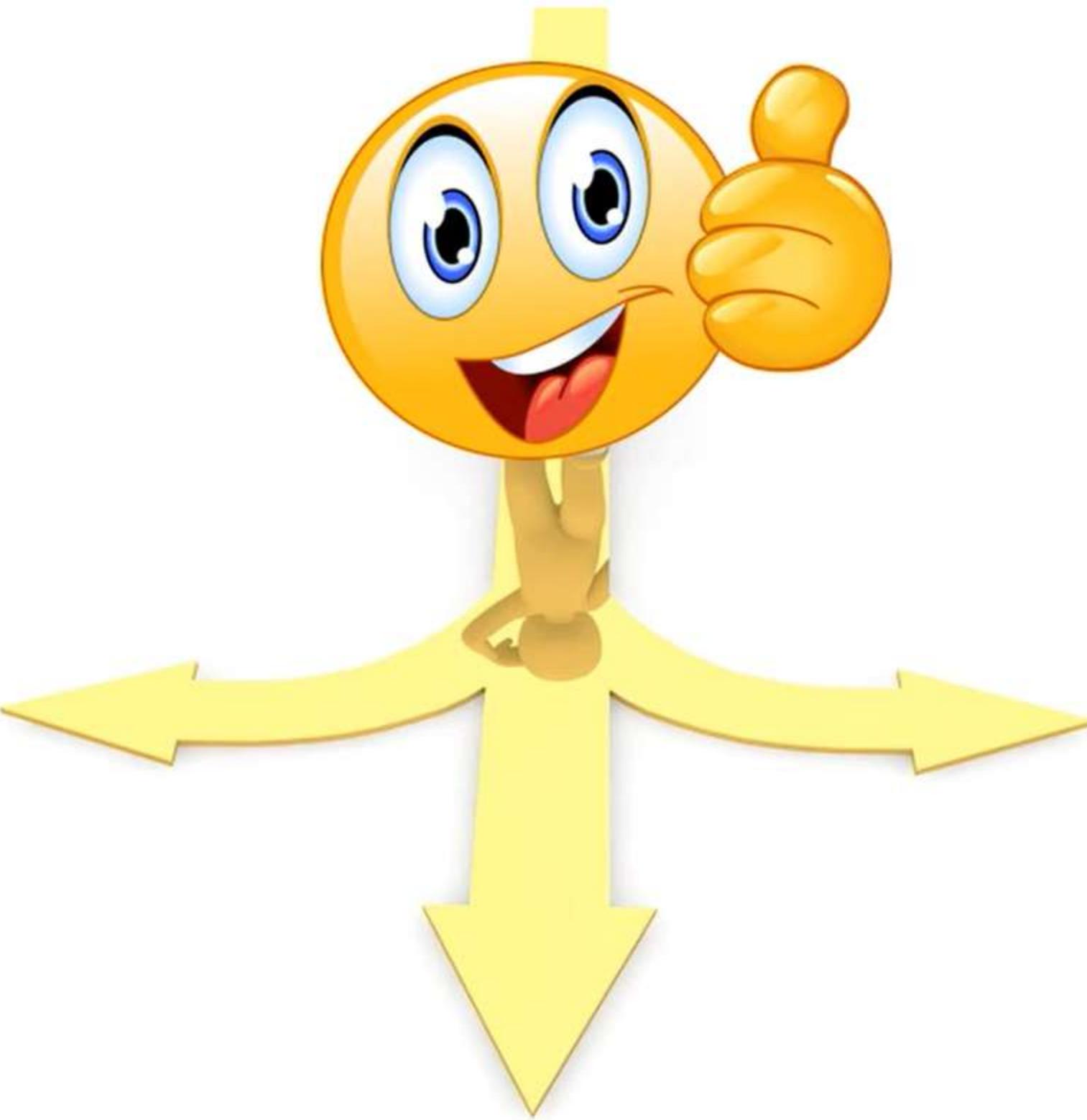
Time To think

```
Input the age
Declare variable age, registration_status
IF age>= 18THEN
PRINT "You may vote"
ELSE
PRINT "You must register to vote"
IF registration_status=='YES' THEN
ENDIF
PRINT "You are not Eligible to vote"
ELSE
ENDIF
BEGIN
END
```



Time To think

```
BEGIN  
    Declare variable age, registration_status  
    Input the age  
    IF age>= 18THEN  
        IF registration_status=='YES' THEN  
            PRINT "You may vote"  
        ELSE  
            PRINT "You must register to vote"  
        ENDIF  
    ELSE  
        PRINT "You are not Eligible to vote"  
    ENDIF  
END
```



Summary

- The flow of program
- Introduction to selection statements
- Categories of selection statements
- How to construct algorithm, flowchart and pseudo codes
for selection statements



LOGIC DEVELOPMENT



In this module you will learn

- Looping statements
- Categories of Looping statements
- How to construct algorithm, flowchart and pseudo codes
for Looping statements



LOOPING OR ITERATION



It may be necessary to repeat a certain process until a certain condition is satisfied



LOOPING OR ITERATION

Looping
statements can be
categorized as

- For:
- While
- Do-While (**Do- Until**)
- Nested-For

FOR-LOOP STATEMENT

Repeats a statement or sequence of statements multiple times.

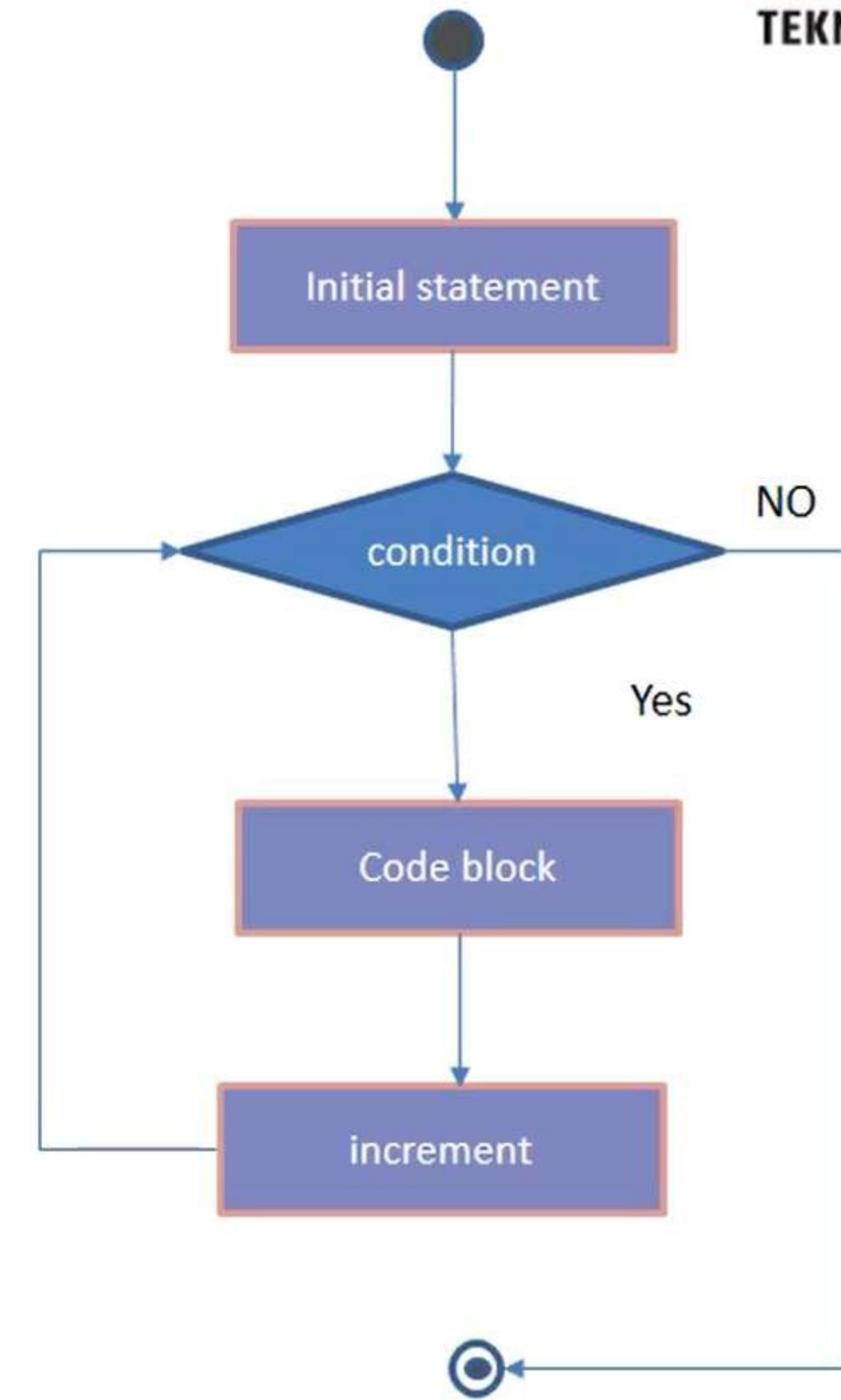
Iterates the sequence of statements until the condition becomes false.

 Code may be executed 0 number of times or as many times as per the required condition.

 For loop is Entry Controlled - condition is checked before entering the block of statements .

Used to execute block of statements, when the number of iterations are known

**FOR (specific condition)
DO
....
END FOR**



ALGORITHM FOR - FOR LOOP LOGIC

Sample algorithm for printing the sum of marks of 4 subjects

STEP 1:

Start the process

STEP 2:

Let $i=0, sum=0$

STEP 3:

Check whether $i < 4$, if YES GOTO step 4, if NO GOTO step 7

STEP 4:

Read number

STEP 5:

Add sum and the number, store the result in sum

STEP 6:

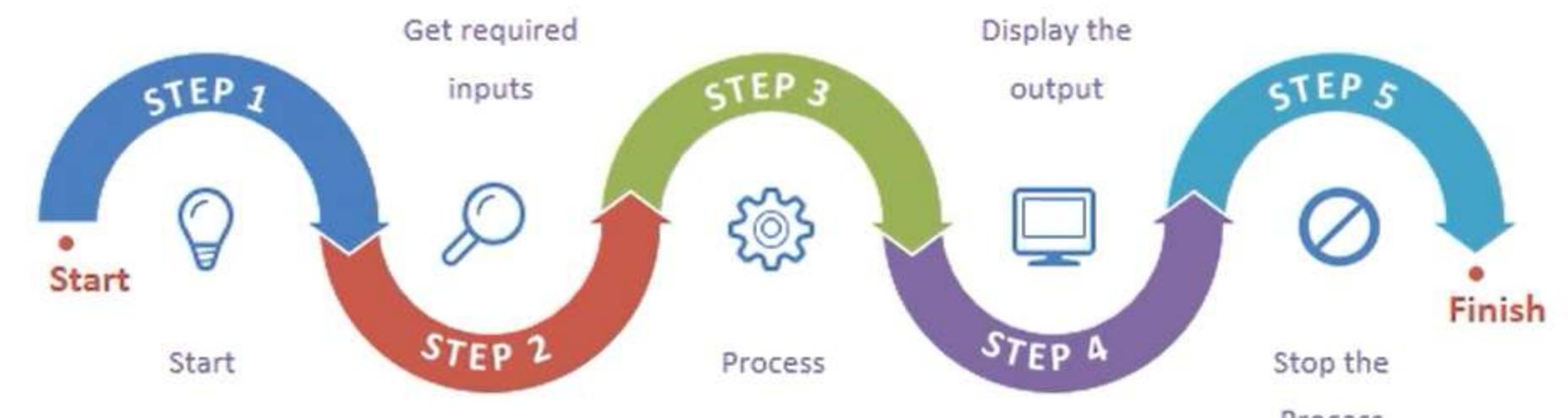
Increment the i value by 1, GOTO step 3

STEP 7:

Print sum

STEP 8:

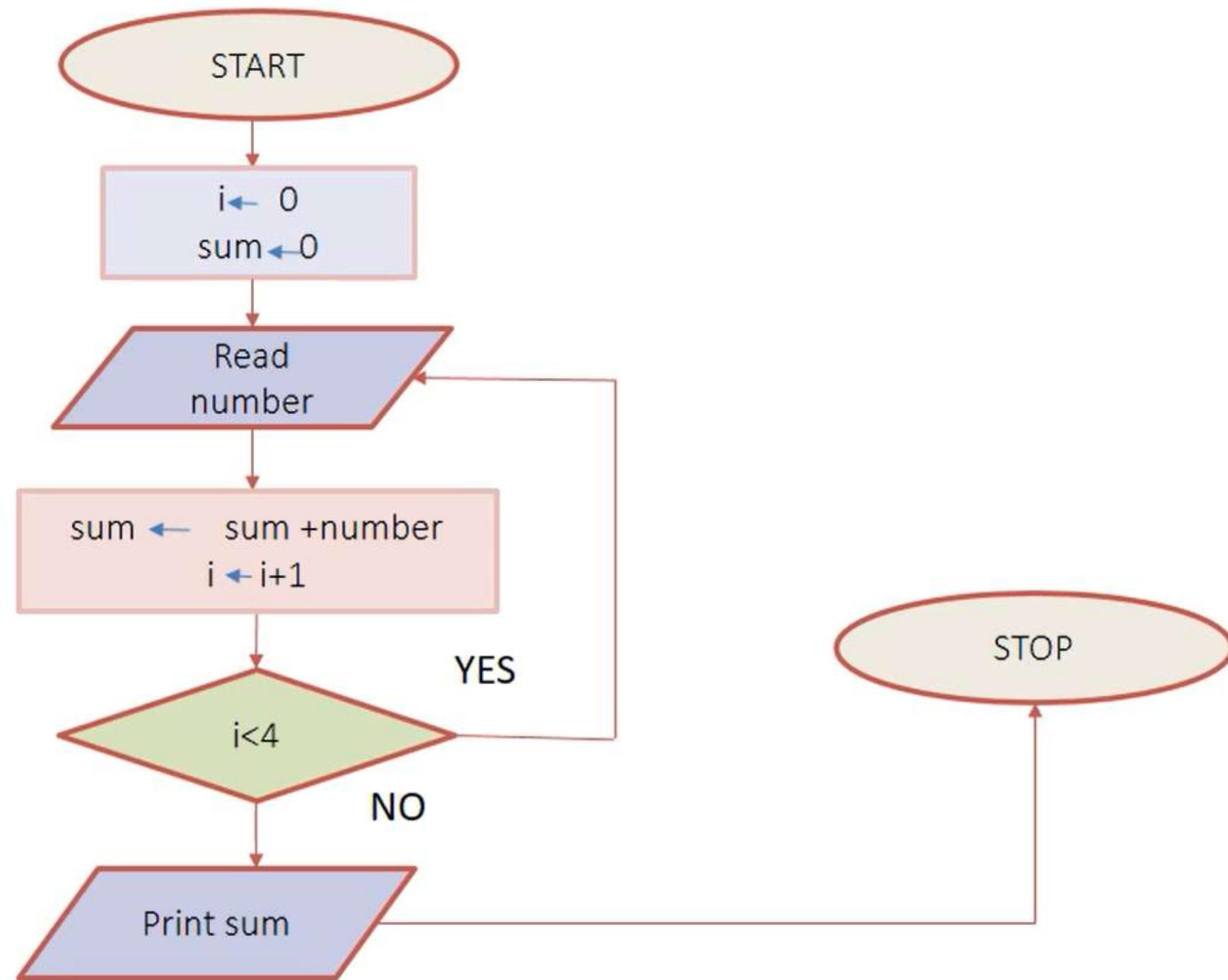
Stop the process



In this problem scenario the exact number of iterations for execution is known. Loop has to continue for the given number of times i.e., 4 times until it meets the condition . Hence FOR loop is used.

FLOWCHART FOR -FOR LOOP LOGIC

Basic flowchart for printing the sum of marks of 4 subjects



PSEUDOCODE FOR -FOR LOOP LOGIC

Pseudocode representation for printing the sum of marks of 4 subjects

BEGIN

 DECLARE variables i , sum, number

 SET i=0,sum=0

 FOR i=0 to 3 do

 READ number

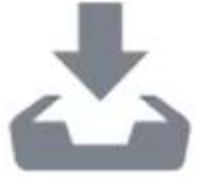
 sum ← sum +number

 i=i+1

 END FOR

 PRINT sum

END



FOR LOOP –DRY RUN

Step no	i	Sum	Condition satisfied(Y/N) $i < 4$	Read number	$sum = sum + number$	$i=i+1$	Output
1	0	0					
2	0	0	Y				
3				70			
4					$0+70=70$		
5						$0+1=1$	
2	1	70	Y				
3				60			
4					$70+60=130$		
5						$1+1=2$	



2,3,4,5 steps are the block of statements which repeats 4 times for getting the marks of the four subjects

FOR LOOP–DRY RUN

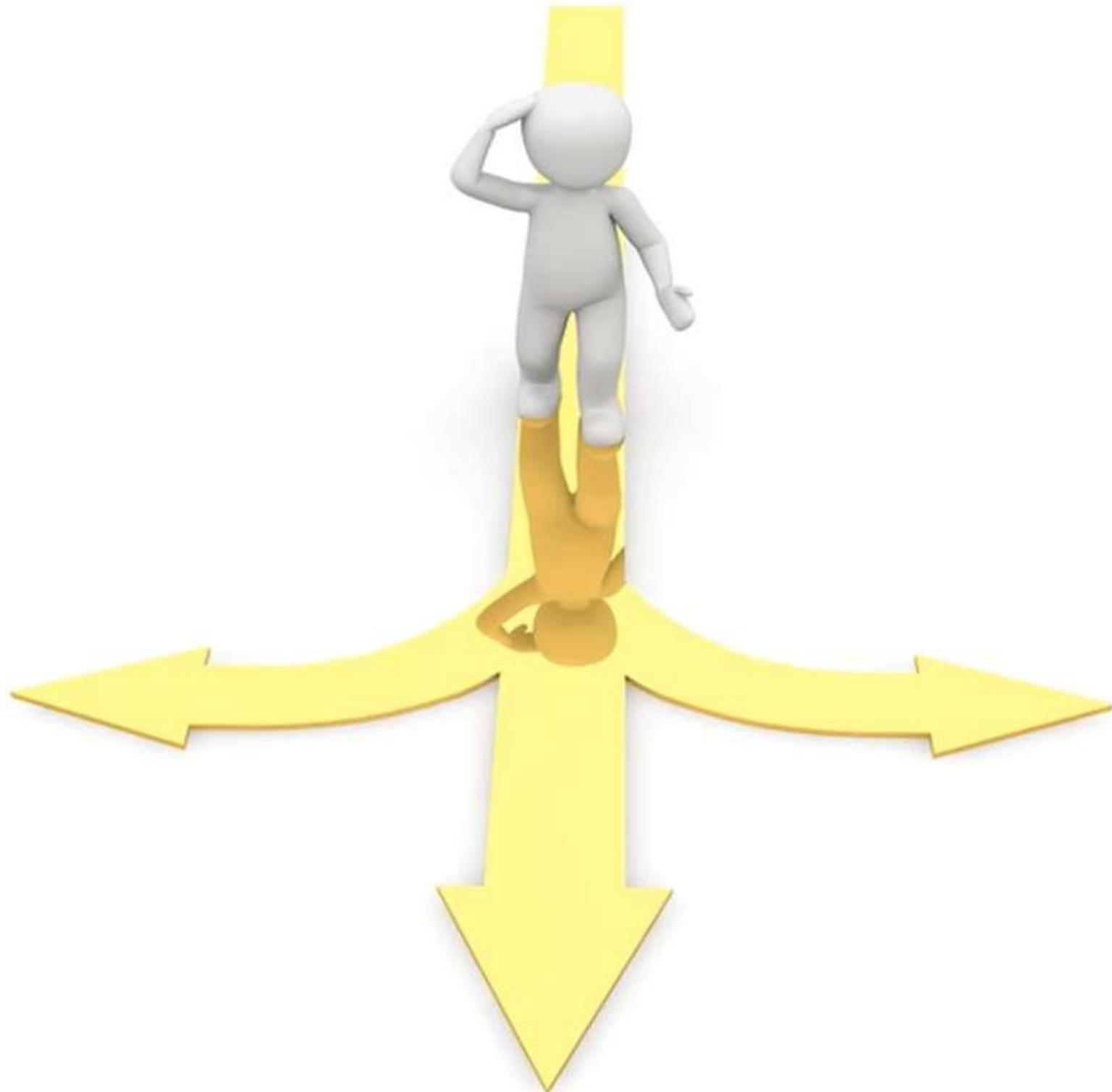
Step no	i	Sum	Condition satisfied(Y/N) $i < 4$	Read number	$sum =$ $sum + number$	$i=i+1$	Output
2	2	130	Y				
3				80			
4					$130+80=210$		
5						$2+1=3$	
2	3	210	Y				
3				75			
4					$210+75=285$		
5						$3+1=4$	
2	4	285	N				
7							Prints 285



When condition fails, the loop gets terminated and it moves to step 7, where it prints the sum value as 285.

Time To think

Let say, user wants to print
the numbers from 1 to 10.
What is the correct
pseudo code to print the
numbers from 1 to 10?



TIME TO THINK

Pseudocode to Print the numbers from 1 to 10

DECLARE variable number

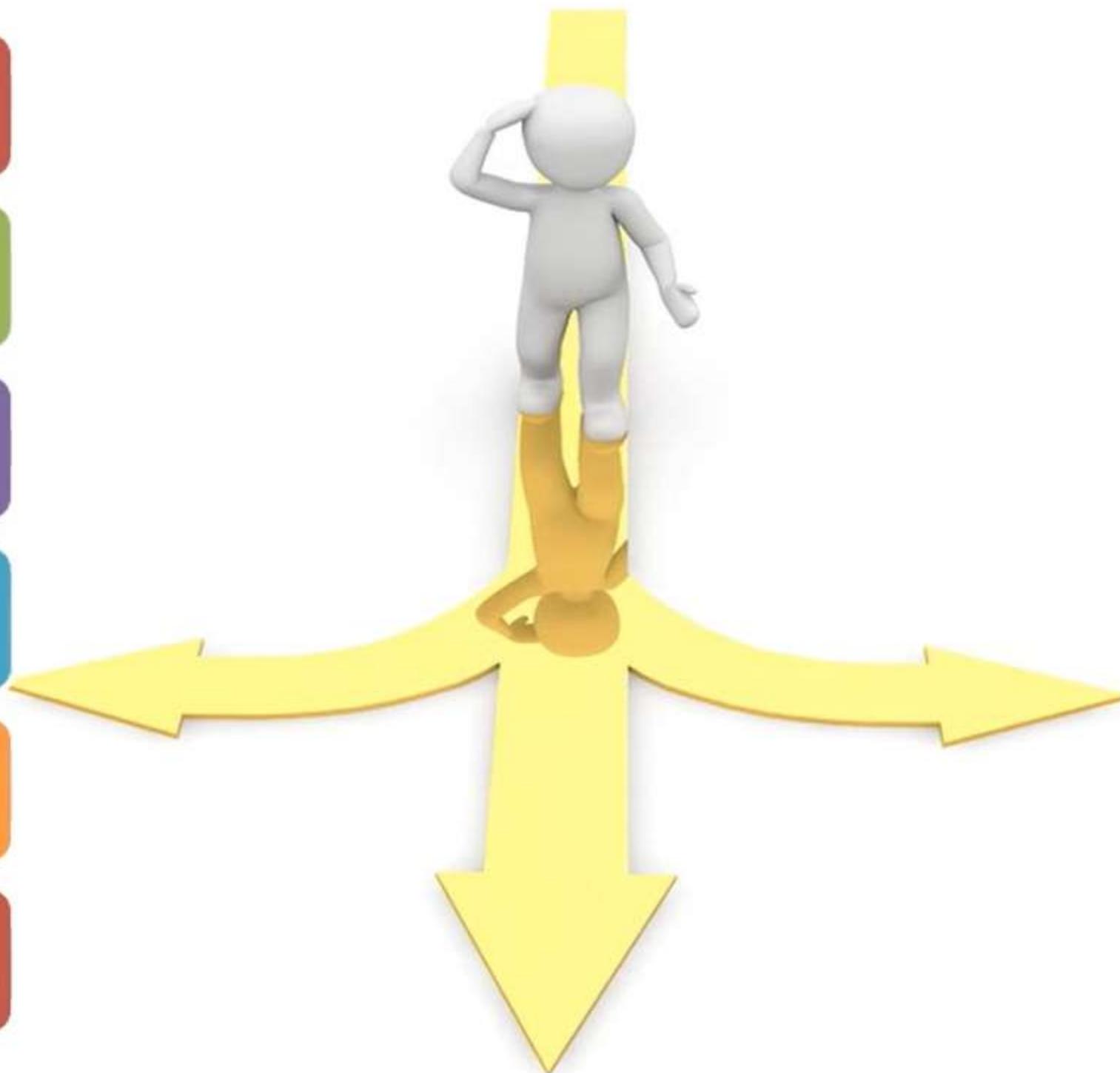
BEGIN

PRINT number

END

FOR number=0 to 9 **THEN**

END FOR

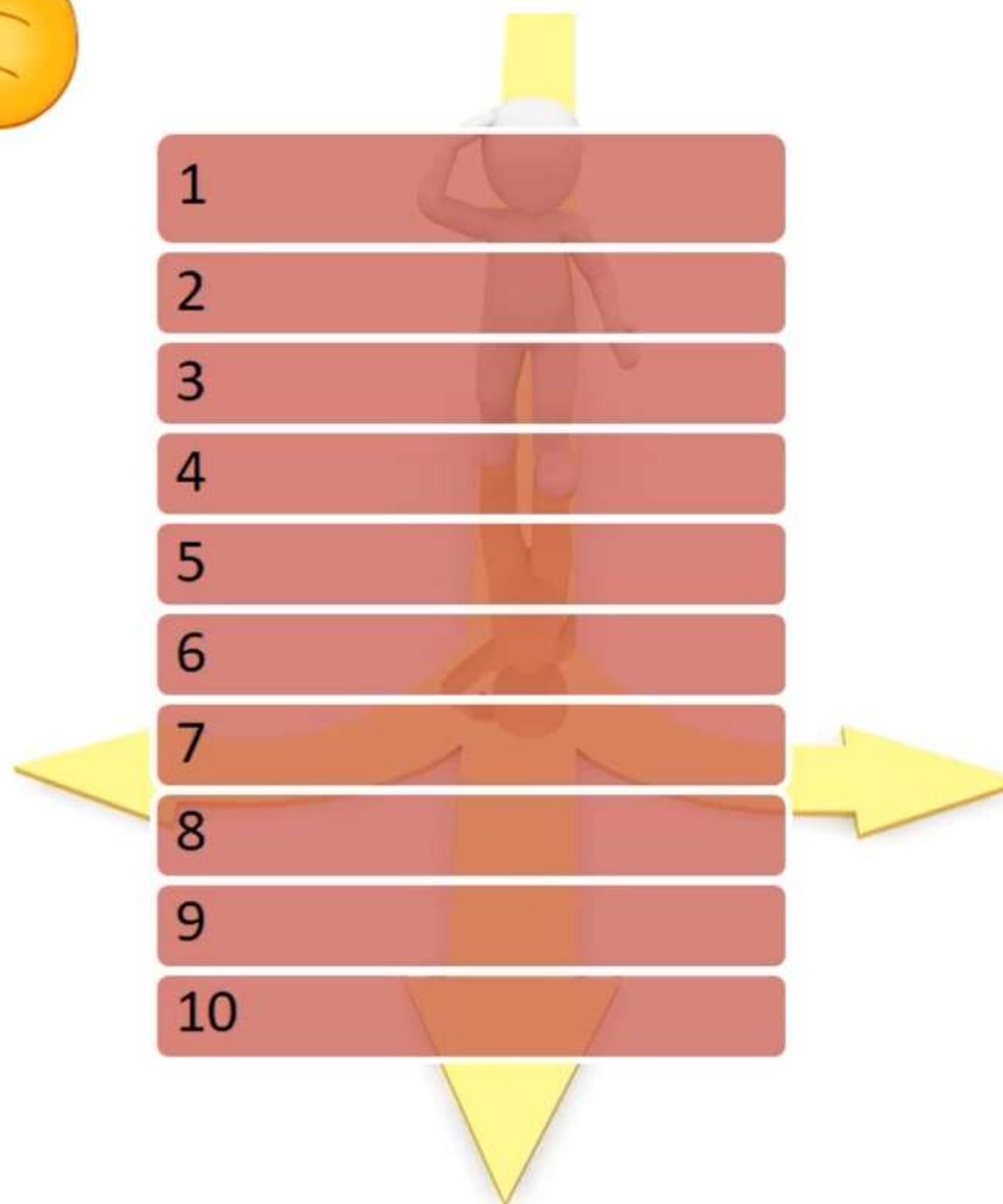


TIME TO THINK

Pseudocode to print the numbers from 1 to 10



```
BEGIN  
    Declare variable number  
    FOR number=0 to 9 THEN  
        PRINT number+1  
    END FOR  
END
```



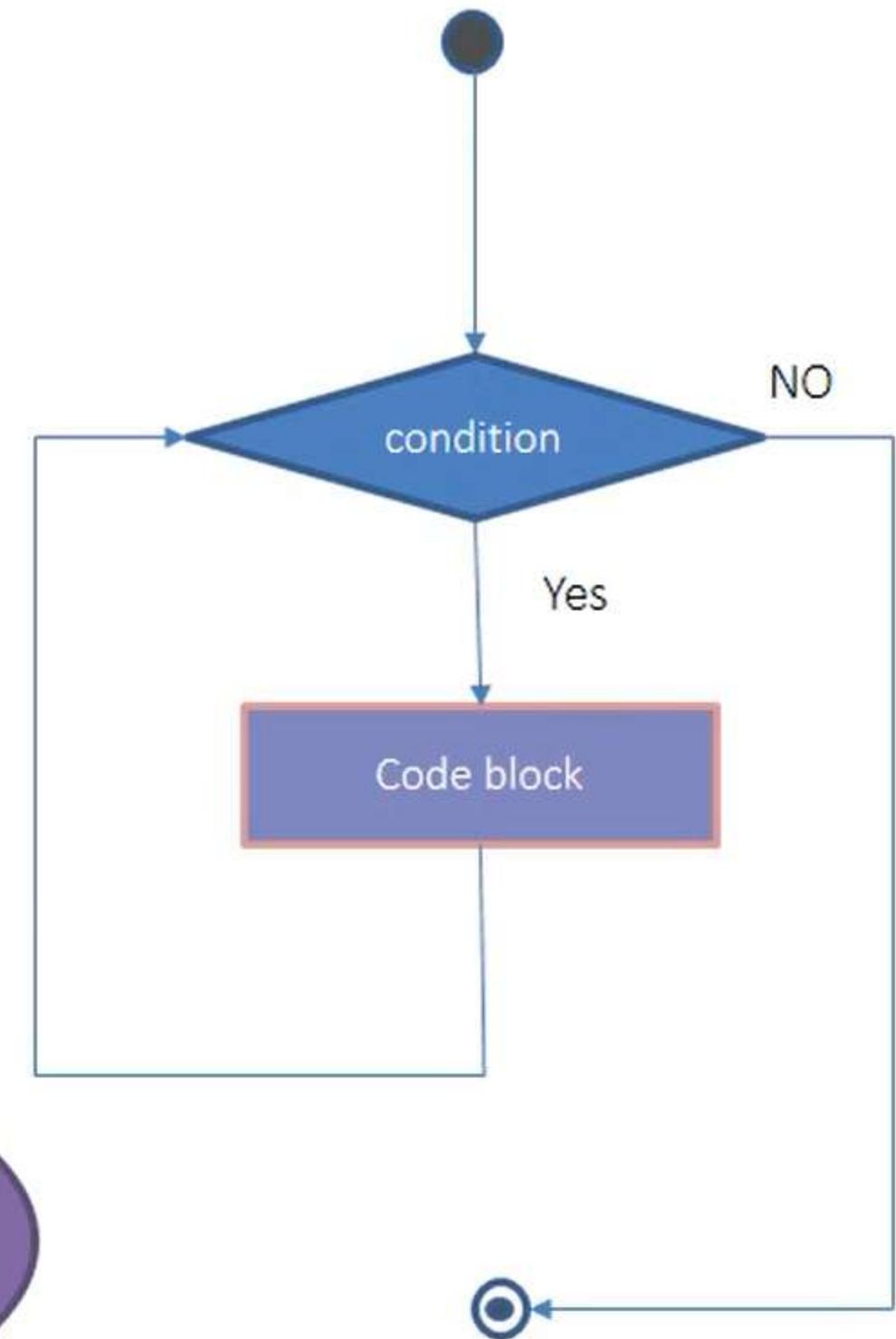
WHILE-LOOP STATEMENT

Repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.

- Code may be executed 0 number of times or as many times as per the required condition.
- While loop is also entry controlled as the conditions are checked before entering the block of statements.

Used to execute block of statements, when the number of iterations are not known

**WHILE (condition)
DO
.....
END WHILE**



ALGORITHM FOR -WHILE LOOP

Sample algorithm for accepting the numbers, printing it only if the number is greater than 0. Loop continues until a negative number is given.

STEP 1:

Start the process

STEP 2:

Get the number

STEP 3:

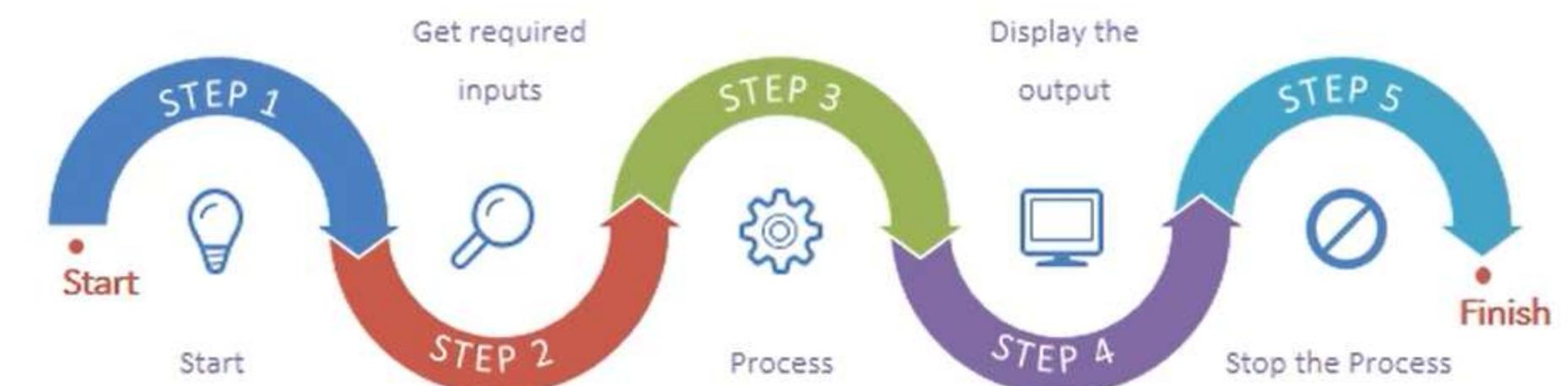
Check whether number>0,if YES GOTO step 4, if NO
GOTO step 5

STEP 4:

Display the number. GOTO Step 2

STEP 5:

Stop the process



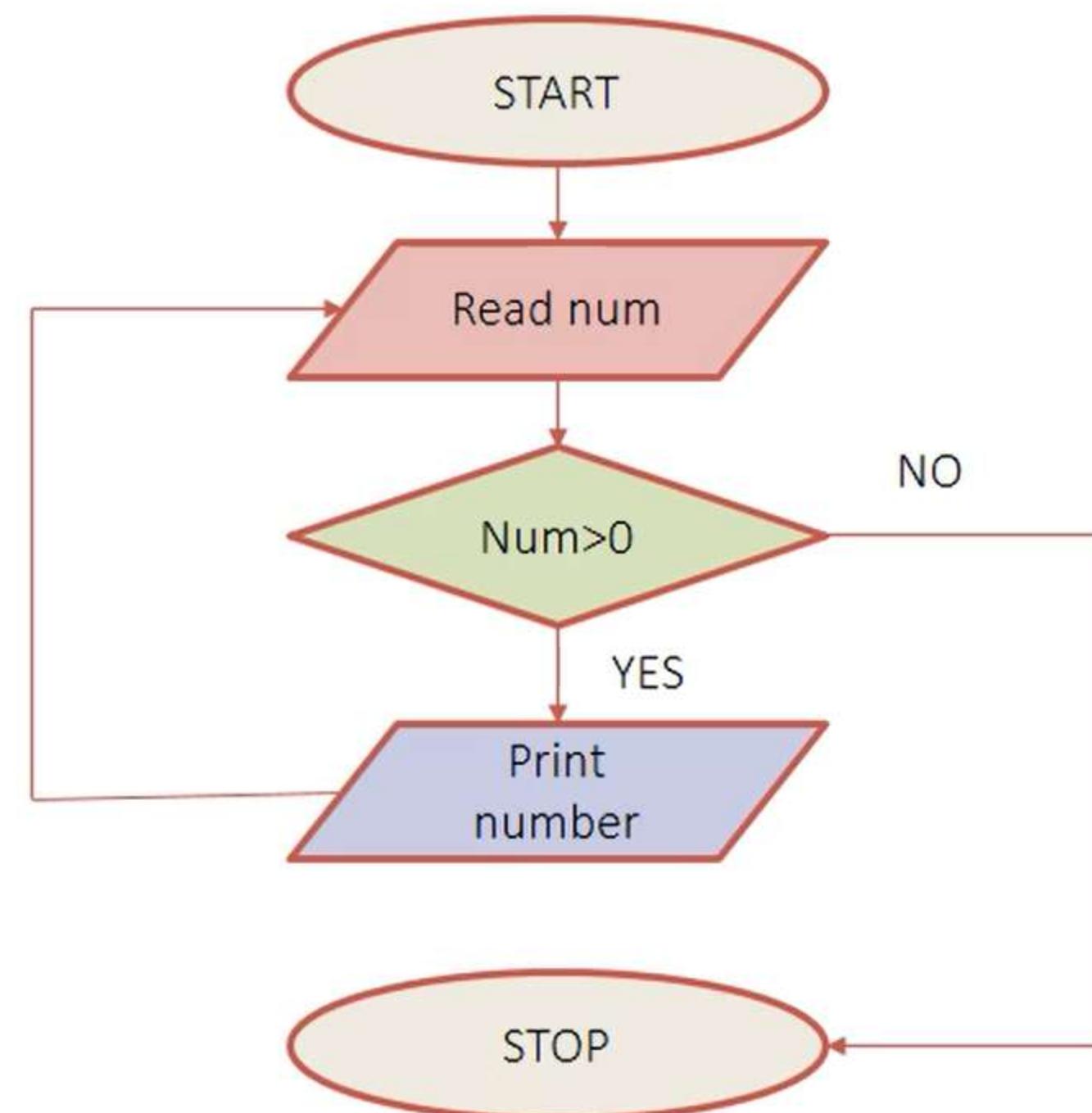
Example algorithm for while-loop logic



In this problem scenario the exact number of iterations for the execution is unknown. Loop has to continue until the user enters a valid number which meets the condition. Hence the logic of WHILE loop is used.

FLOWCHART FOR – WHILE LOOP

Basic flowchart for accepting the numbers and printing it only if the number is greater than 0



(Start the process)

(Let the variable `num` be num and the value for `num=5`)

(checks if num is greater than 0. Here 5 is greater than 0.)

(Prints 5 and the loop continues.. until user enters a negative value)

(End the process)

PSEUDOCODE FOR -WHILE LOOP

Pseudocode representation for accepting the numbers and printing it only if the number is greater than 0.

BEGIN

DECLARE variable number



READ number



WHILE number>0



PRINT number



READ number



END WHILE

END

WHILE LOOP –DRY RUN

Step no	Read number	Condition satisfied(Y/N) number<6	Output
1	1		
2		Y	
3			Prints 1
1	2		
2		Y	
3			Prints 2



The numbers 1 and 2 gets displayed as the numbers satisfy the condition that it should be greater than 0. That's why the numbers 1 and 2 get printed as output.

WHILE LOOP –DRY RUN

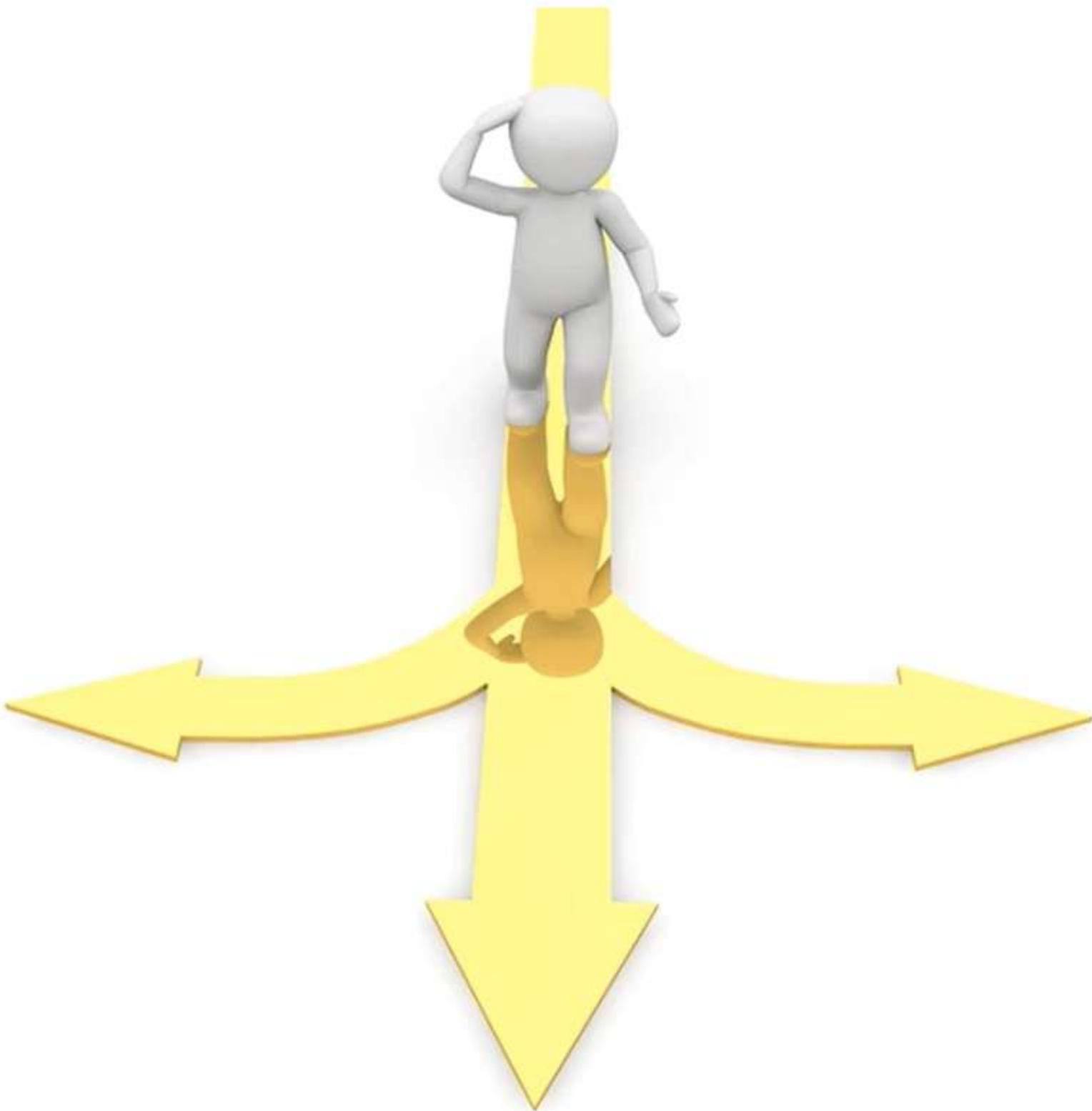
Step no	Read number	Condition satisfied(Y/N) number<6	Output
1	3		
2		Y	
3			Prints 3
1	-7		
2		N	
5	-	-	-



The number 3 get displayed but when the number is less than 0, i.e -7, in this case the loop get terminated and goes to the end of the process.

Time To think

Let's say, a user enters a n-digit number. What is the correct pseudo code to find the sum of digits?



TIME TO THINK

Pseudocode to calculate the sum of digits

DECLARE variables n, sum

BEGIN

PRINT sum

SET sum=0

WHILE n != 0

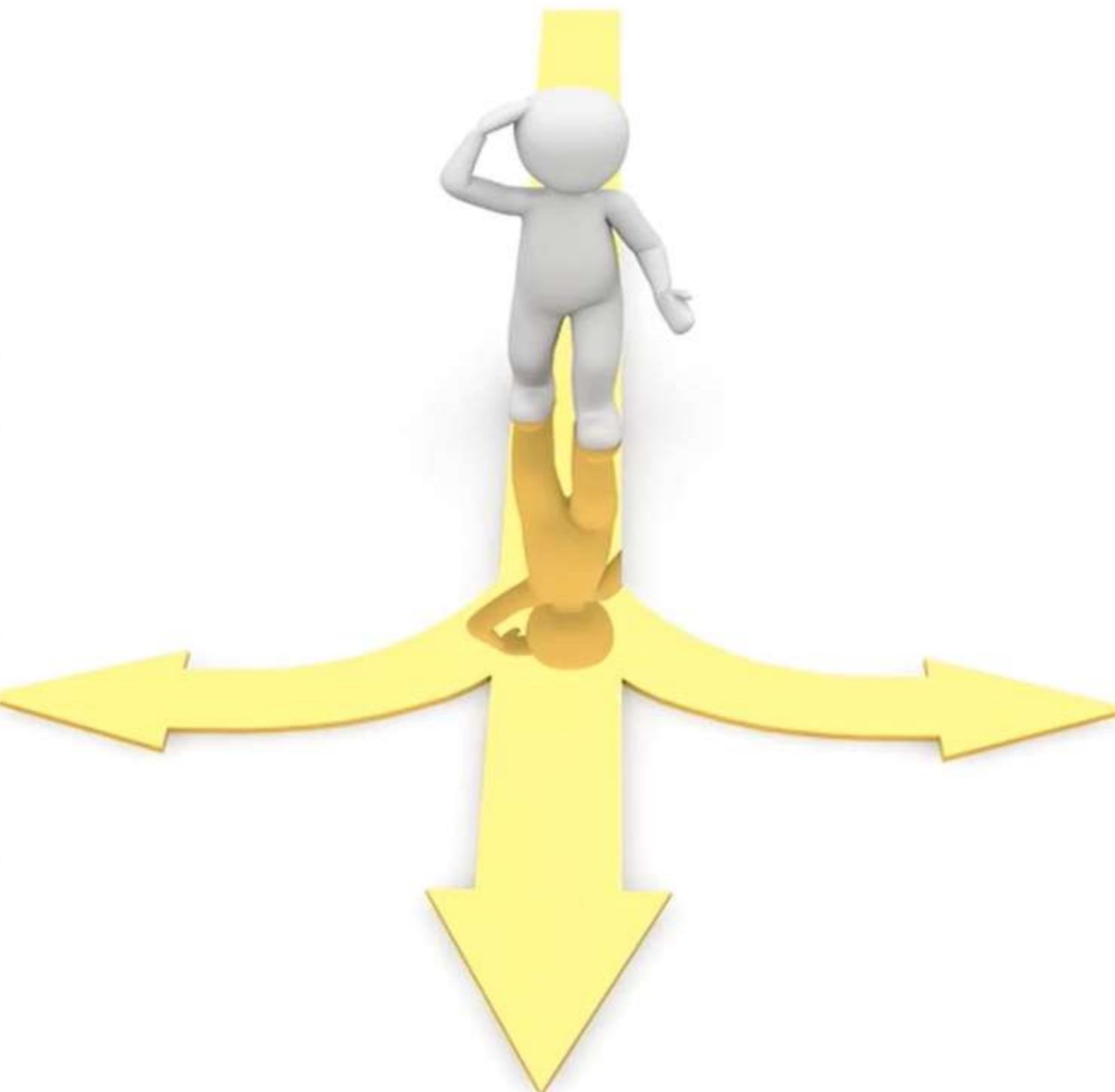
Sum= sum+(n%10)

n=n%10

READ n

END WHILE

END



TIME TO THINK

Pseudocode to calculate the sum of digits

BEGIN

DECLARE n, sum

SET sum=0

READ n

WHILE n != 0

Sum= sum+(n%10)

n=n%10

END WHILE

PRINT SUM

END



Let the number be 5843

$n=5843, sum=0$

$sum=sum+(n \% 10)=0+3; n=n/10= 584$

$sum=sum+(n \% 10)=3+4 ; n=n/10 = 58$

$sum=sum+(n \% 10)=7+8 ; n=n/10 = 5$

$sum=sum+(n \% 10)=15+5 \rightarrow n=n/10=0$

Output: 20

DO-WHILE -LOOP STATEMENT

More like a while statement.

Except that it tests the condition at the end of the loop, after executing the block of statements at least once.

- Code may be executed at least once or as many times as per the required condition.
- Do-While loop is exit controlled as condition is checked after entering the block of statements.



ALGORITHM- DO WHILE LOOP

Sample algorithm for accepting the numbers and printing it at least once before checking the condition

STEP 1:

Start the process

STEP 2:

Get the number

STEP 3:

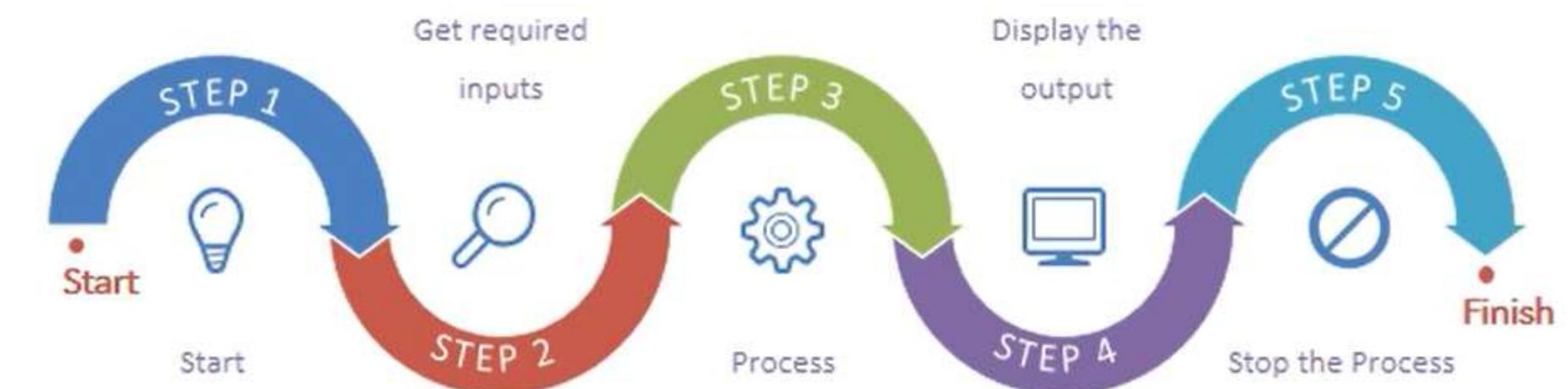
Display the number. GOTO Step 4

STEP 4:

Check whether number<6,if YES GOTO step 2, if NO
GOTO step 5

STEP 5:

Stop the process

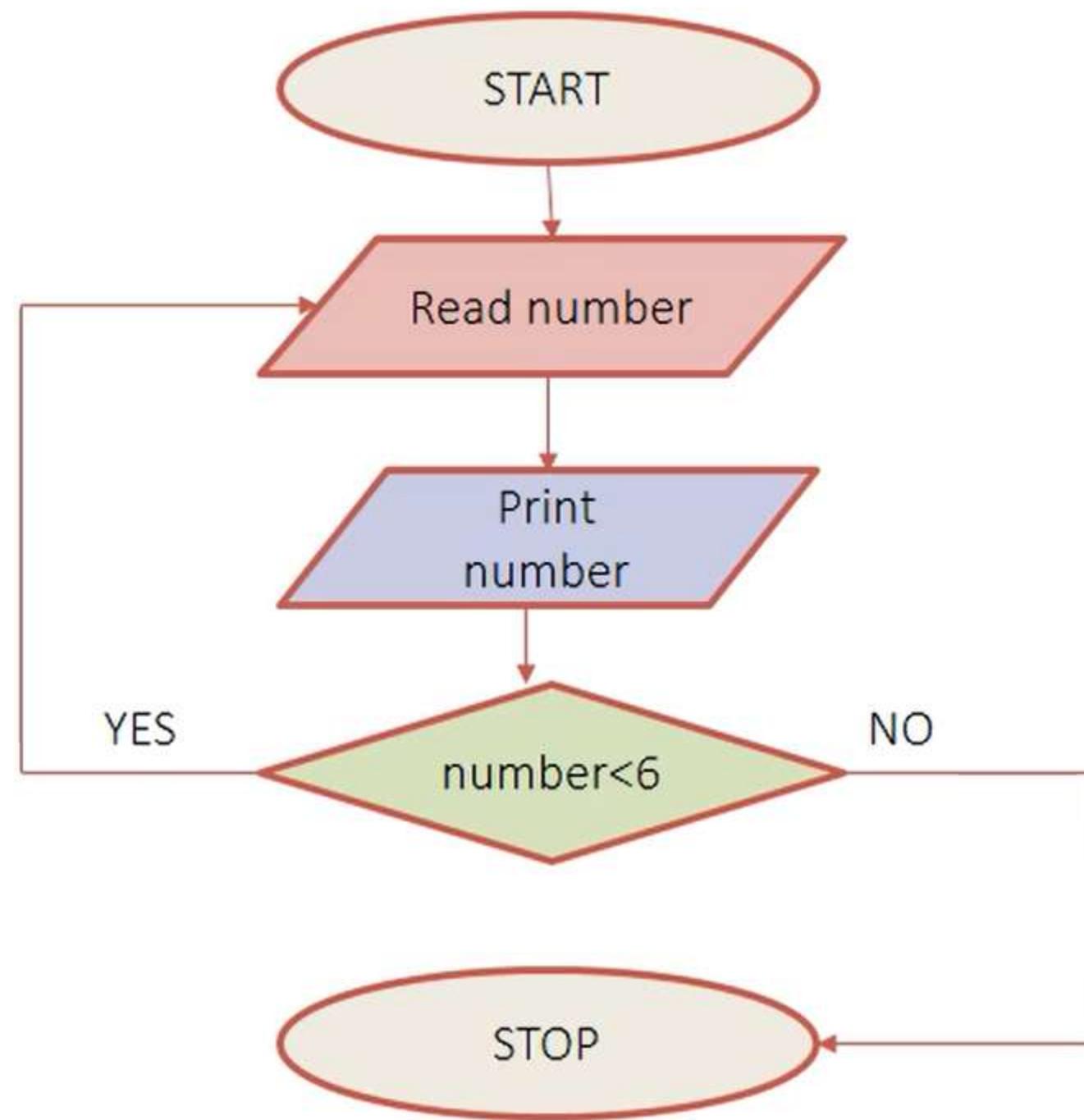


Example algorithm for Do while-loop logic

- In this problem scenario the statements execute at least once, before checking the actual condition in the loop. Loop has to continue until the user enters a valid number which meets the condition. Hence the logic of DO WHILE loop is used.

FLOWCHART FOR – DO WHILE LOOP

Basic flowchart for accepting the numbers and printing it at least once before checking the condition



(Start the process)

(Let the variable be number and the value for
number=5)

(Prints 5)

(checks if number is less than 6. Here 5 is less
than 6. and the loop continues..)

(End the process)

PSEUDOCODE FOR –DO WHILE LOOP

Pseudocode representation for accepting the numbers and printing it only if the number is less than 6.

BEGIN

DECLARE variable number

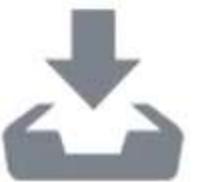
DO

READ number

PRINT number

UNTIL number<6

END

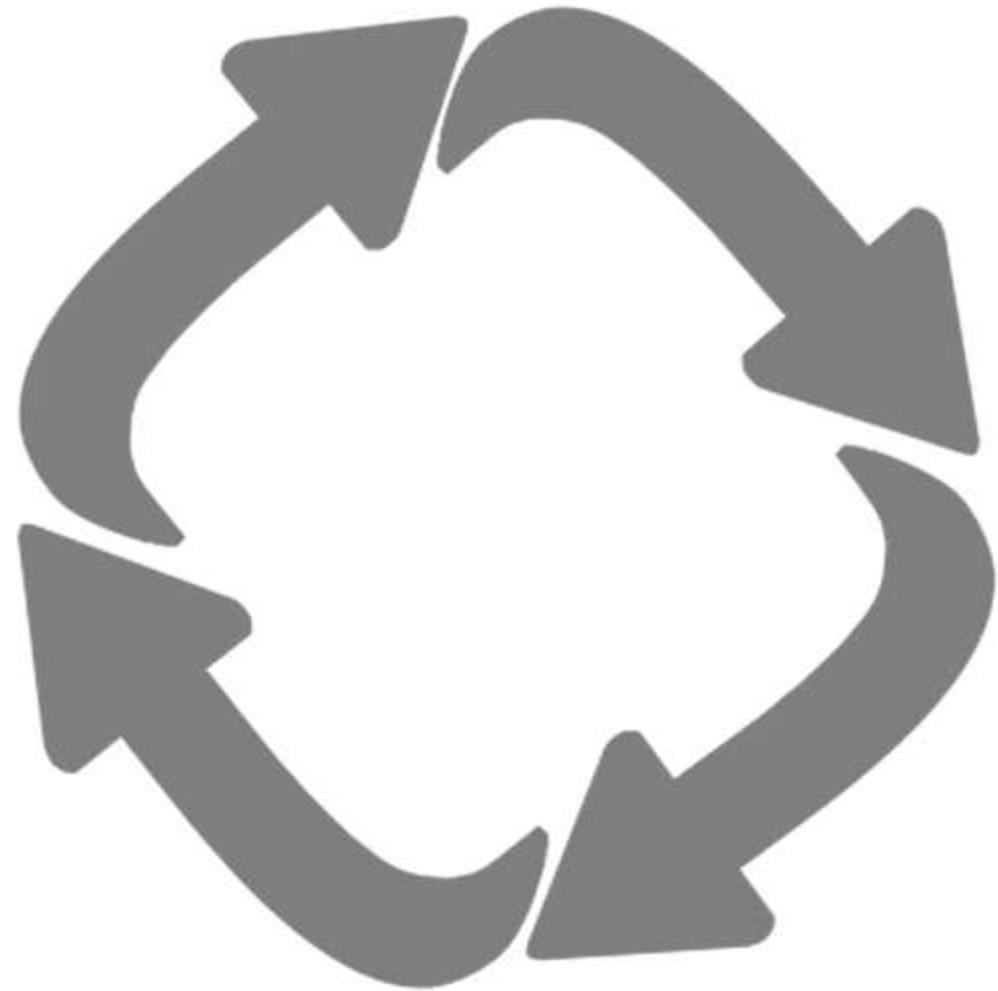


Nested-LOOP Statement

Used when one or more loops are needed inside another looping statements like while, for, or do.. while .
Loops can be nested to any level.

-  A final note on loop nesting is that you can put any type of loop inside any type of loop. For example, a 'for' loop can be inside a 'while' loop or vice versa.

-  How does this work - the first pass of the outer loop triggers the inner loop, which executes to completion. Then the second pass of the outer loop triggers the inner loop again.



PSEUDOCODE FOR -NESTED FOR LOOP

Pseudocode representation for Printing pattern

BEGIN

DECLARE variables i, j, size

SET size=3

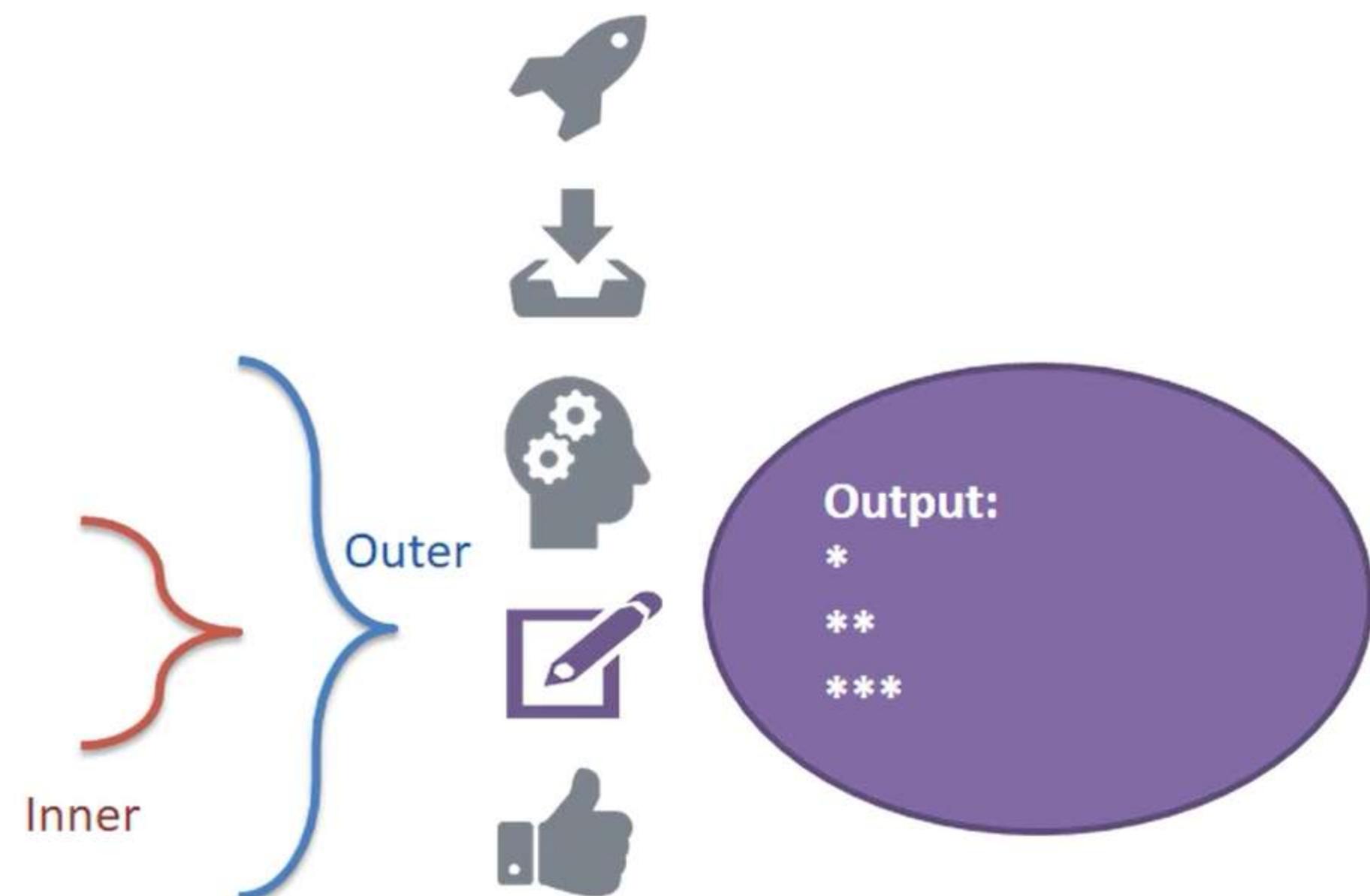
FOR i=0 to size DO
 FOR j=0 to (i+1) DO
 PRINT "**"

END FOR

PRINT newline // Skips to new line

END FOR

END



NESTED FOR LOOP –DRY RUN

Let the size be 3

Step no	i	Condition satisfied(Y/N) i<size	j	Condition satisfied(Y/N) j<i+1	Output	j=j+1	i=i+1
1	0	Y					
2			0	Y			
3					Prints *	j=0+1	
4			1	N			
5					Prints newline		i=0+1
1	1	Y					
2			0	Y			
3					Prints *	j=0+1	
4			1	Y			
3					Prints *	j=1+1	
4			2	N			

NESTED FOR LOOP –DRY RUN

Let the size be 3

Step no	i	Condition satisfied(Y/N) <i>i</i><size	j	Condition satisfied(Y/N) <i>j</i><<i>i</i>+1	Output	<i>j=j+1</i>	<i>i=i+1</i>
5					Prints newline		<i>i=1+1</i>
1	2	Y					
2			0	Y			
3					Prints *	<i>j=0+1</i>	
4			1	Y			
3					Prints *	<i>j=1+1</i>	
4			2	Y			
3					Prints *	<i>j=2+1</i>	
4			3	N			
5	3	N					

NESTED FOR LOOP –DRY RUN

Let the size be 3

Step no	i	Condition satisfied(Y/N) i<size	j	Condition satisfied(Y/N) j<i+1	Output	j=j+1	i=i+1
4			2	N			
5					Prints newline		i=1+1
1	2	Y					
2			0	Y			
3					Prints *	j=0+1	
4			1	Y			
3					Prints *	j=1+1	
4			2	Y			
3					Prints *	j=2+1	
4			3	N			

Nested-LOOP Statement

- ④ Outer loops execute first, based on the condition.
- ④ Outer loop triggers the inner loop to execute completely until the condition fails in the inner loop.
- ④ After which it again comes back to the outer loop for a second outer iteration and triggers the inner loop again .
- ④ This process continues until the condition fails in the outer loop.



Looping structures



Initial conditions that needs to be applied before the loop begins to execute

The invariant relation that must be applied after each iteration of the loop

The condition under which the iterative process must be terminated

Summary

- Looping statements
- Categories of Looping statements
- How to construct algorithm, flowchart and pseudo codes
for Looping statements





Press Esc to exit full screen

LOGIC DEVELOPMENT

INTRODUCTION TO ARRAYS



An array is a variable name which is used to store large amount of data

An array can be considered as a container with equally spaced slots, where the data could be stored

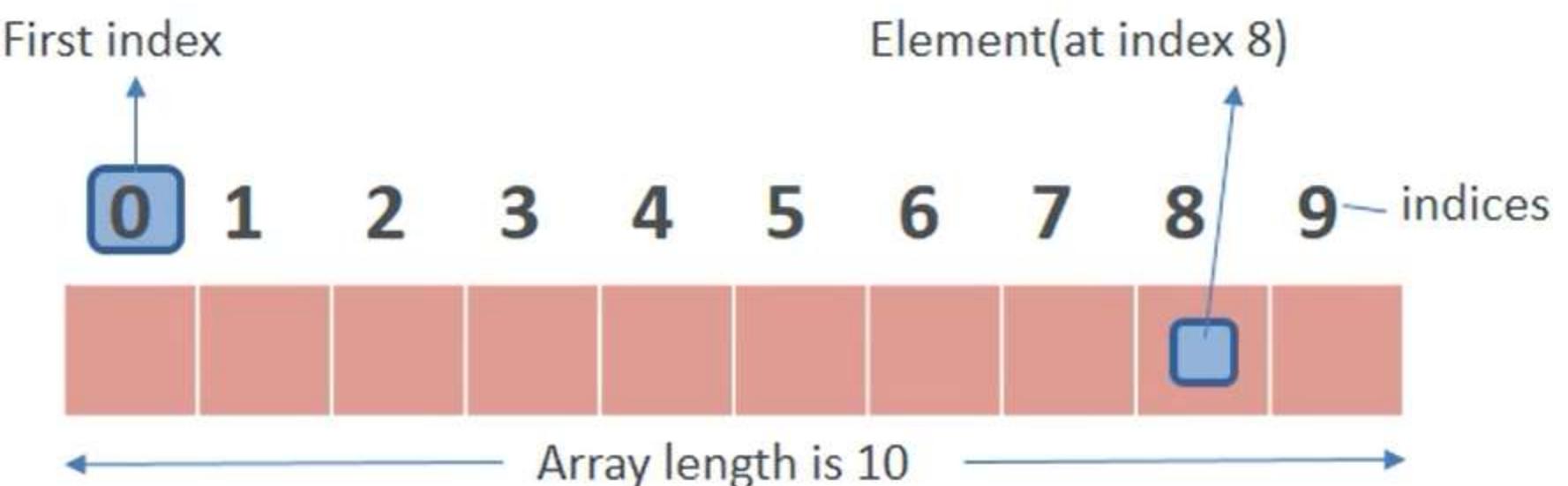
Data would be stored continuously, and the type of data would be the same

ONE-DIMENSIONAL ARRAYS

1-d arrays are arrays treated as a linear list of values

Elements are stored sequentially and can be accessed using INDEX of the array

All locations in an array are numbered from 0 to n-1, where n is the total number of elements.



- Assume that an array variable marks[10] is created for storing marks of 10 subjects,
- To store a mark value 93 at the 1st position, the assignment of value would be marks[0]←93
- To access location 1, we would use notation as array_variable_name[index] i.e., marks[0]

**Accessing values in
1d array**

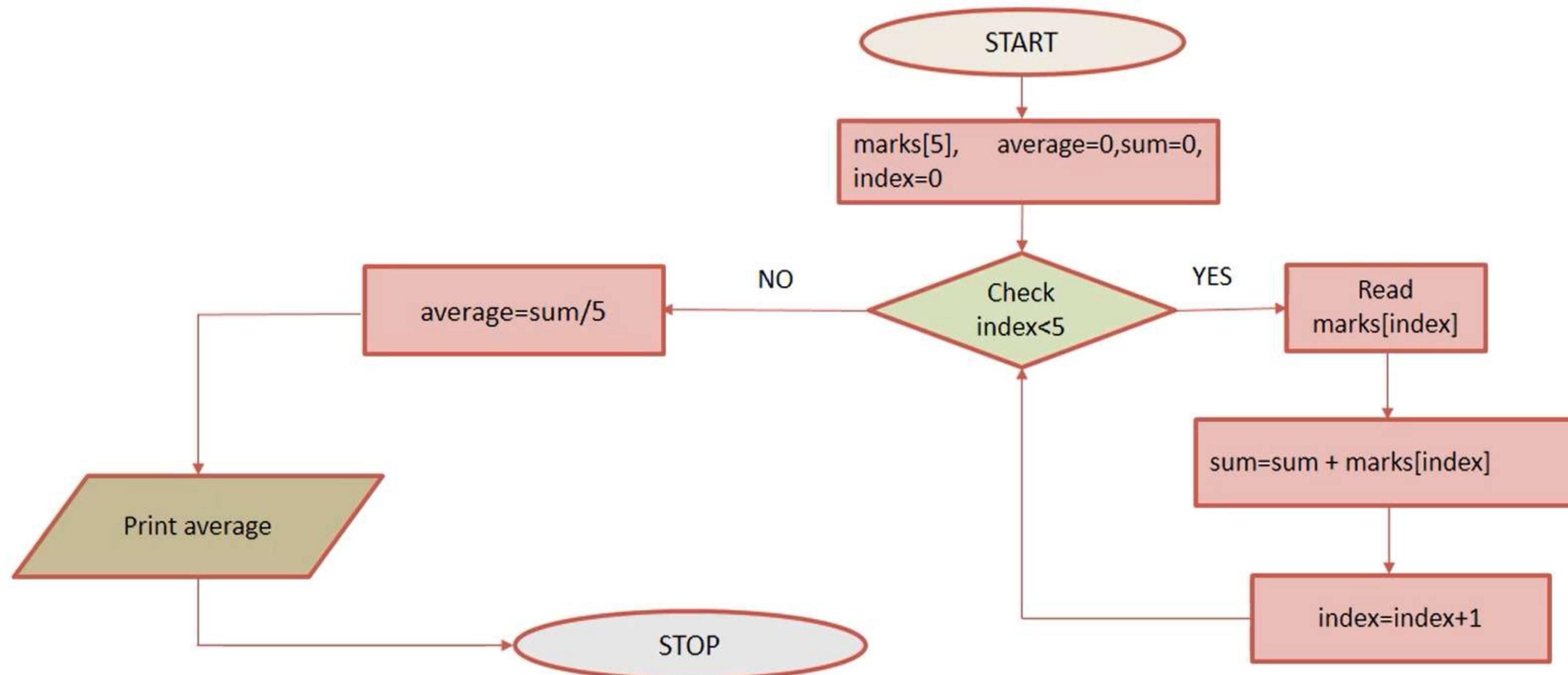
SAMPLE ALGORITHM FOR 1-D ARRAY

To read the marks of 5 students in a group and print the average for that group. Assumption: marks is an array of 5 marks.

-  **STEP 1:**
Start the process. Create array as marks[5]
-  **STEP 2:**
Use variables index, average, sum. Set the index value as 0, sum value as 0. average value as 0.
-  **STEP 3:**
Check whether $i < 5$, if YES GOTO step 4, if NO GOTO step 7.
-  **STEP 4:**
Read marks[index].
-  **STEP 5:**
ADD sum and marks[index] and store the result in sum.
-  **STEP 6:**
Increment the index value by 1. GOTO step 3
-  **STEP 7:**
Divide the sum value by 5 and store the result in average.
-  **STEP 8:**
Display the average value
-  **STEP 9:**
Stop the process

FLOWCHART FOR 1-D ARRAY

Basic flowchart to read the marks of 5 students and print the average for that group. Assumption: marks is an array of 5 marks



SAMPLE PSEUDOCODE FOR 1-D ARRAY

To read the marks of 5 students and print the average. Assumption: marks is an array of 5 marks

BEGIN

DECLARE variables index , sum, average, marks[5]

SET index=0,sum=0

FOR index \leftarrow 0 to 4 do

 READ marks[index]

 sum \leftarrow sum + marks[index]

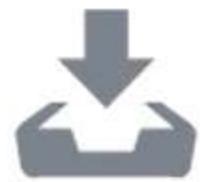
 i \leftarrow i+1

END FOR

AVERAGE \leftarrow sum/5

PRINT average

END



1D ARRAY USING FOR LOOP—DRY RUN

Step no	index	Sum	Condition satisfied(Y/N) i<5	Read marks[index]	sum = sum+ marks[index]	i=i+1	Average = sum/5	Output
1	0	0						
2	0	0	Y					
3				98				
4					0+98=98			
5						0+1=1		
2	1	98	Y					
3				87				
4					98+87=185			
5						1+1=2		



Step no. 2,3,4 and 5 are the block of statements which repeats for getting the marks of five subjects.

1D ARRAY USING FOR LOOP—DRY RUN

Step no	index	Sum	Condition satisfied(Y/N) i<5	Read marks[index]	sum = sum+ marks[index]	i=i+1	Average = sum/5	Output
2	2	185	Y					
3				95				
4					185+95=280			
5						2+1=3		
2	3	280	Y					
3				87				
4					280+87=367			
5						3+1=4		



Step no. 2,3,4 and 5 are the block of statements which repeats for getting the marks of five subjects.

1D ARRAY USING FOR LOOP—DRY RUN

Step no	index	Sum	Condition satisfied(Y/N) i<5	Read marks[index]	sum = sum+ marks[index]	i=i+1	Average = sum/5	Output
2	4	367	Y					
3				88				
4					367+88=455			
5						4+1=5		
2	5	455	N					
7							91	
8								Prints 91



Step no. 2,3,4 and 5 are the block of statements which repeats for getting the marks of five subjects.

PSEUDOCODE FOR 1-D ARRAY- FINDING MAXIMUM

Assume the size of an array is 10. Given Pseudocode To find the maximum element in an array of 10 elements

BEGIN

 DECLARE variables index , maximum, array[10]

 FOR index ← 0 to 9 DO

 READ array[index]

 END FOR

 SET maximum ← array[0]

 FOR index ← 0 to 9 DO

 IF maximum < array[index]

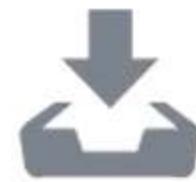
 maximum ← array [index]

 END IF

END FOR

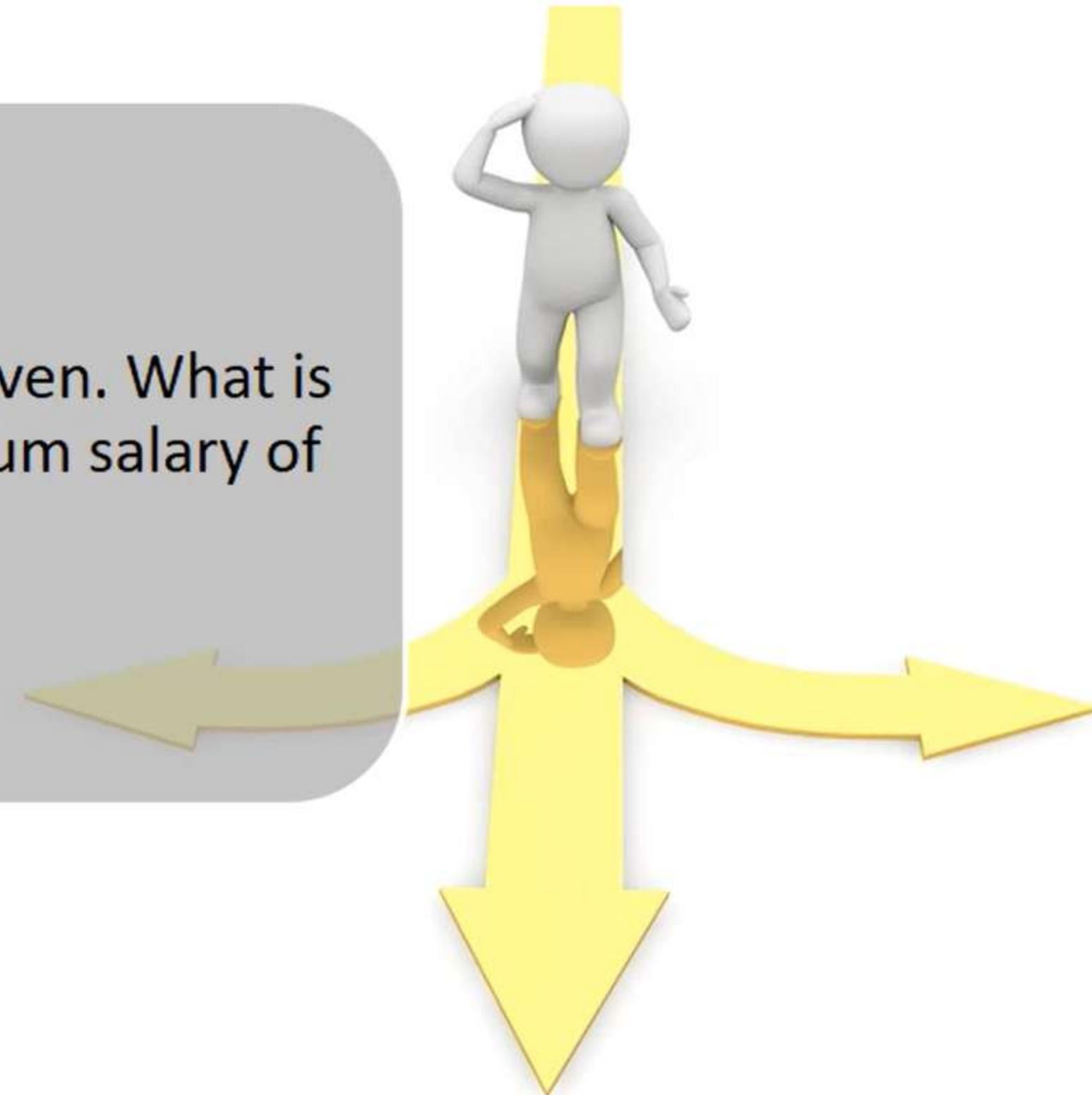
PRINT maximum

END



Time To think

Let's say, The salary of 10 employees are given. What is the correct pseudo code to find the minimum salary of employees?



TIME TO THINK

Assume the size of an array is 10.

```
DECLARE variables index , minimum_salary, array[10]
```

```
BEGIN
```

```
PRINT minimum_salary
```

```
SET minimum_salary ← array[0]
```

```
FOR index ← 0 to 9 DO
```

```
IF minimum_salary > array[index]
```

```
FOR index ← 0 to 9 DO
```

```
READ array[index]
```

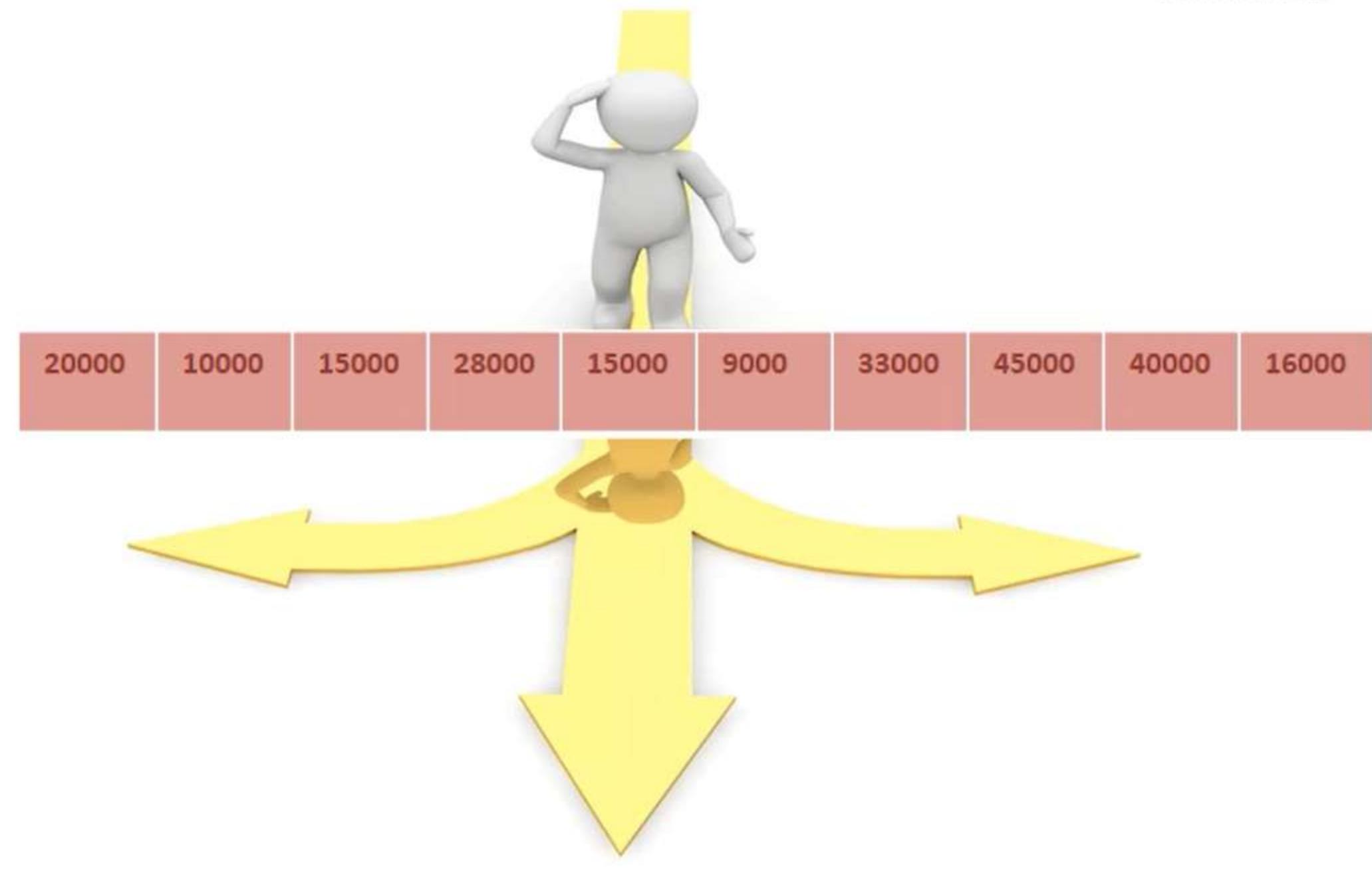
```
END FOR
```

```
Minimum_salary ← array [index]
```

```
END FOR
```

```
END
```

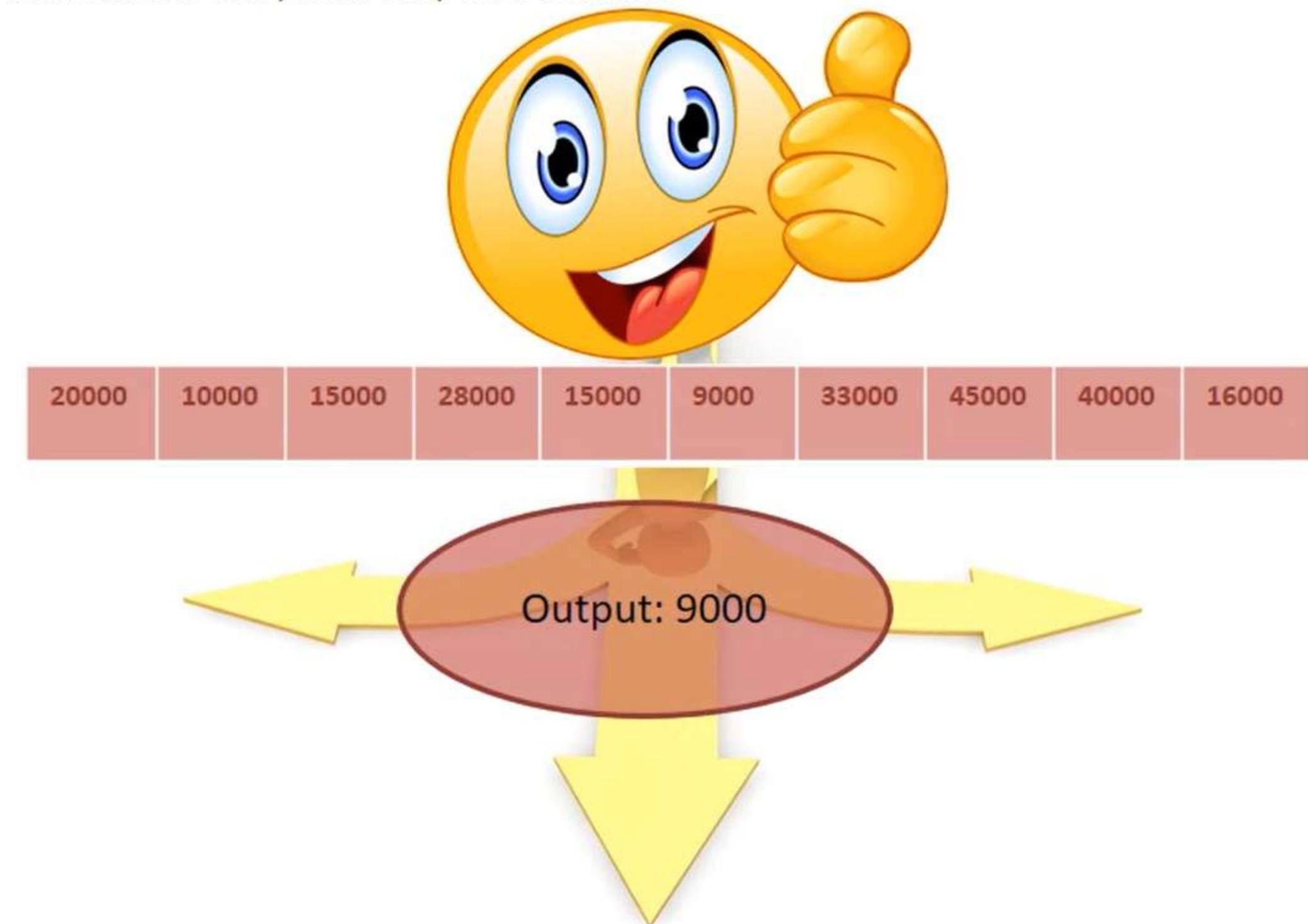
```
END IF
```



TIME TO THINK

Assume the size of an array is 10. Given Pseudocode To find the minimum salary in an array of 10 elements

```
BEGIN
  DECLARE variables index , minimum_salary, array[10]
  FOR index ← 0 to 9 DO
    READ array[index]
  END FOR
  SET minimum_salary← array[0]
  FOR index ← 0 to 9 DO
    IF minimum_salary > array[index]
      minimum_salary← array [index]
    END IF
  END FOR
  PRINT minimum_salary
END
```



ARRAYS -EXAMPLE

Two Arrays are said to be compatible

- if they are of the same size
- if the i'th element in the first array is greater than or equal to the i'th element in the second array for all i values.



Array 1

2	3	8	6	1
---	---	---	---	---

Array 2

1	1	1	1	1
---	---	---	---	---

- ✓ Size of array 1 is equal to size of array 2
- ✓ 1st column : $2 > 1$
- ✓ 2nd column : $3 > 1$
- ✓ 3rd column : $8 > 1$
- ✓ 4th column : $6 > 1$
- ✓ 5th column : $1 \geq 1$

Compatible

PSEUDOCODE FOR COMPATIBLE ARRAYS

Assume the size of two arrays is 15.

```
BEGIN  
    DECLARE variables i , number, a[15], b[15], flag  
    GET number  
    SET flag=0  
    FOR i ← 0 to number-1 DO  
        READ a[i]  
    END FOR  
    FOR i ← 0 to number-1 DO  
        READ b[i]  
    END FOR  
  
    FOR index ← 0 to number-1 DO  
        IF a[i] < b[i]  
            SET flag=1  
            BREAK  
        END IF  
    END FOR  
    IF flag==0  
        PRINT Compatible  
    ELSE  
        PRINT Incompatible  
    END
```



2-D Array

The two-dimensional array is treated like a matrix.

The values are arranged in rows and columns.

	0	1	2	3
0	45	22	-5	11
1	-1	45	89	23
2	0	34	90	76

An array which has 3 rows and 4 columns

2-D Array

Pseudo code for reading the marks of 3 students for
5 subjects and printing the average of each student

(contd...)

```
FOR student_index= 0 TO 2
    FOR mark_index=0 TO 4
        SET sum TO sum+ student_marks[student_index][mark_index]
    NEXT mark_index
    SET average TO sum/5
    PRINT average
NEXT student_index
END
```

Manipulating Arrays



great
Feel the difference.



WHAT IS SEARCHING?

Process of finding a particular item in a collection of items

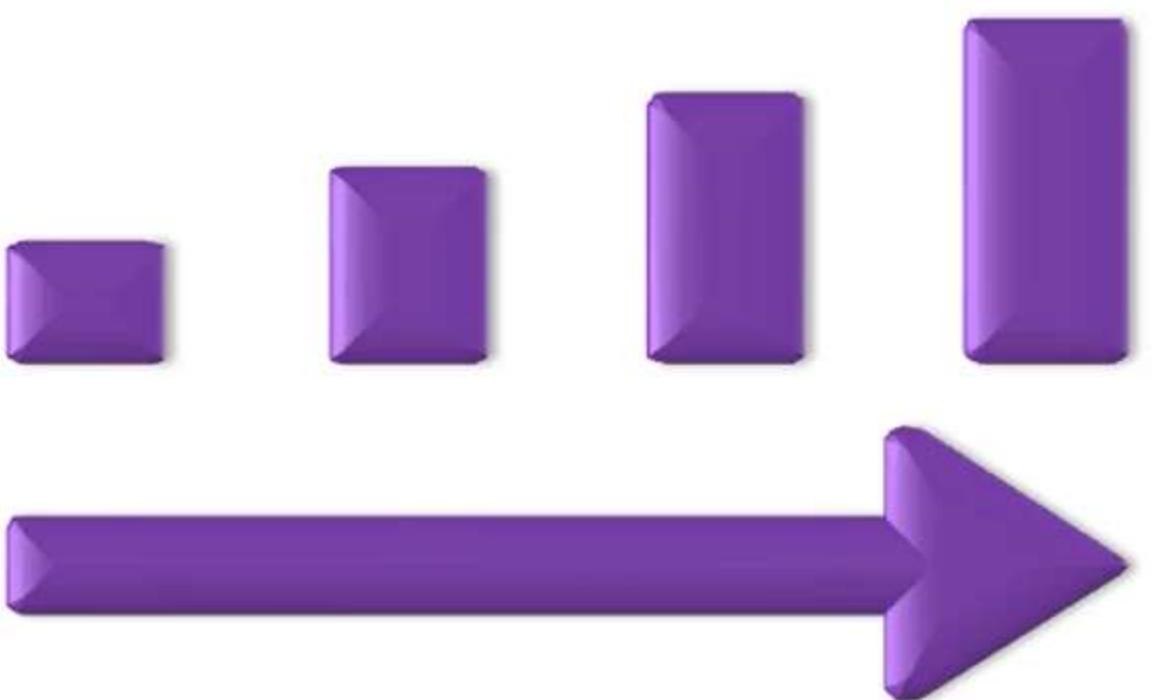
- A search typically answers either True or False depending on whether the item is present or not.
- Search operation can be done using searching algorithms



SEARCHING

If an element is to be found from a specific location in an array, the array has to be traversed from the first position until the element is found.

- It sequentially scans the array , comparing each array item with the search element.
- If search element is found in an array then the index of that element is returned.



ALGORITHM FOR SEARCHING



STEP 1:

Start the process



STEP 2:

Create an array variable employee_id[8], i , n



STEP 3:

Set i value as 0.



STEP 4:

Check whether $i < 5$, if YES GOTO step 5 and if NO GOTO step 14.



STEP 5:

Read employee_id[i]



STEP 6:

Increment i value by 1, and GOTO 4



STEP 7:

Get n from user (number to be searched)



STEP 8:

Get n from user (number to be searched)



STEP 9:

Set i value as 0 and Check whether $i < 5$, if YES GOTO step 10, if NO GOTO step 13



STEP 10:

Compare n value with employee_id[i], if Equal GOTO step 11 and not EQUAL GOTO step 12

ALGORITHM FOR SEARCHING



STEP 11:

Display "Number is present". GOTO step 14

3



STEP 12:

Increment i value by 1 and GOTO step 9

3



STEP 13:

Display "Number is not present in an array"

3



STEP 14:

Stop the process.

3

3

6	4	1	9	7	3	2	8
---	---	---	---	---	---	---	---

6	4	1	9	7	3	2	8
---	---	---	---	---	---	---	---

6	4	1	9	7	3	2	8
---	---	---	---	---	---	---	---

6	4	1	9	7	3	2	8
---	---	---	---	---	---	---	---

6	4	1	9	7	3	2	8
---	---	---	---	---	---	---	---

6	4	1	9	7	3	2	8
---	---	---	---	---	---	---	---

Sorting is the process of placing elements from a collection
in some kind of order

- Ordering of data allows for easy and faster access of data

Eg:

- Sorting names in telephone directory
- Sorting list of cities by using population

Sorting - Activities

A sort repeatedly compares adjacent elements of an array



Go through multiple passes over the array.

In every pass:

- Compare adjacent elements in the list
- Exchange the elements if they are out of order
- Each pass moves the largest elements to the end or smallest element to the beginning.



After every pass, all elements after the last swap are sorted and do not need to be checked again, thereby skipping to track swapped variables.



Sorting - Activities

A sort repeatedly compares adjacent elements of an array



Go through multiple passes over the array.

In every pass:

- Compare adjacent elements in the list
- Exchange the elements if they are out of order
- Each pass moves the largest elements to the end or smallest element to the beginning.



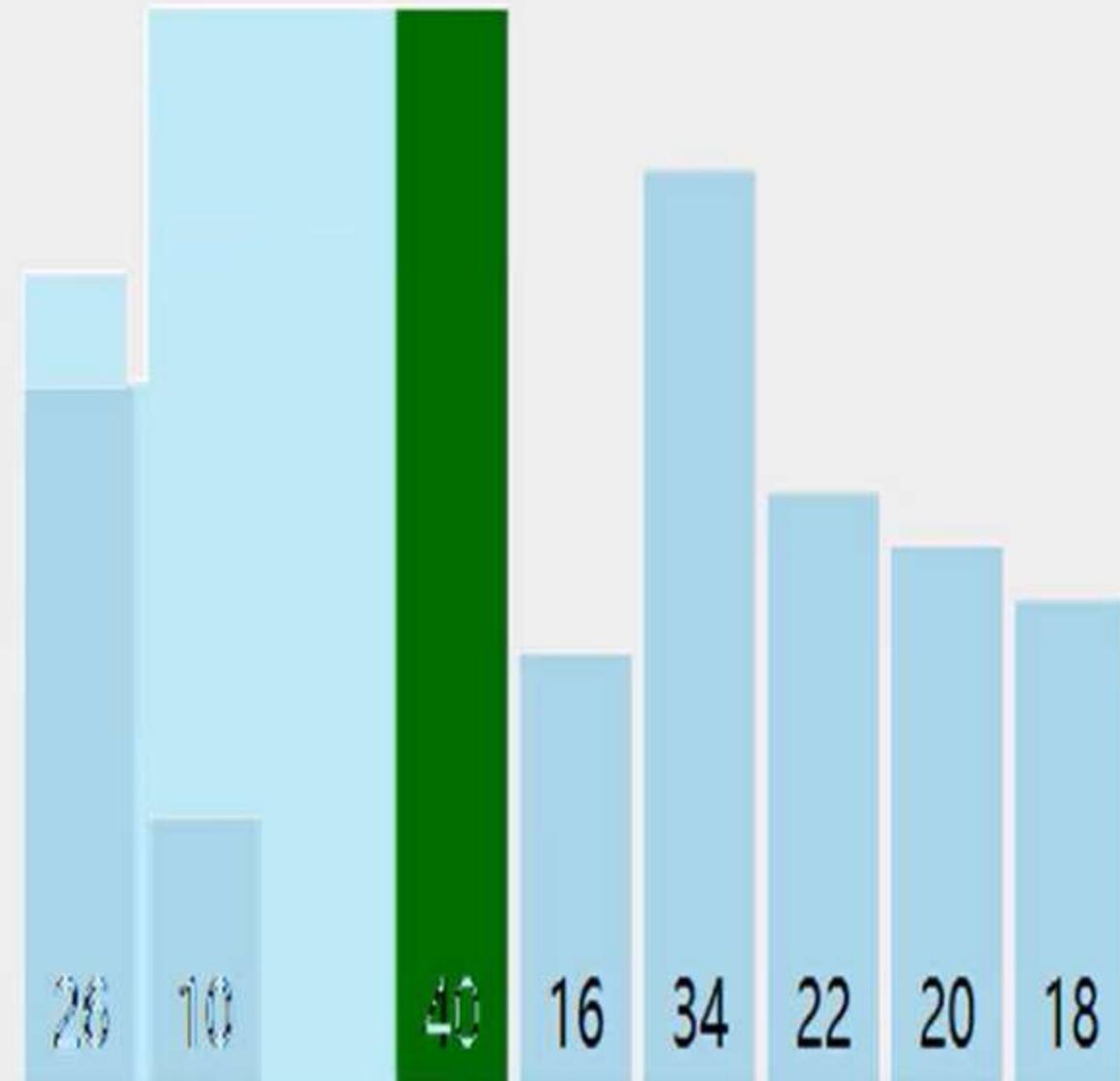
After every pass, all elements after the last swap are sorted and do not need to be checked again, thereby skipping to track swapped variables.



Sorting- Algorithm

Let A be a linear array of n numbers. Swap is a temporary variable for swapping (or interchange) the position of the numbers

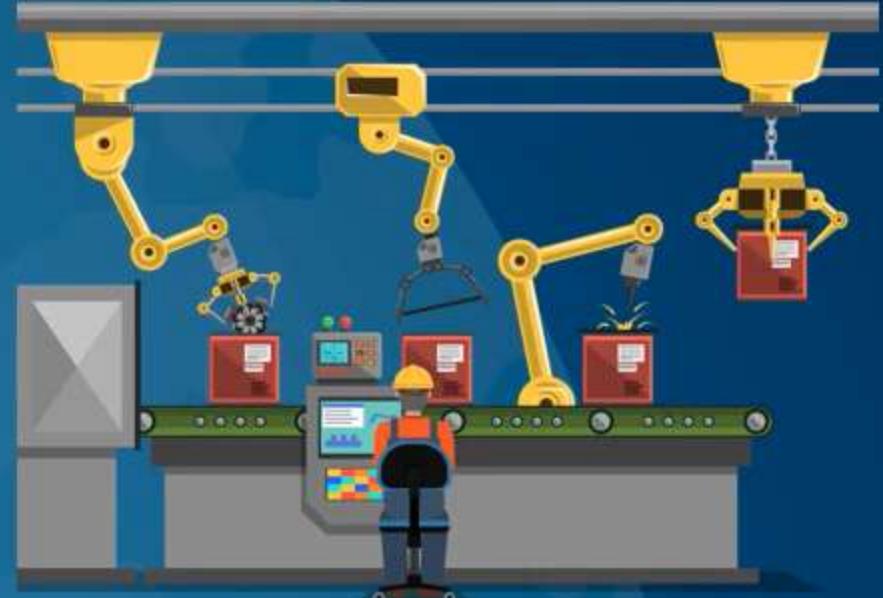
1. Input n numbers of an array A
2. Initialize $i = 0$ and repeat through step 4 if ($i < n$)
3. Initialize $j = 0$ and repeat through step 4 if ($j < n - i - 1$)
4. If ($A[j] > A[j + 1]$)
 - (a) $Swap = A[j]$
 - (b) $A[j] = A[j + 1]$
 - (c) $A[j + 1] = Swap$
5. Display the sorted numbers of array A
6. Exit.



Summary

- Introduction to Arrays
 - 1-Dimension and 2-Dimensional arrays
 - Algorithm for manipulating arrays
- Searching and Sorting Techniques





Entertainment



How these networks are created
&
The technologies they use ?

How these networks are created & The technologies they use ?





1
to
10 Devices

Size



100



- ✓ Networking Technologies
- ✓ Protocols
- ✓ Services



Wired Vs Wireless





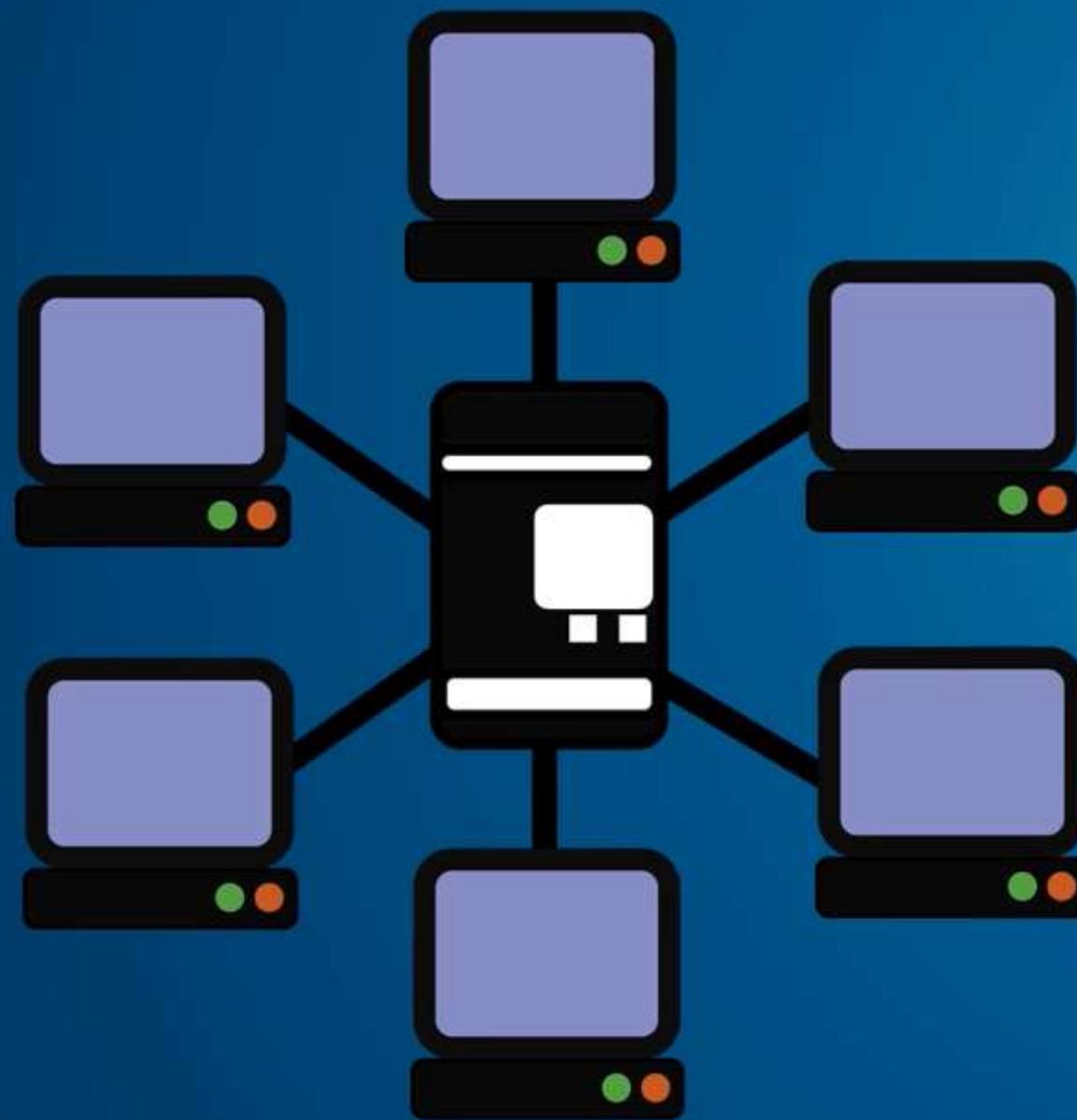
Wired Vs Wireless



Designing Networks

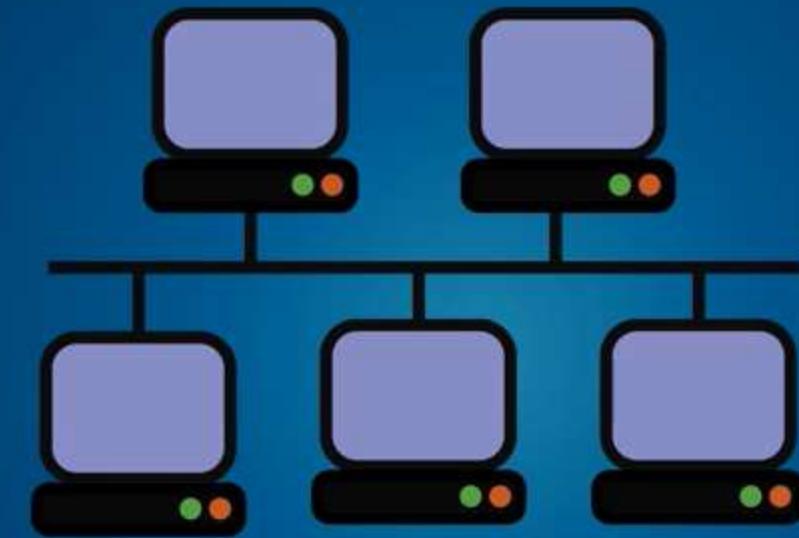
&

Selecting Connection Protocols

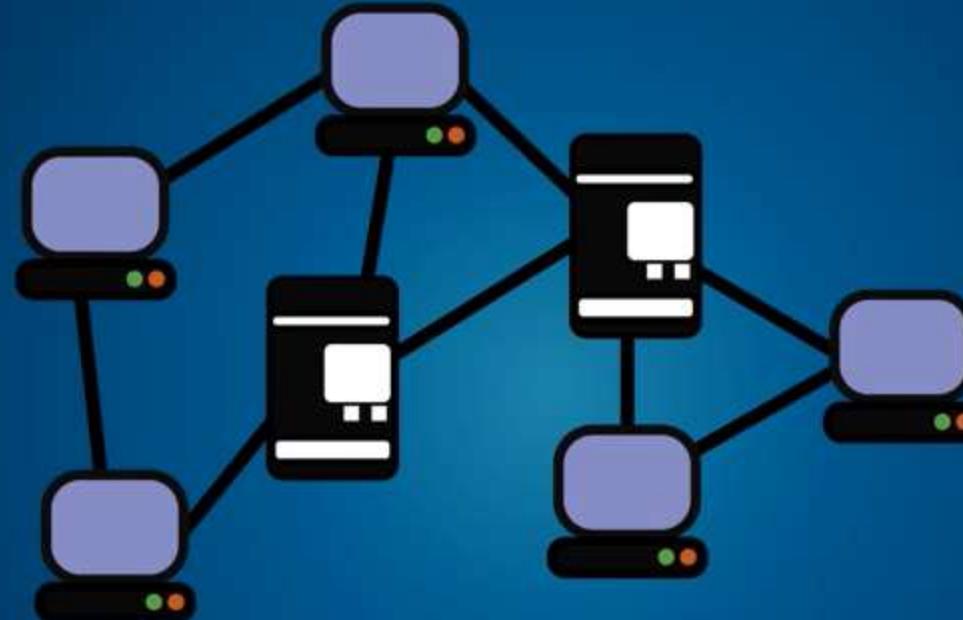


Network Topology

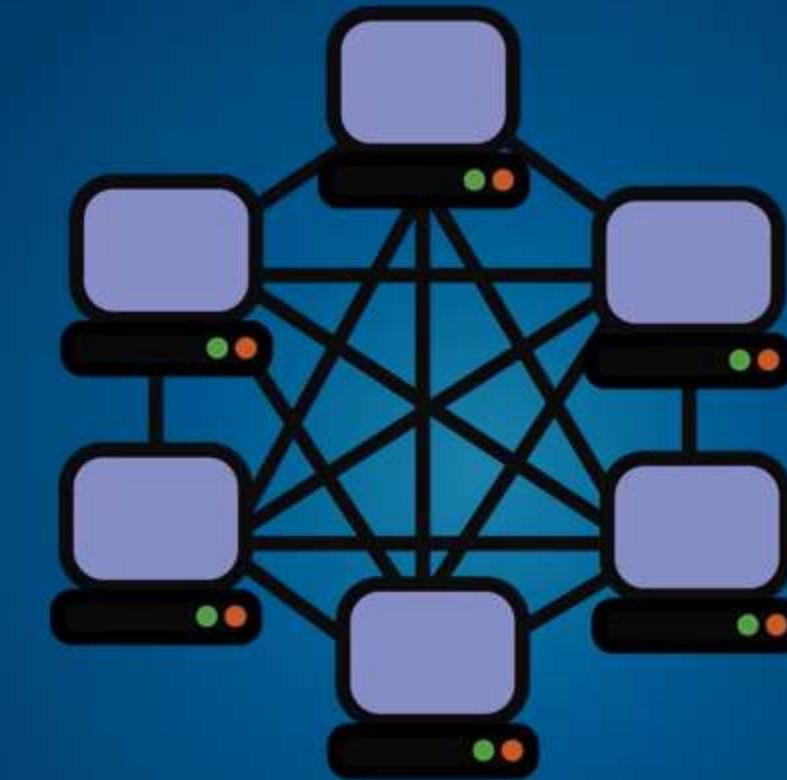




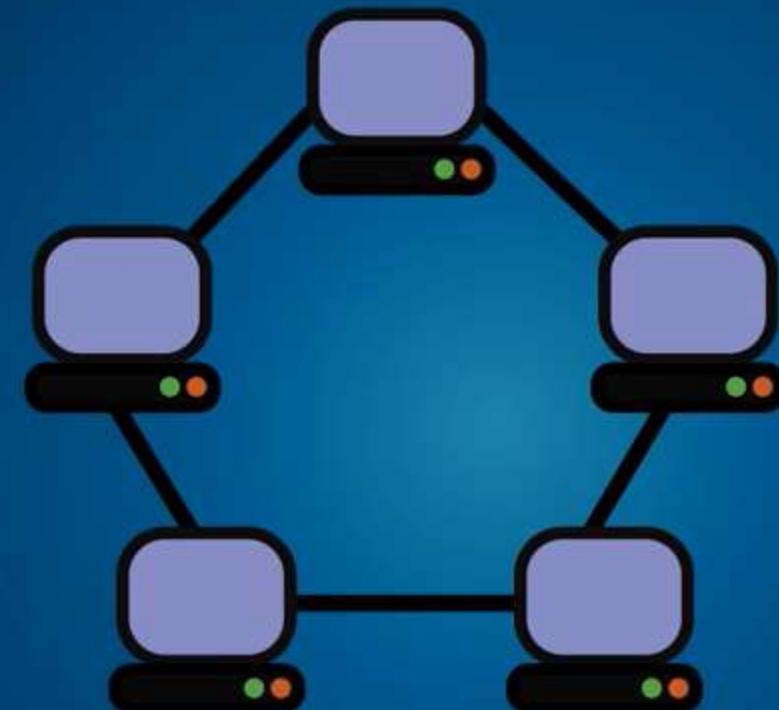
Bus



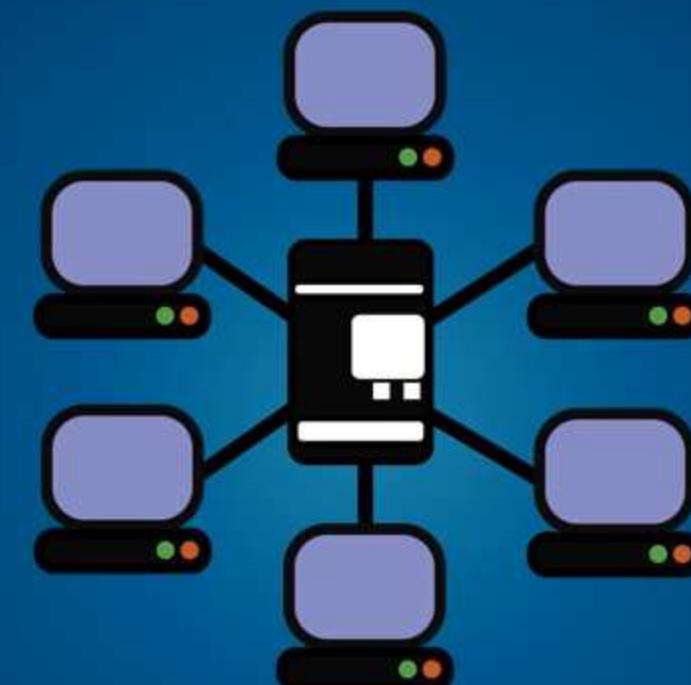
Mesh



Hybrid

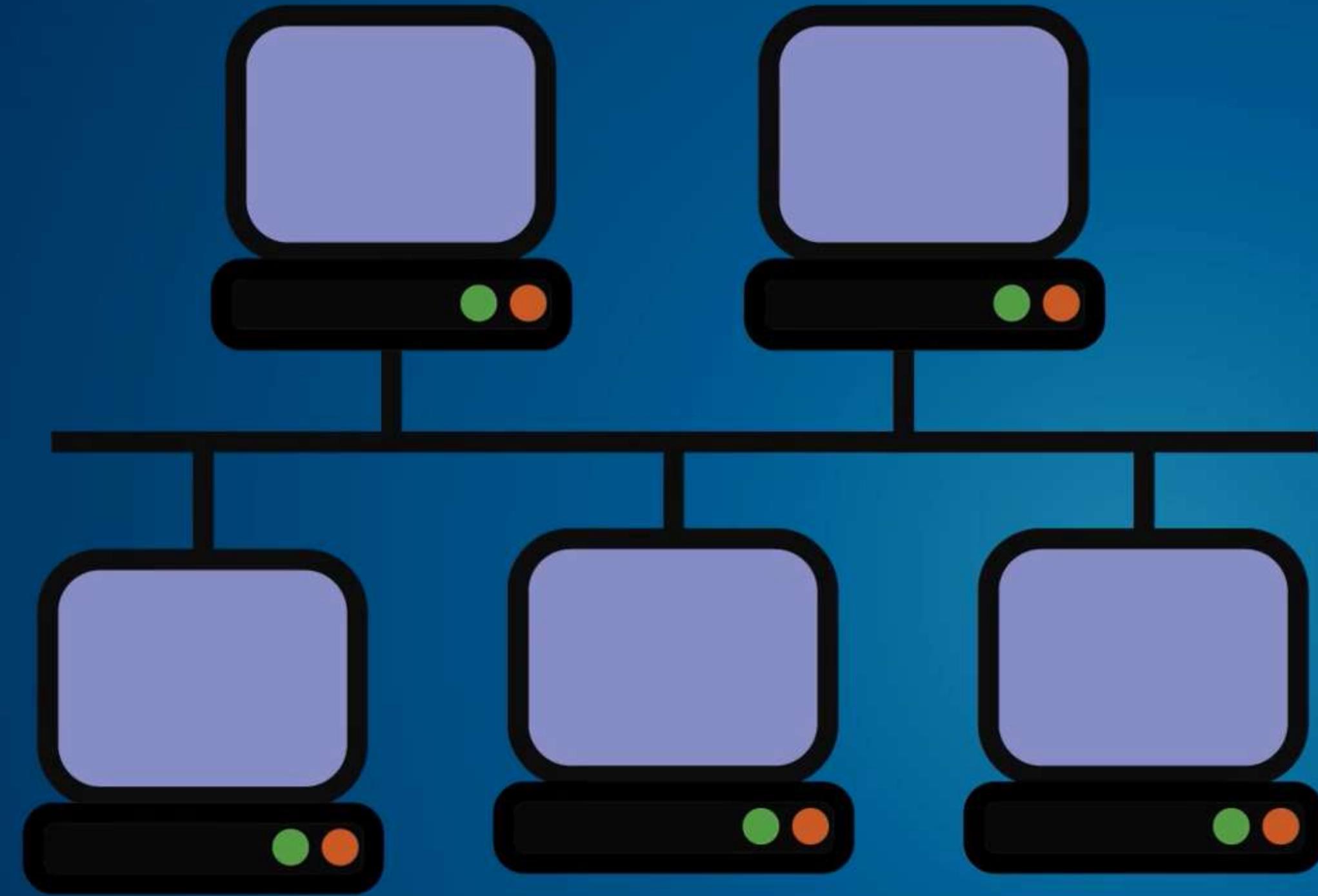


Ring



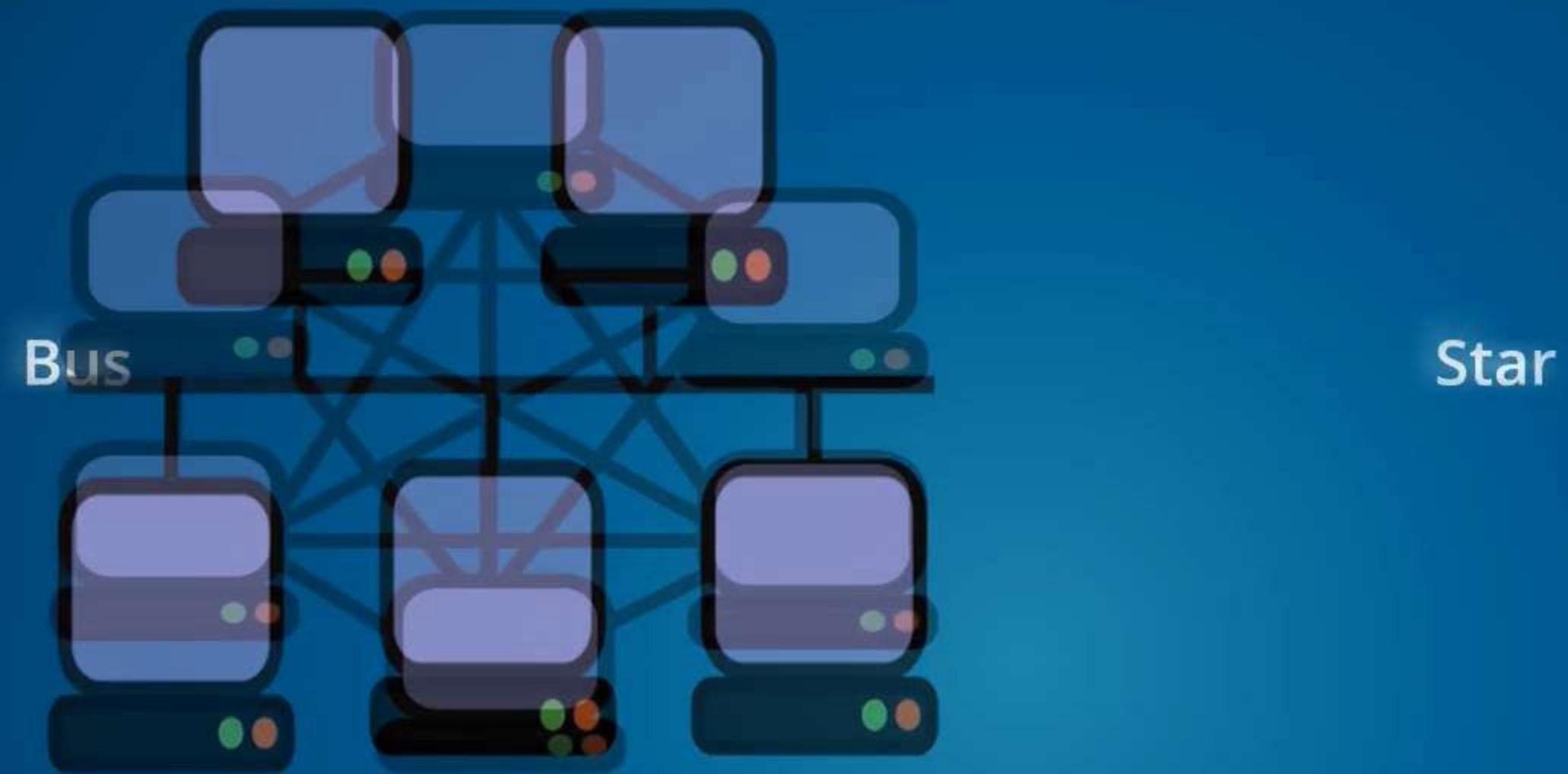
Star

Network
Topologies

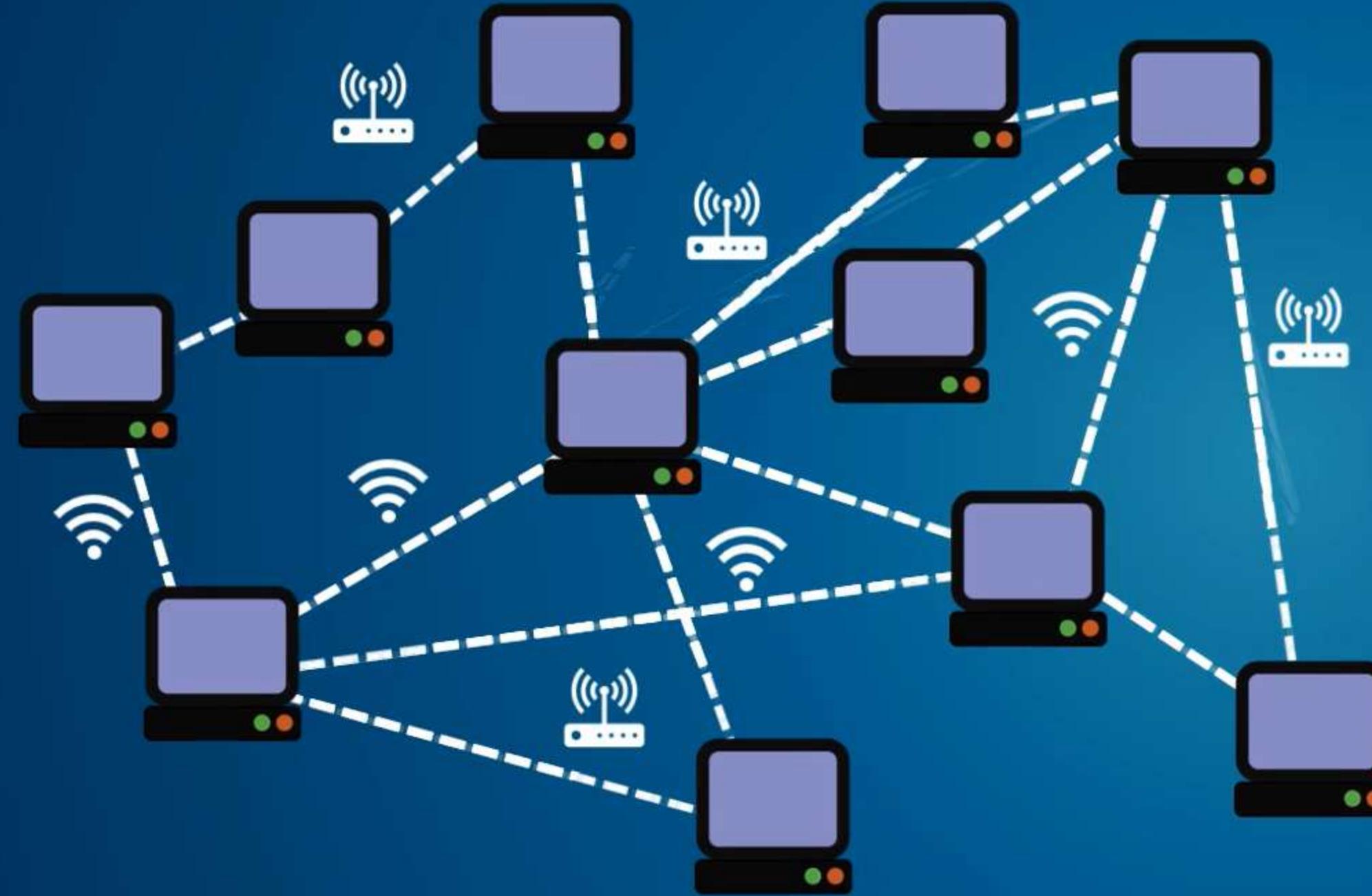


Bus

Early Ethernet
Networks



Modern Ethernet and Wi-Fi Networks



Physical vs Logical

Logical Topology

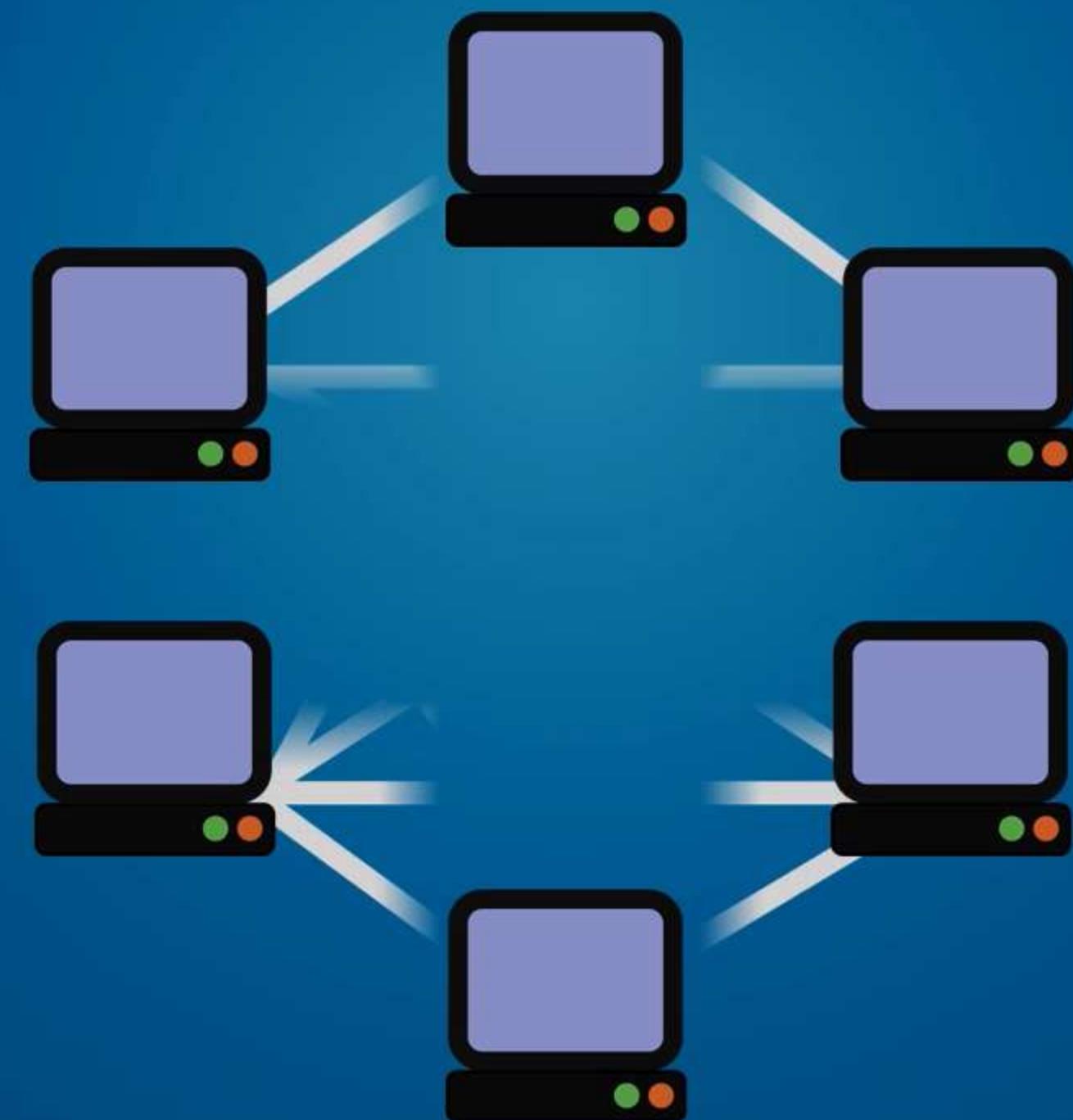
Peer to Peer & Client Server

Logical Topology



Peer to Peer & Client Server

Peer to Peer & Client Server

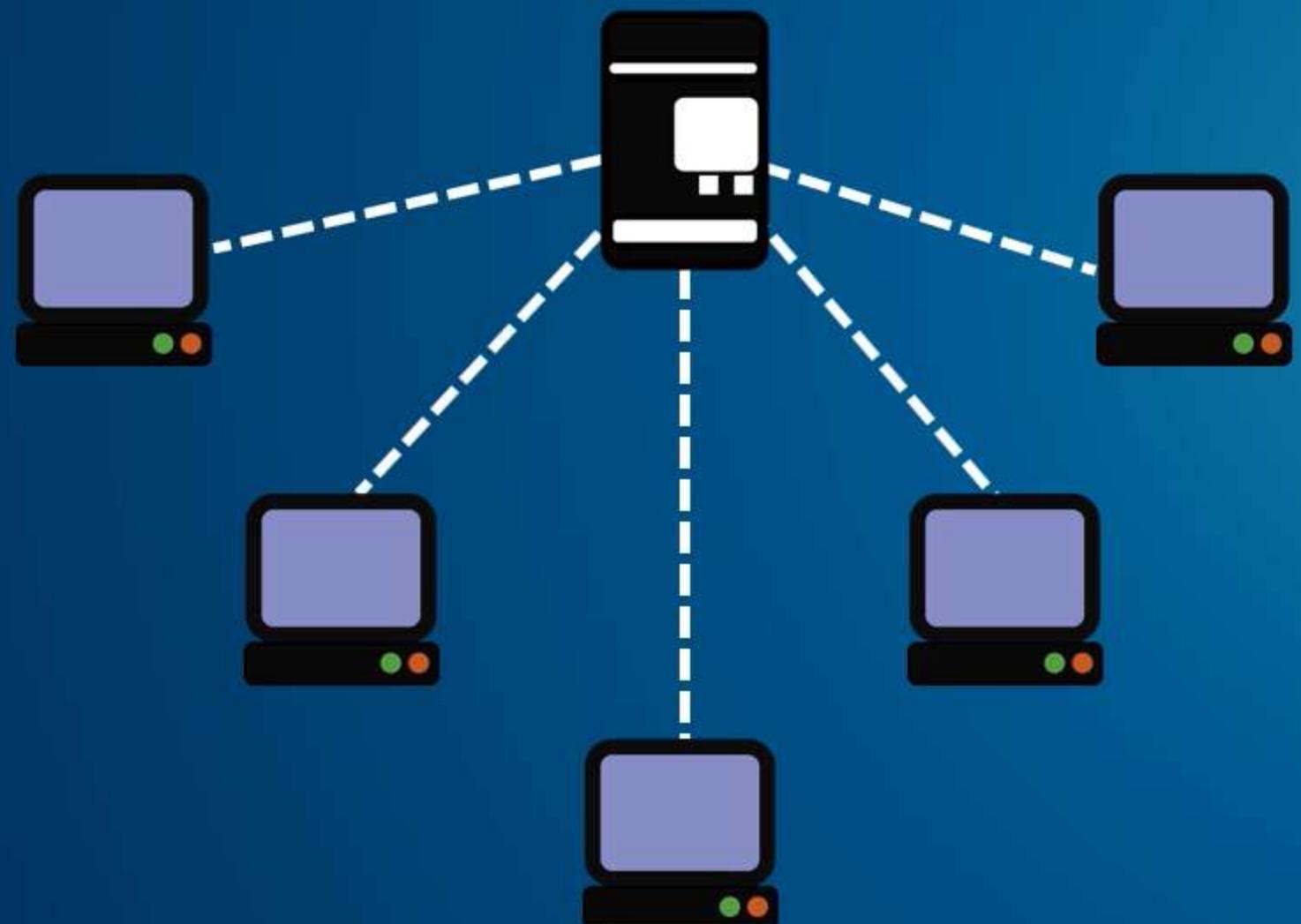


Peer to Peer & Client Server



- ✓ File Server
- ✓ Domain Controller
- ✓ Web Server

Peer to Peer & Client Server



Examples

- ✓ Web
- ✓ Facebook
- ✓ Twitter
- ✓ Google Search

Network Size



MAN

Metropolitan Area Network



Network Size

WAN

Wide Area Network



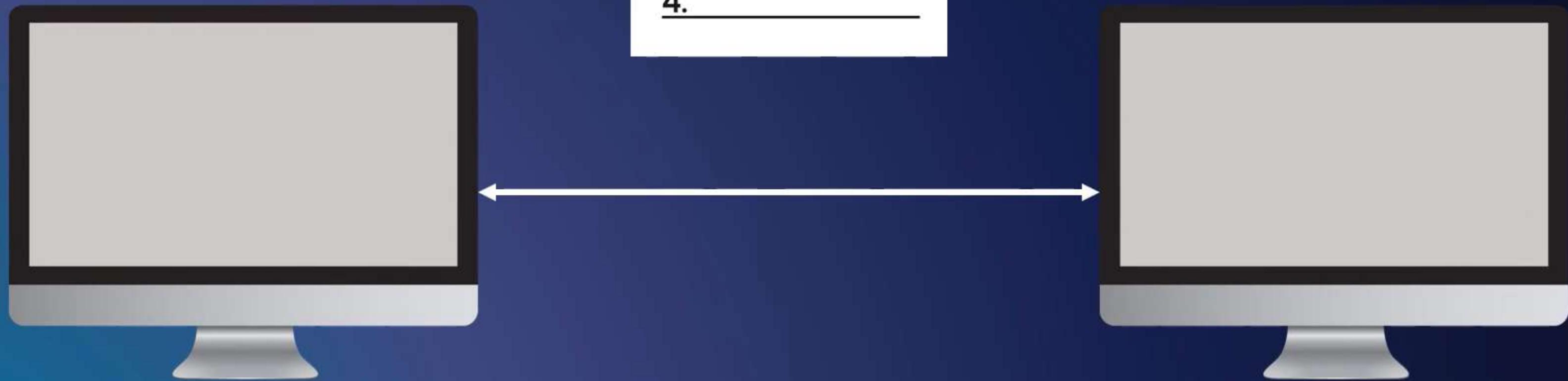
Network Size



Networking
Protocol

RULES

1. _____
2. _____
3. _____
4. _____



Networking Protocol

Wi-Fi



Ethernet



Datalink Protocol

Higher Level Protocols

IP

Physical Level Address

Wi-Fi

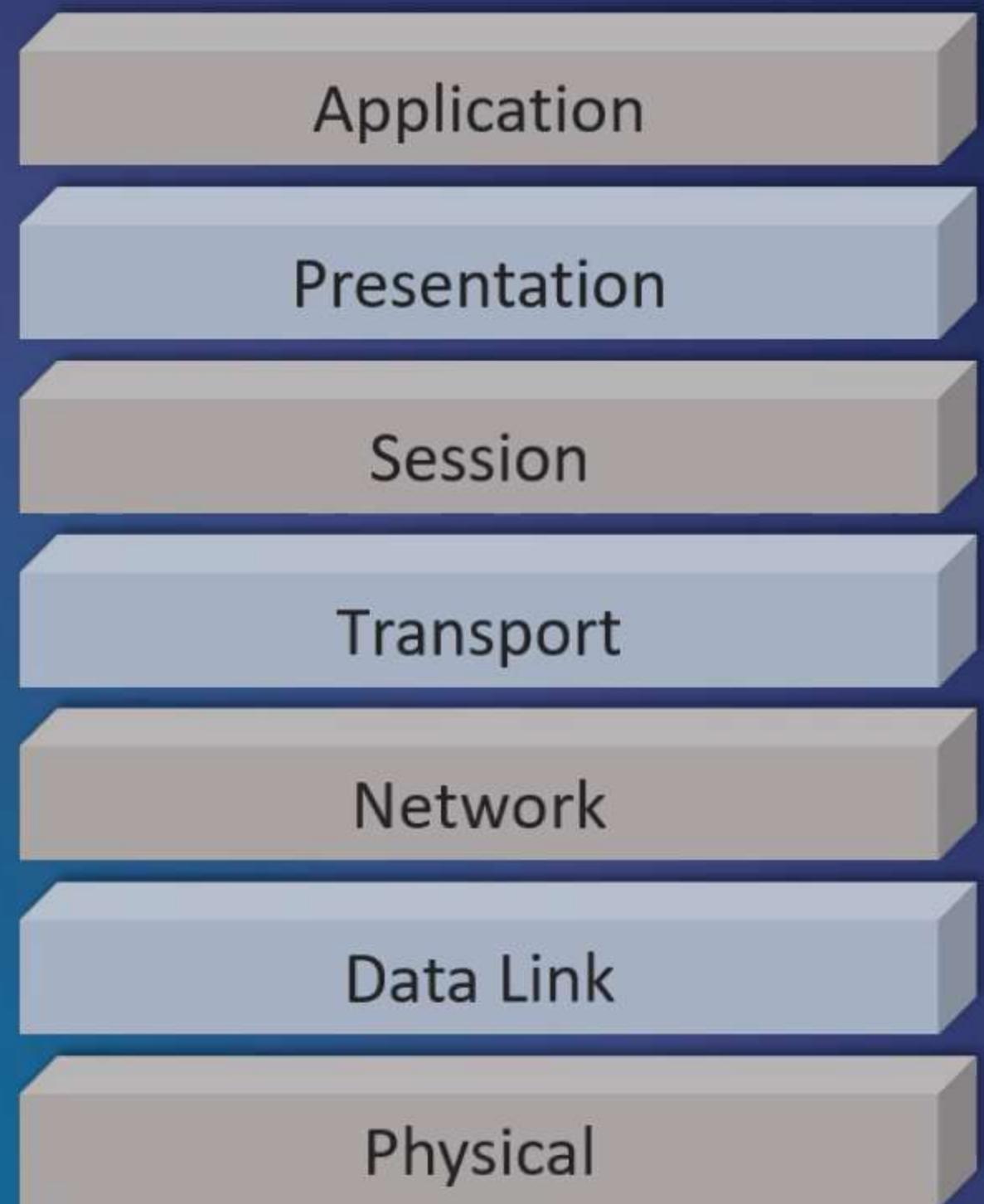


MAC

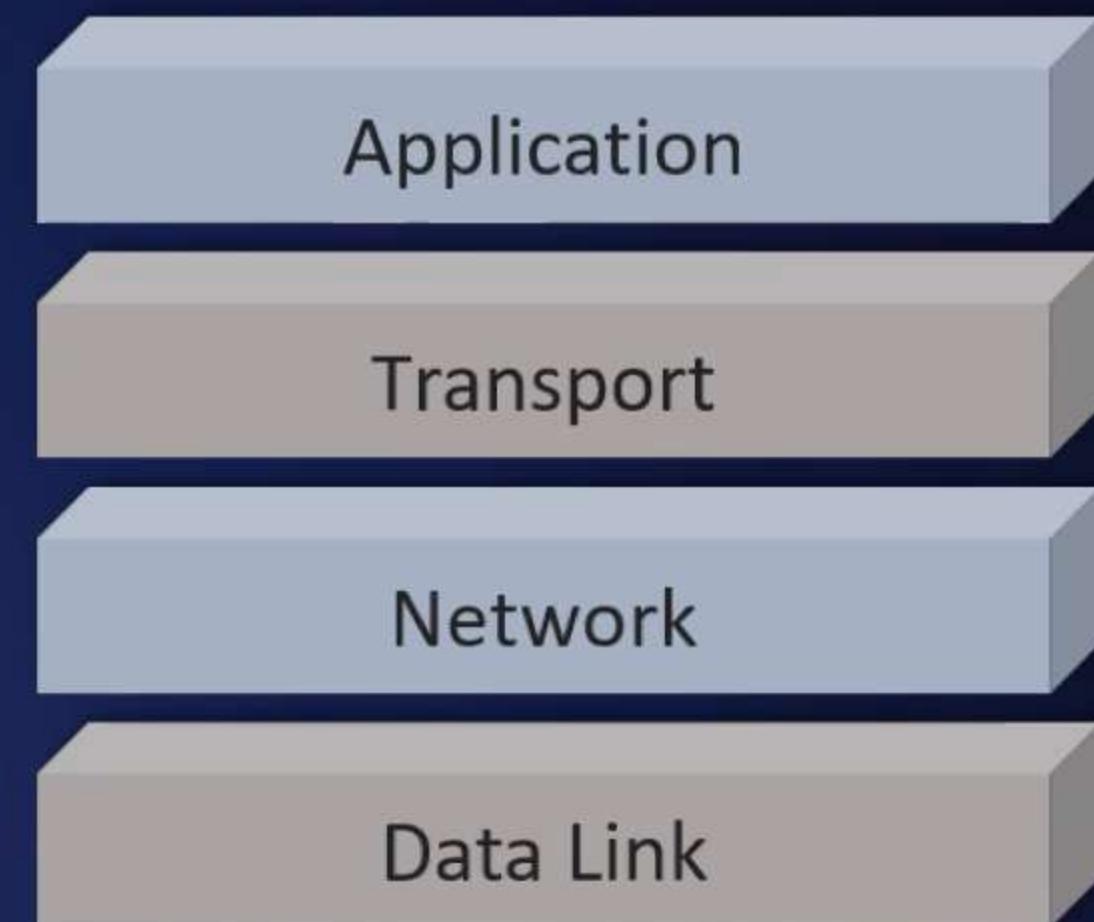
Ethernet



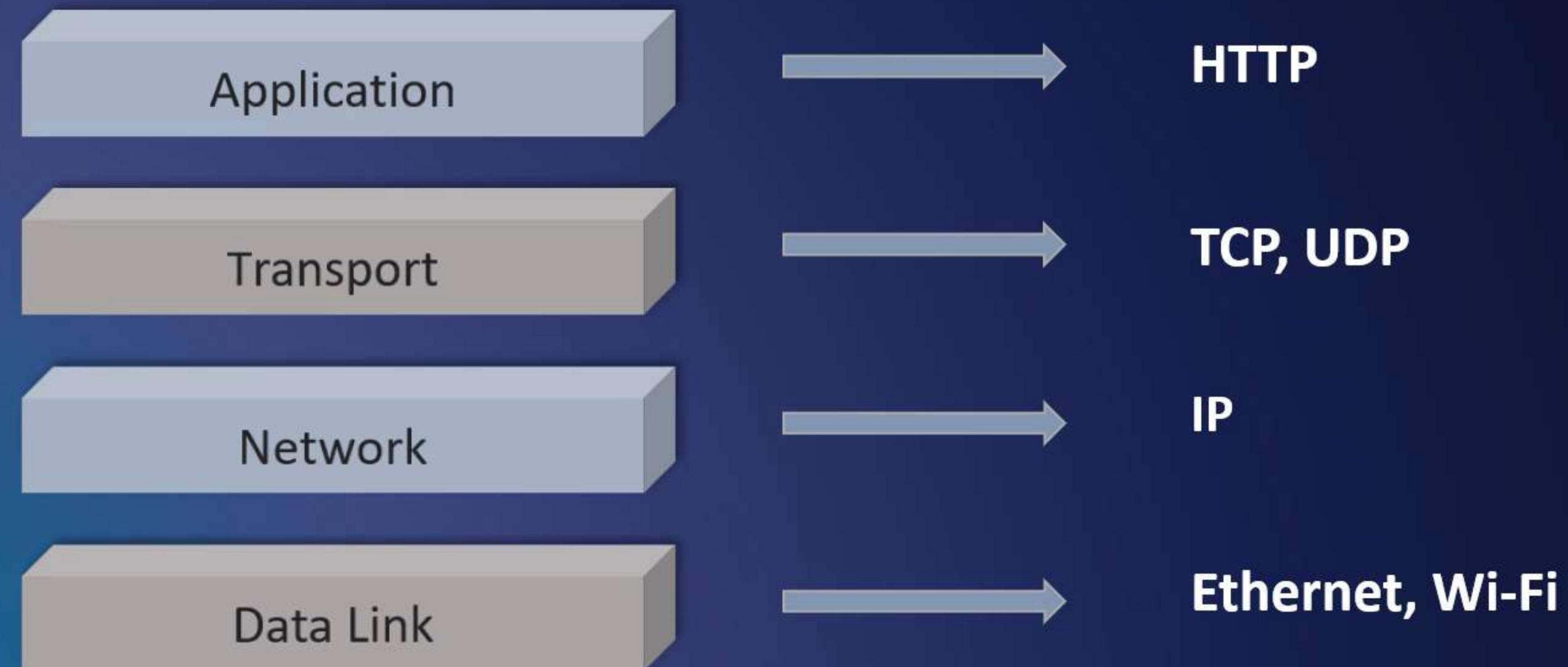
OSI



TCP/IP

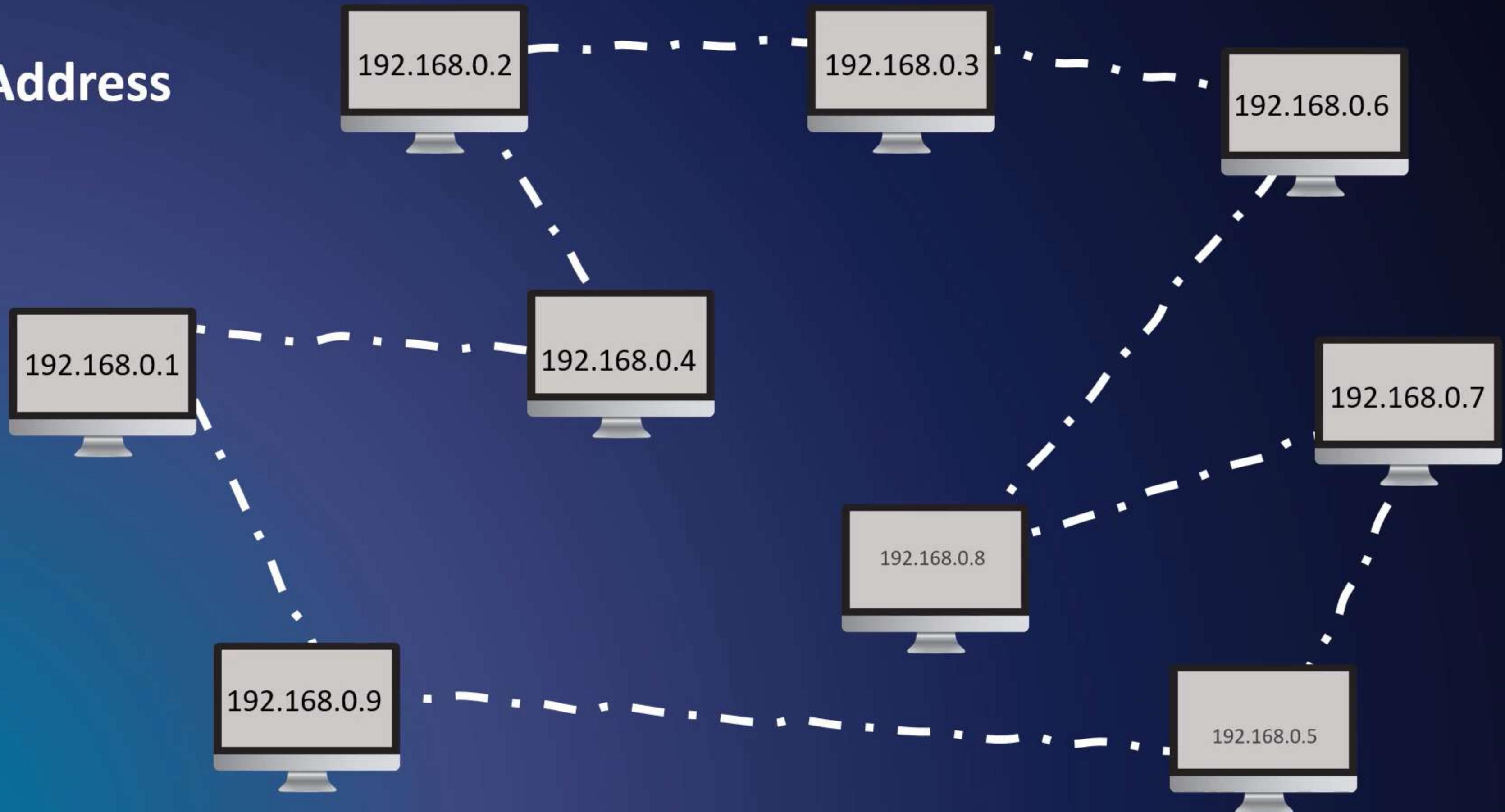


TCP/IP



Network Addressing

IP Address



IP Address

- IPv4
- IPv6



IPv4

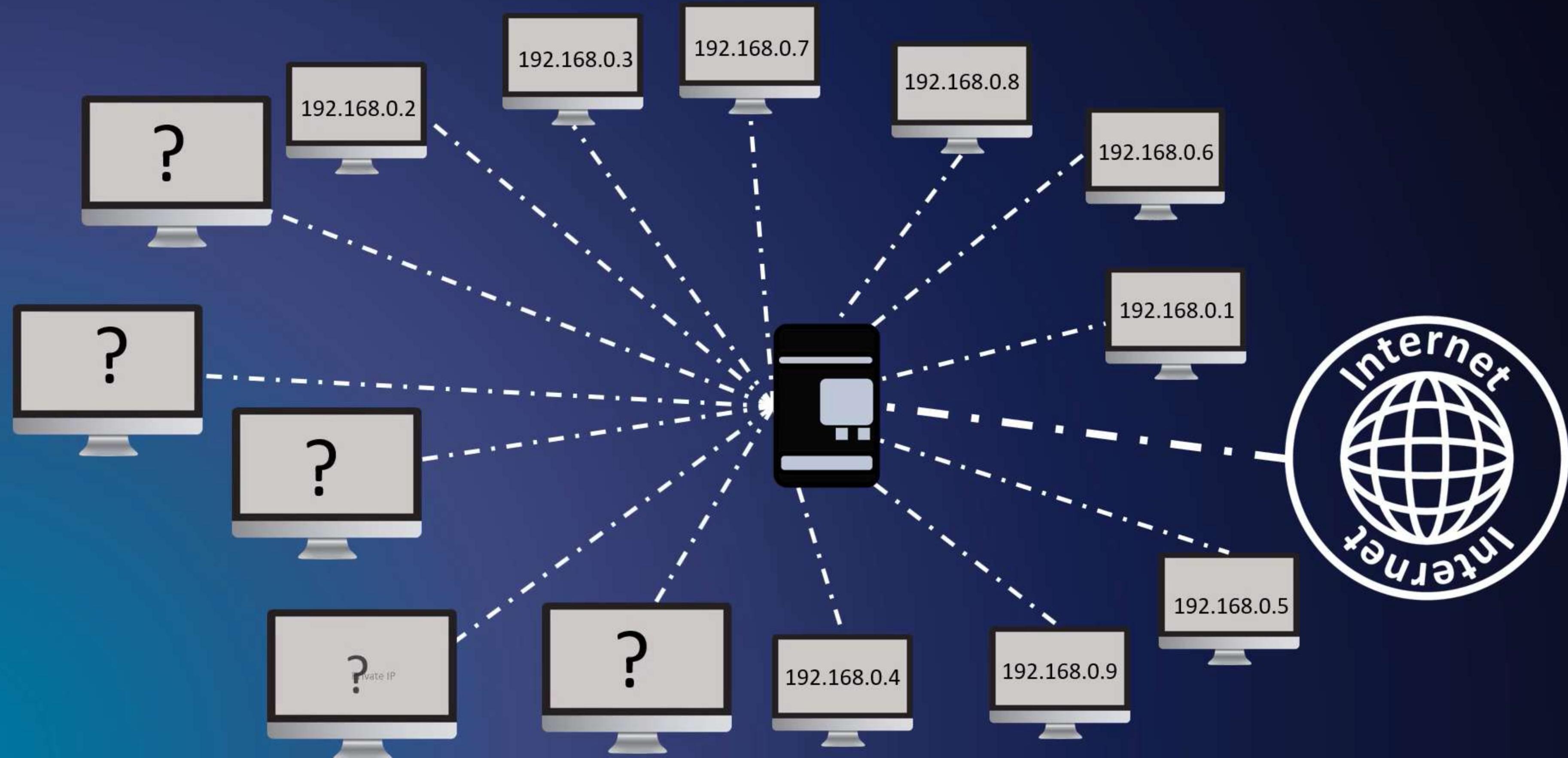




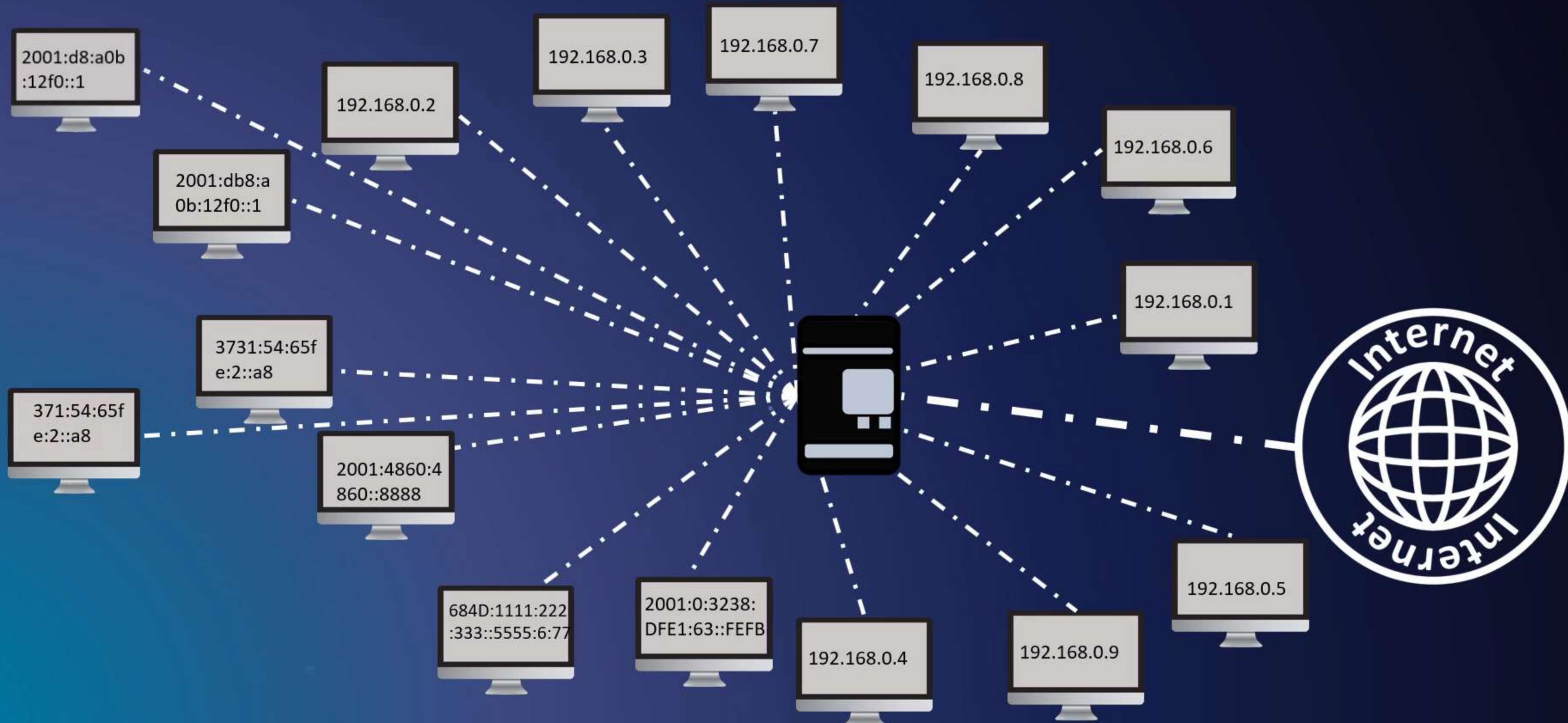
IPv4

32 bits for address

NAT (Network Address Translation)



128 bits for address



Classes of IPv4

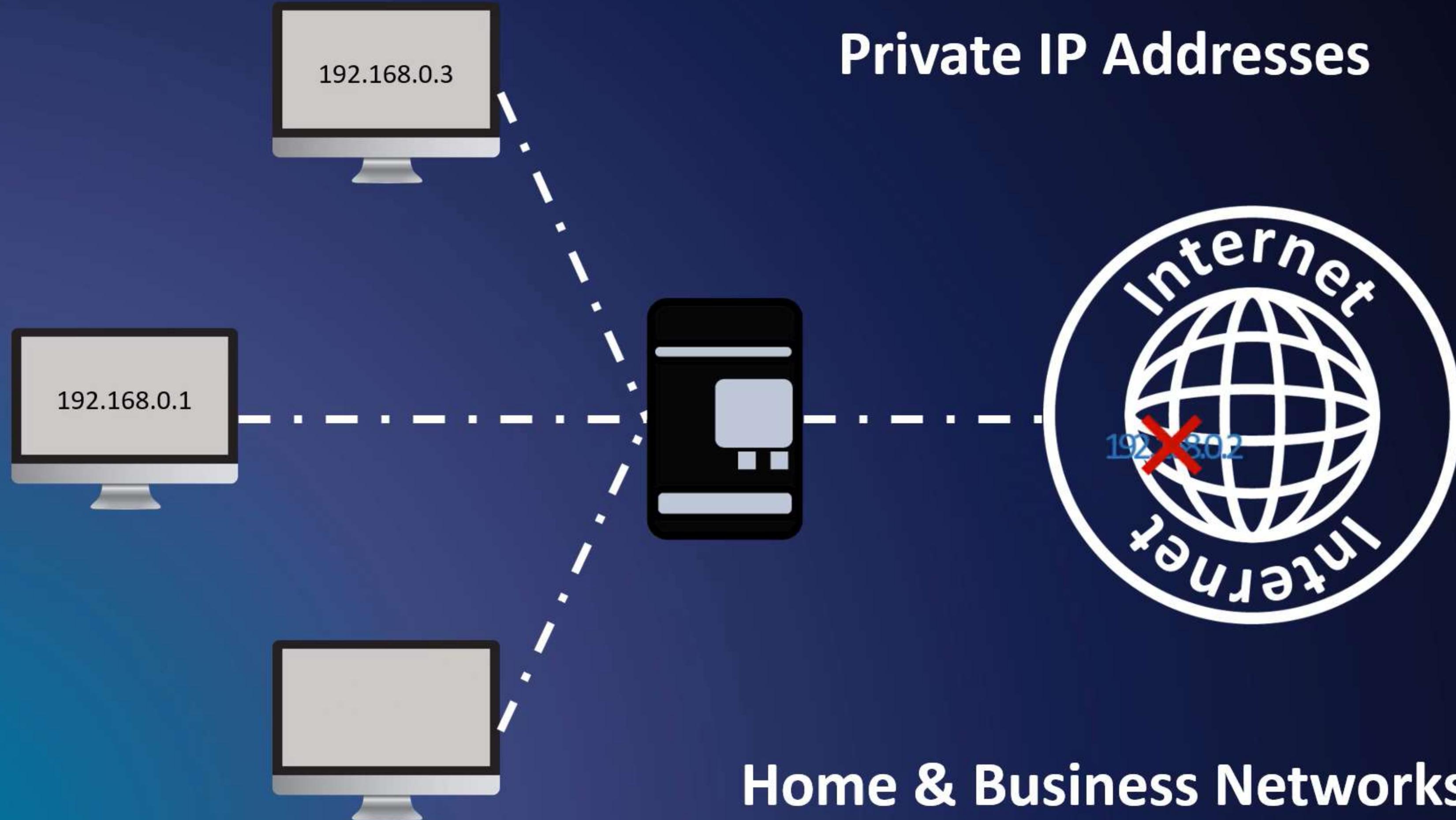
- Class A IP - 0.0.0.0 and 127.255.255.255
- Class B IP - 128.0.0.0 and 191.255.255.255
- Class C IP - 192.0.0.0 and 223.255.255.255
- Class D IP - Multicast

Public & Private IP Addresses

IPv4

IPv6

Private IP Addresses

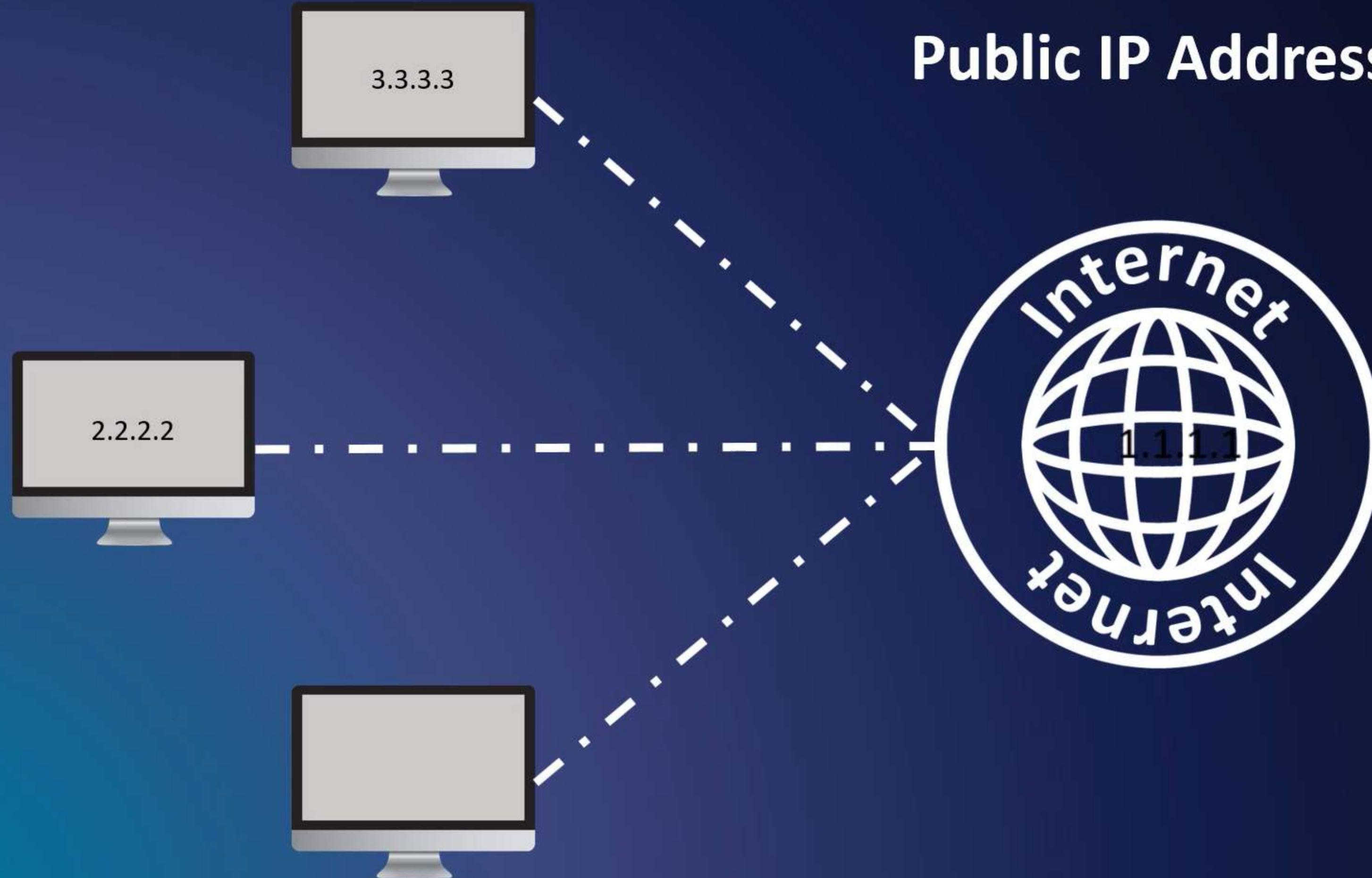


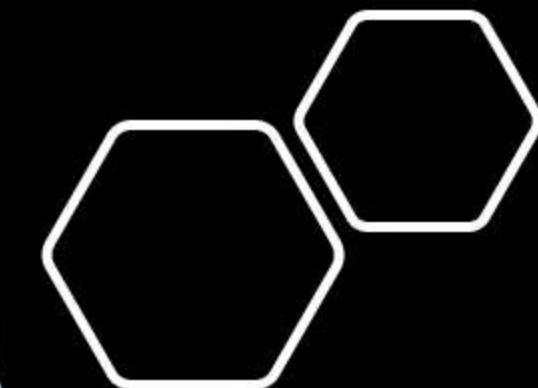
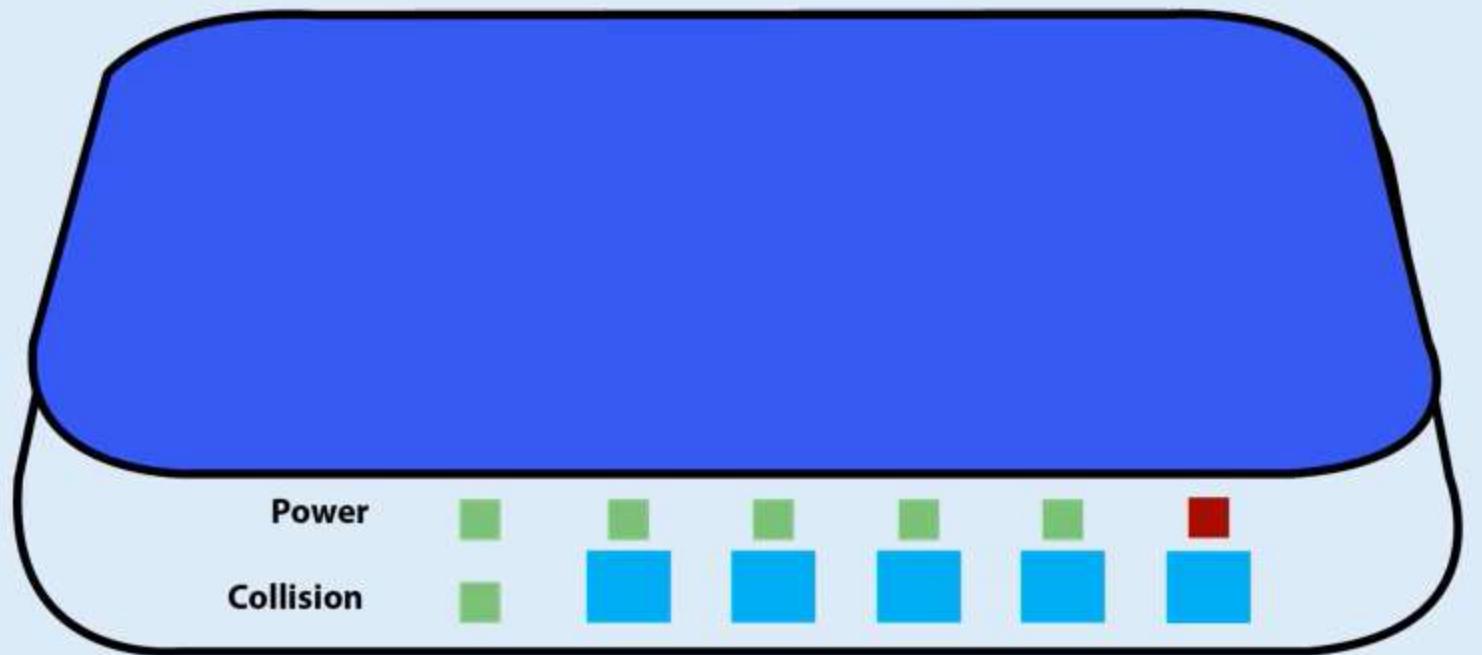
IPv4 Private IP Addresses

10.x.x.x

192.168.x.x

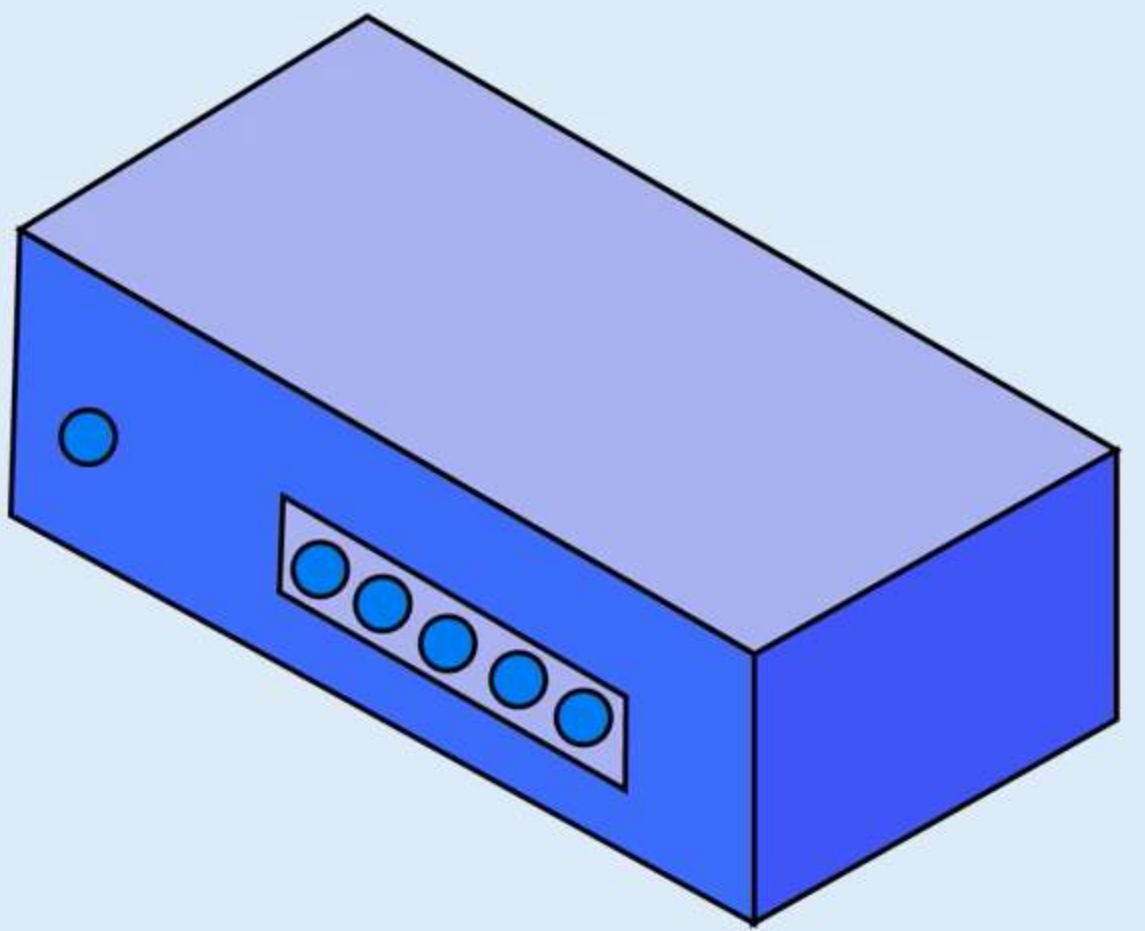
Public IP Addresses



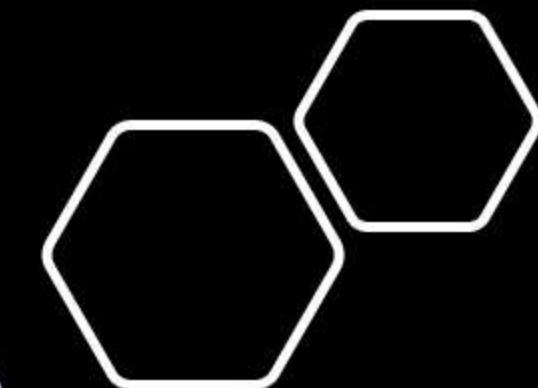


A hub is one of the most basic networking devices that
connects two or more computers together

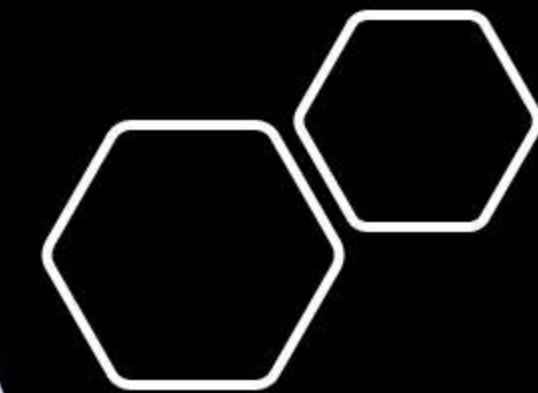
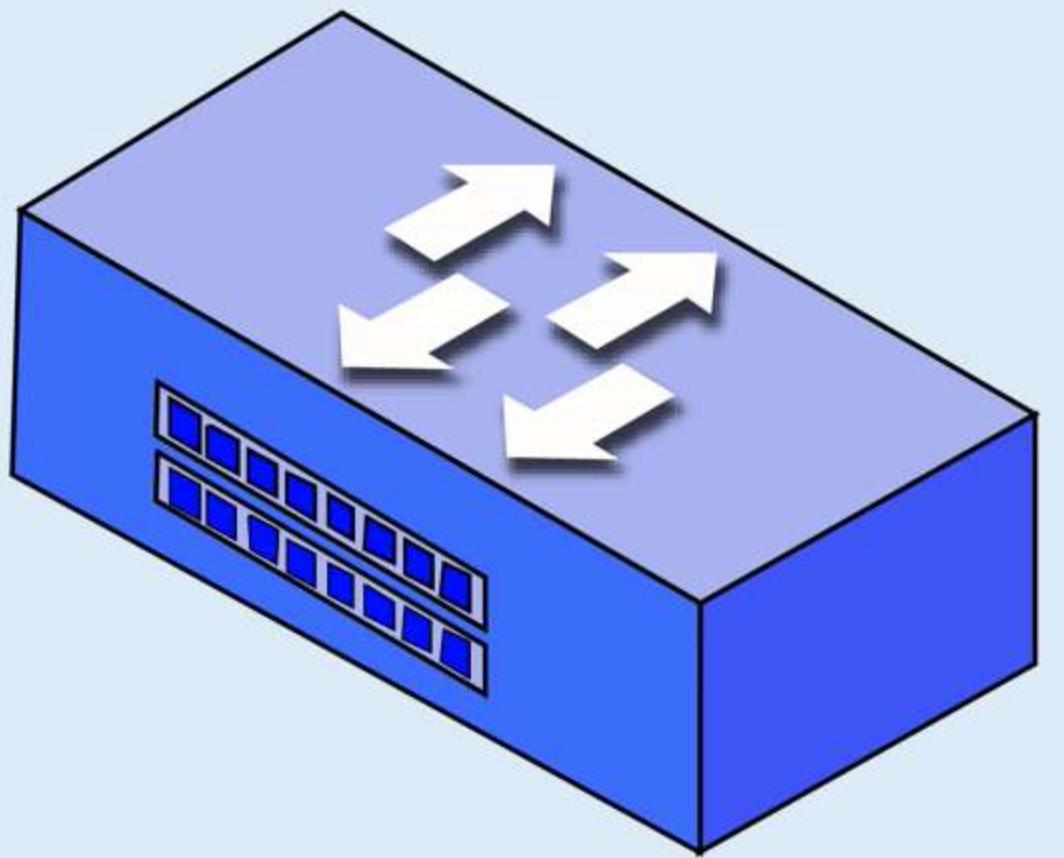
Network Hub



A bridge connects two network segments together and is a
selective repeater

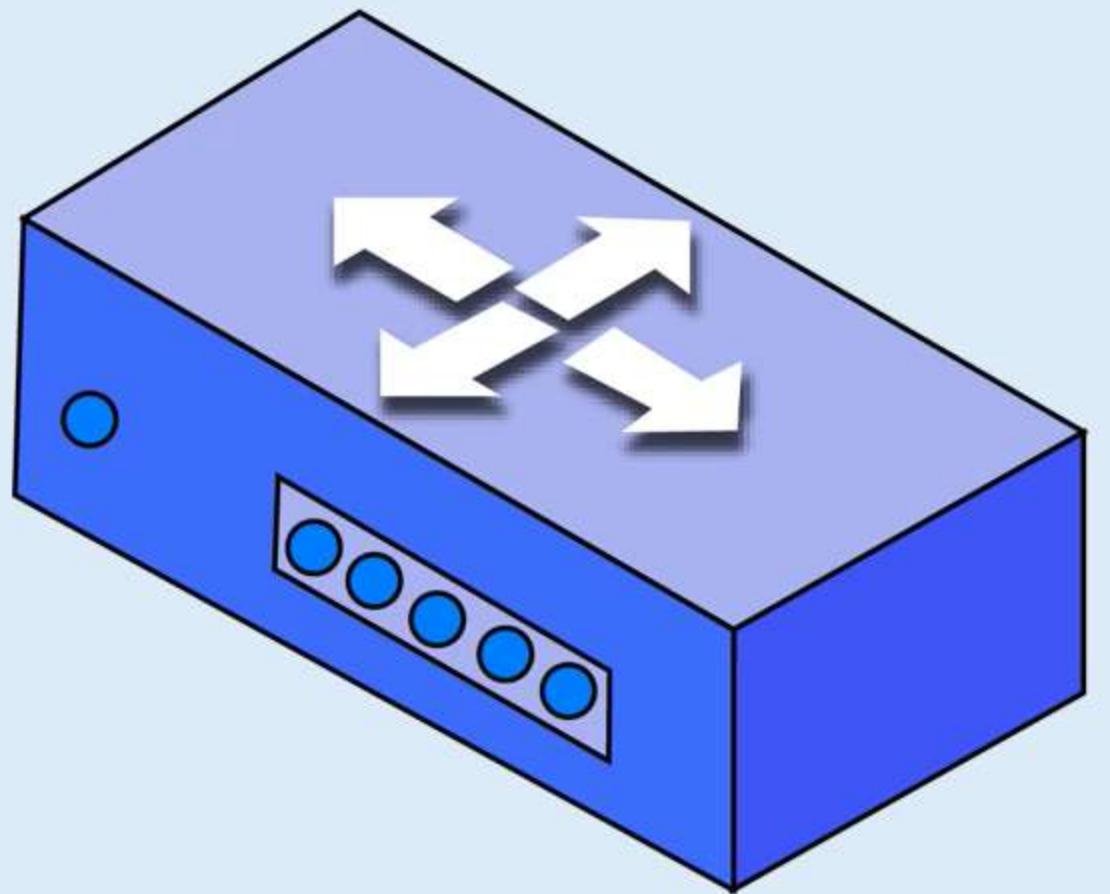


Network Bridge



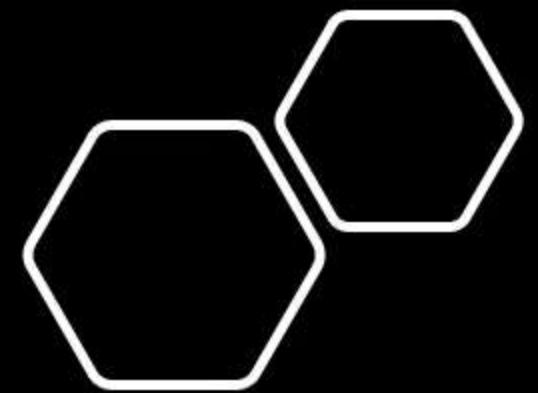
A switch bonds two or more computers together and is
widely used today in preference to a hub or bridge

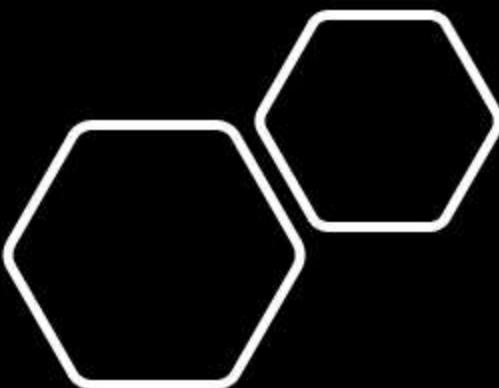
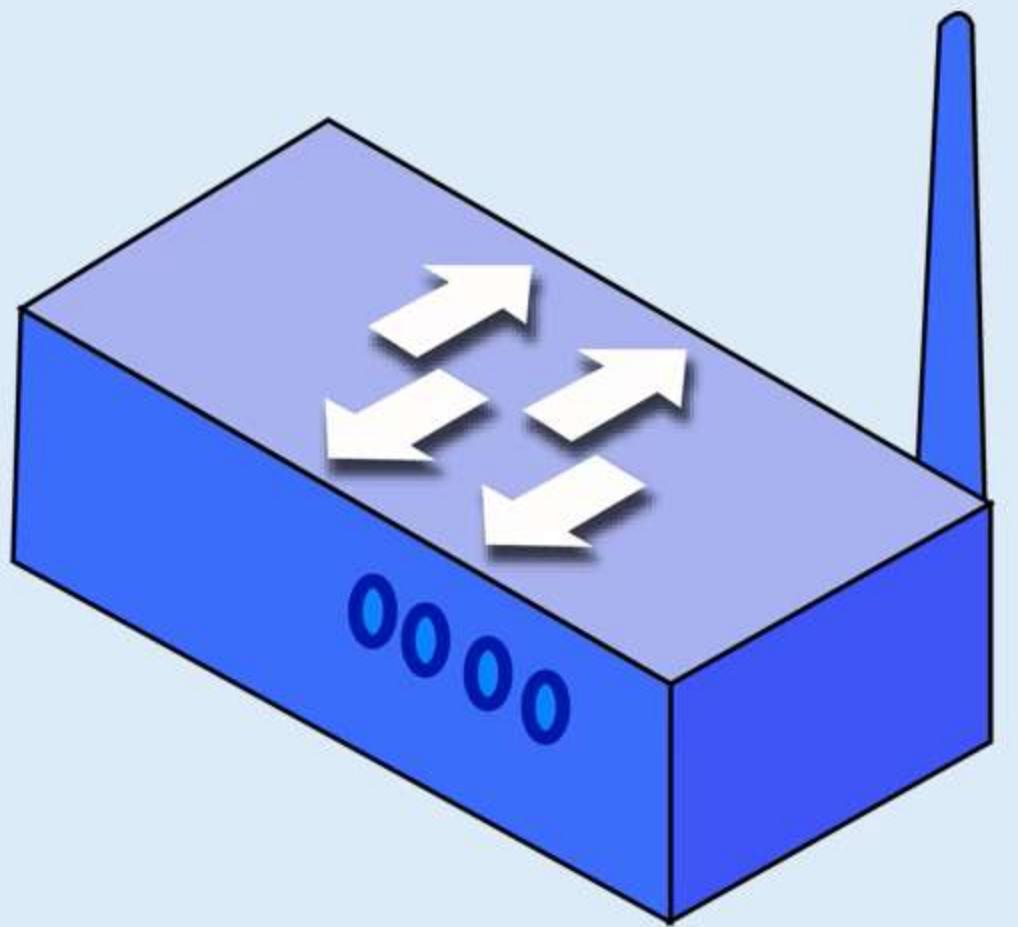
Network Switch



A router connects networks together

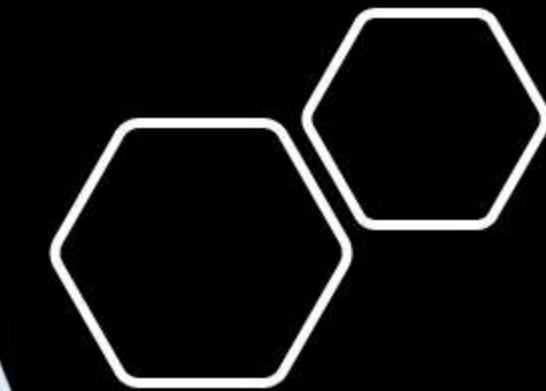
Router





A wireless access point connects wireless devices to an
Ethernet network, and to each other

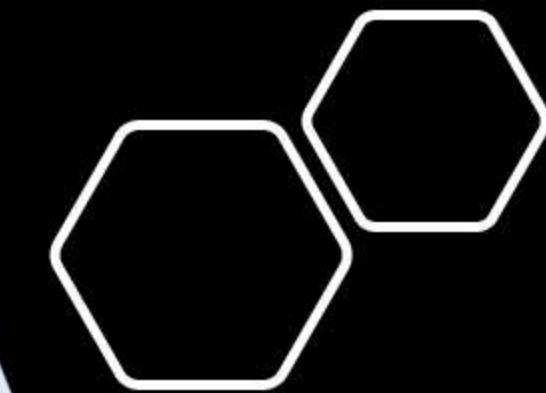
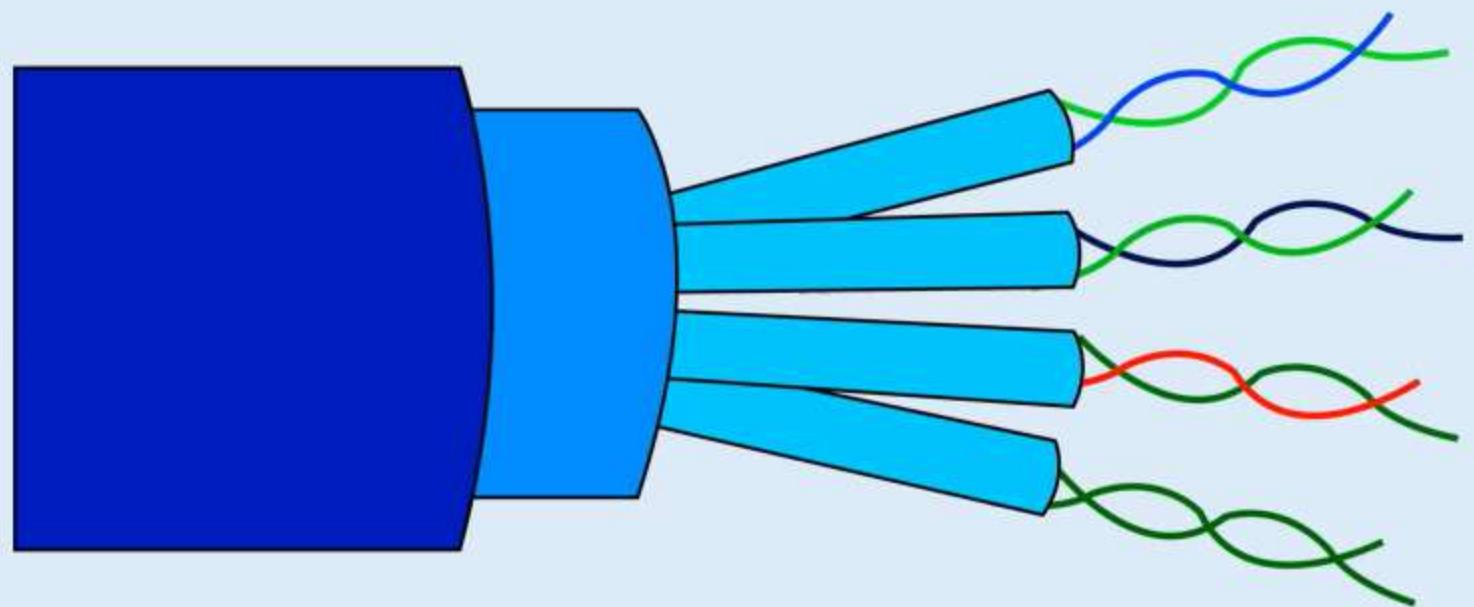
Wireless
Access Point



A firewall functions like a router.

**Apart from that, it also blocks traffic from the external
network according to user configured rules**

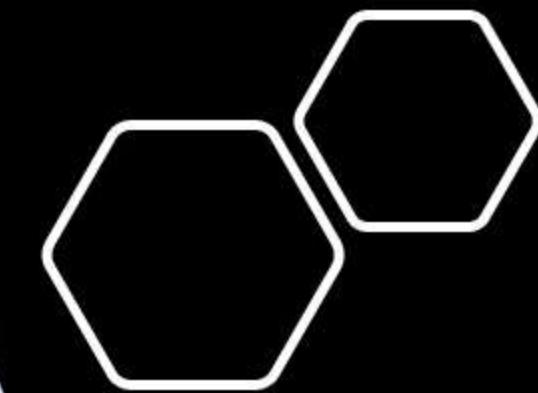
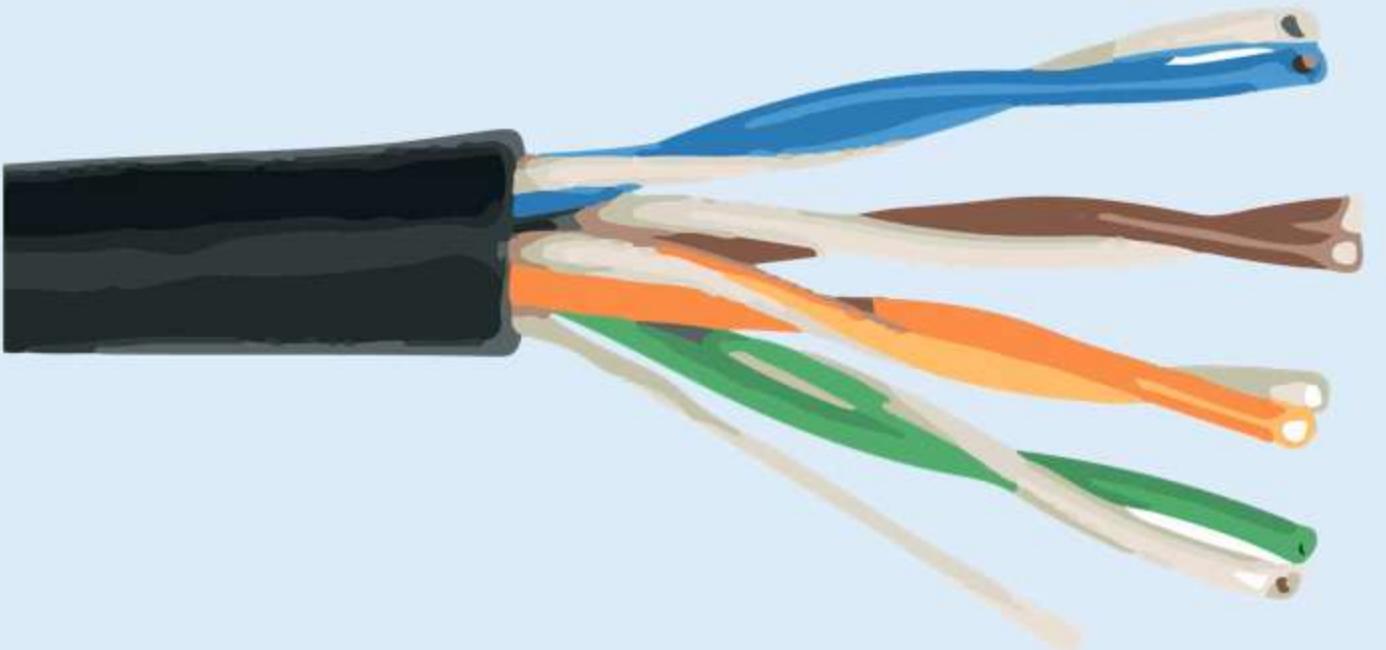
**Firewall
&
NAT Router**



**STP cables come as a twisted wire pair, shielded in
metal foil.**

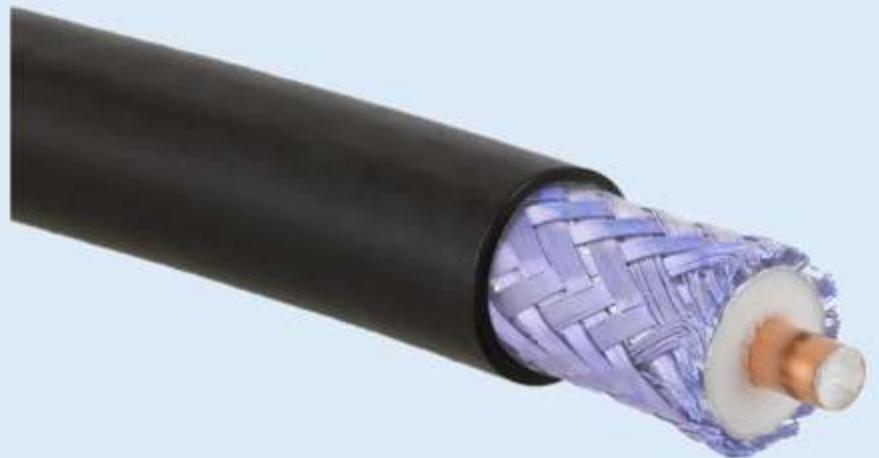
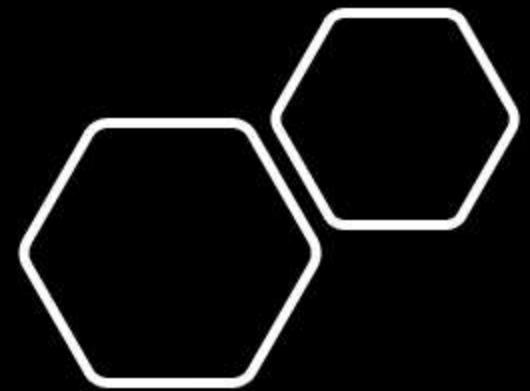
**This helps in blocking noise interference and reducing
crosstalk**

STP cables



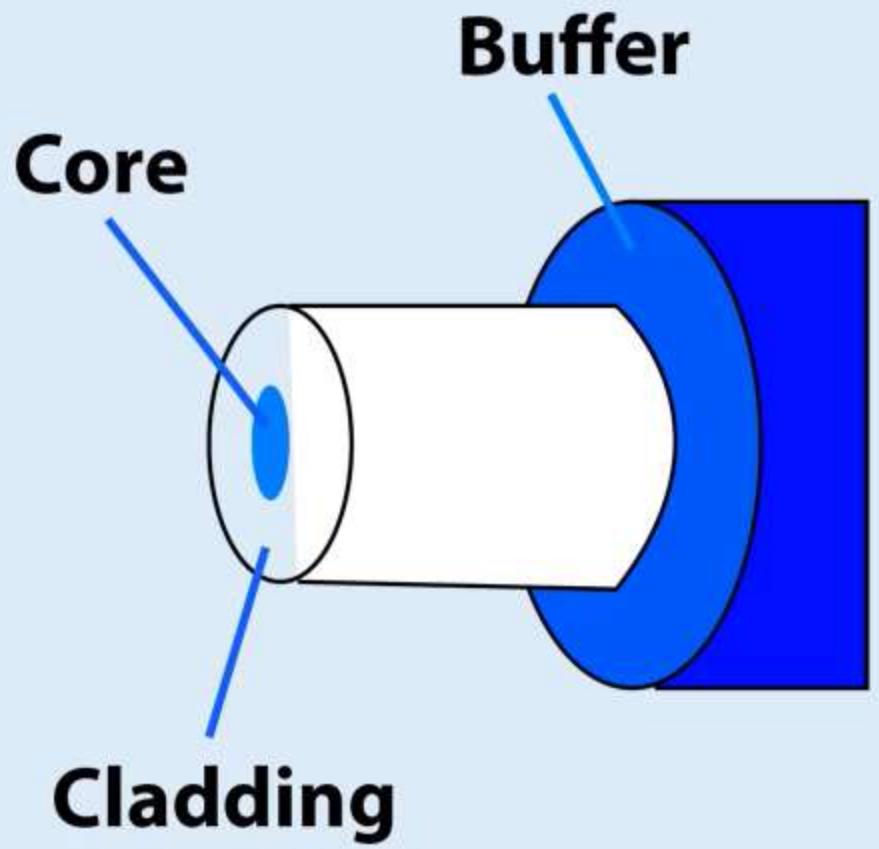
**UTP cables have seven categories, each of them suitable
for a precise use**

UTP cables



Coaxial Cable generally called **coax**, are copper cables with metal shielding intended to provide immunity against noise and greater bandwidth

Coaxial Cable



Optical Fiber the signal is transmitted by use of light

through the glass fiber

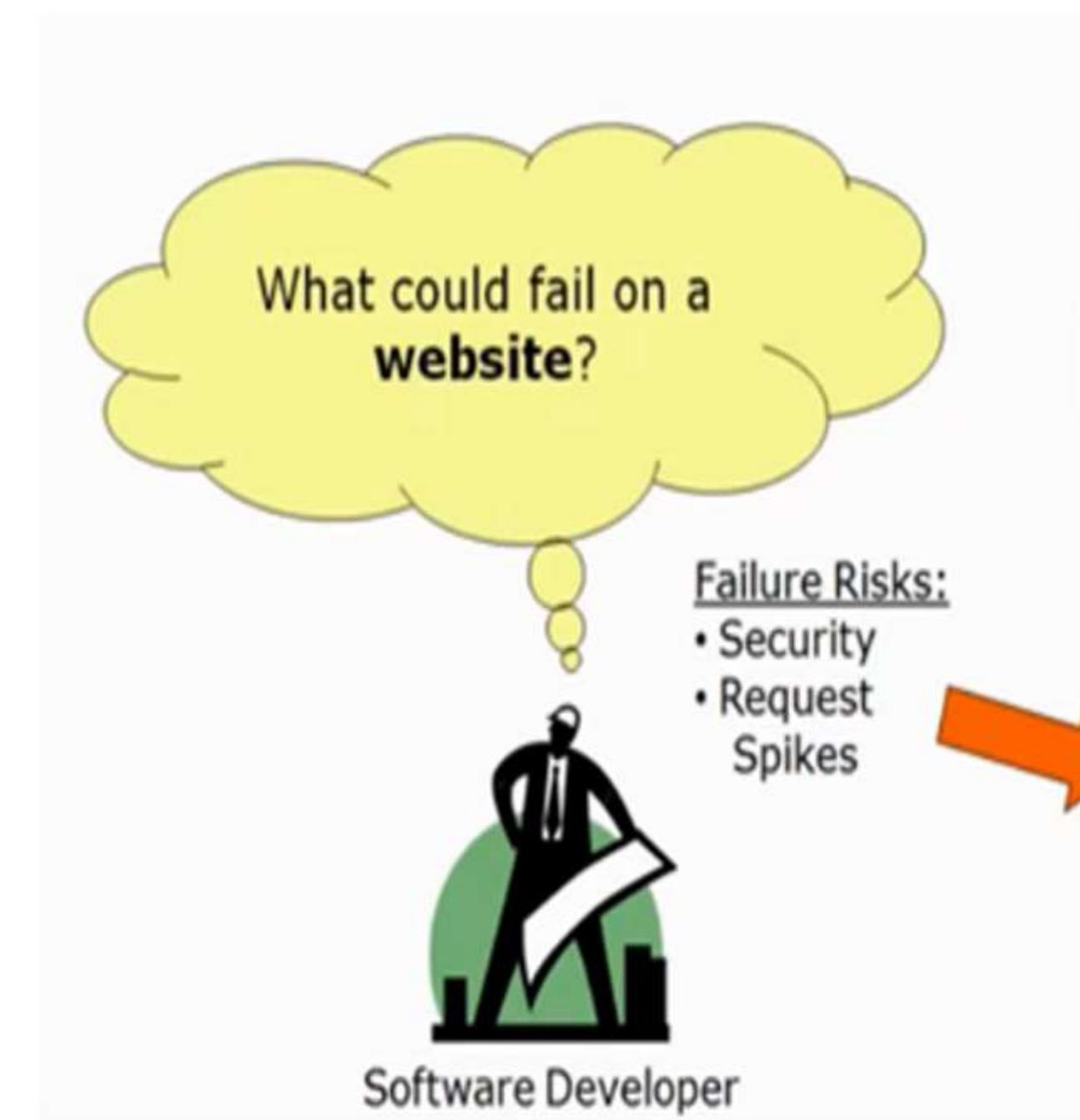
Optical Fiber



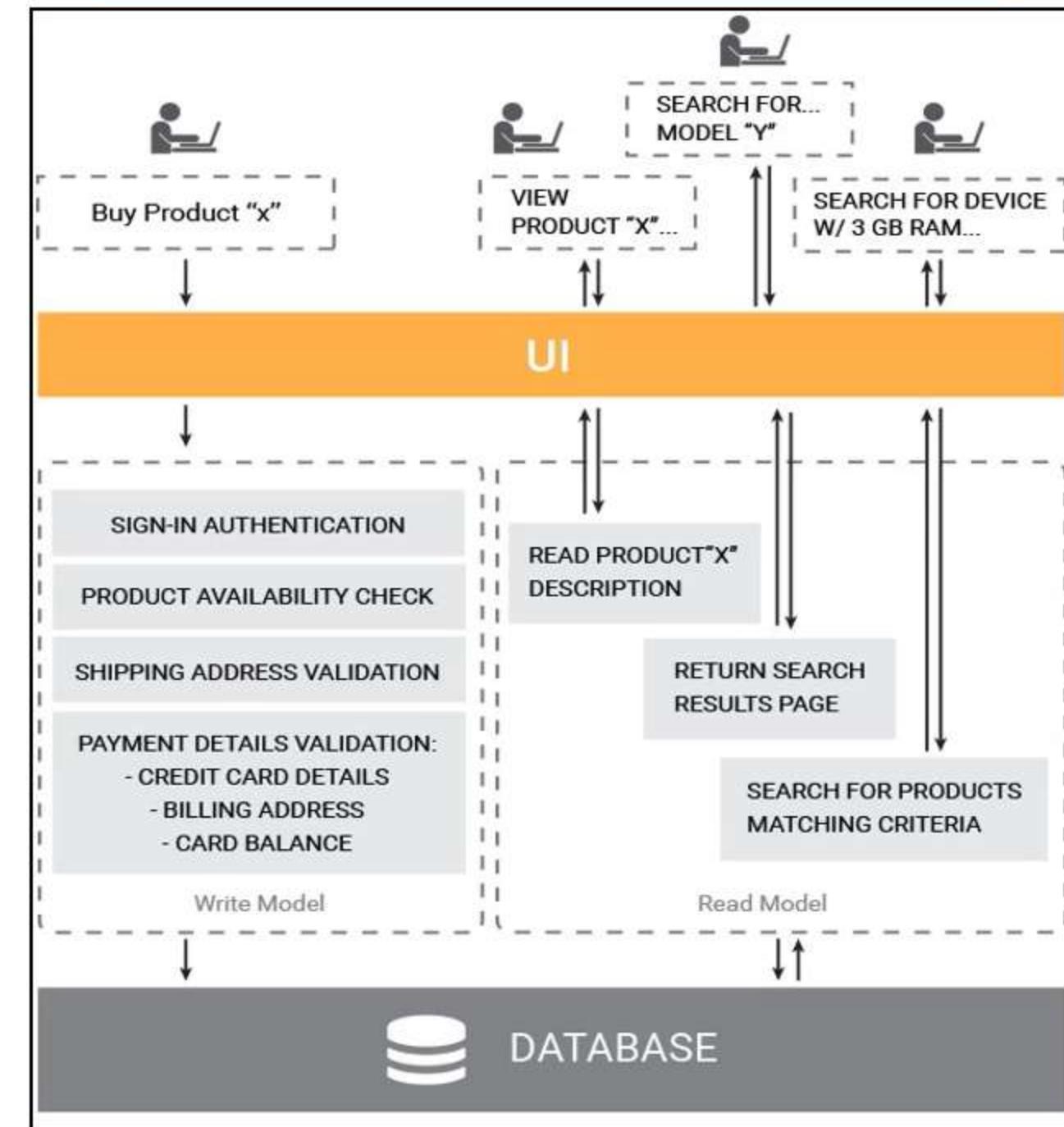
N-TIER ARCHITECTURE



Overview



Software Architectural Pattern
simplifies the problem



In this module you will learn

- Tier Architecture introduction
- Presentation Layer
- Services Layer
- Data Layer and implementation
- Integration Layers



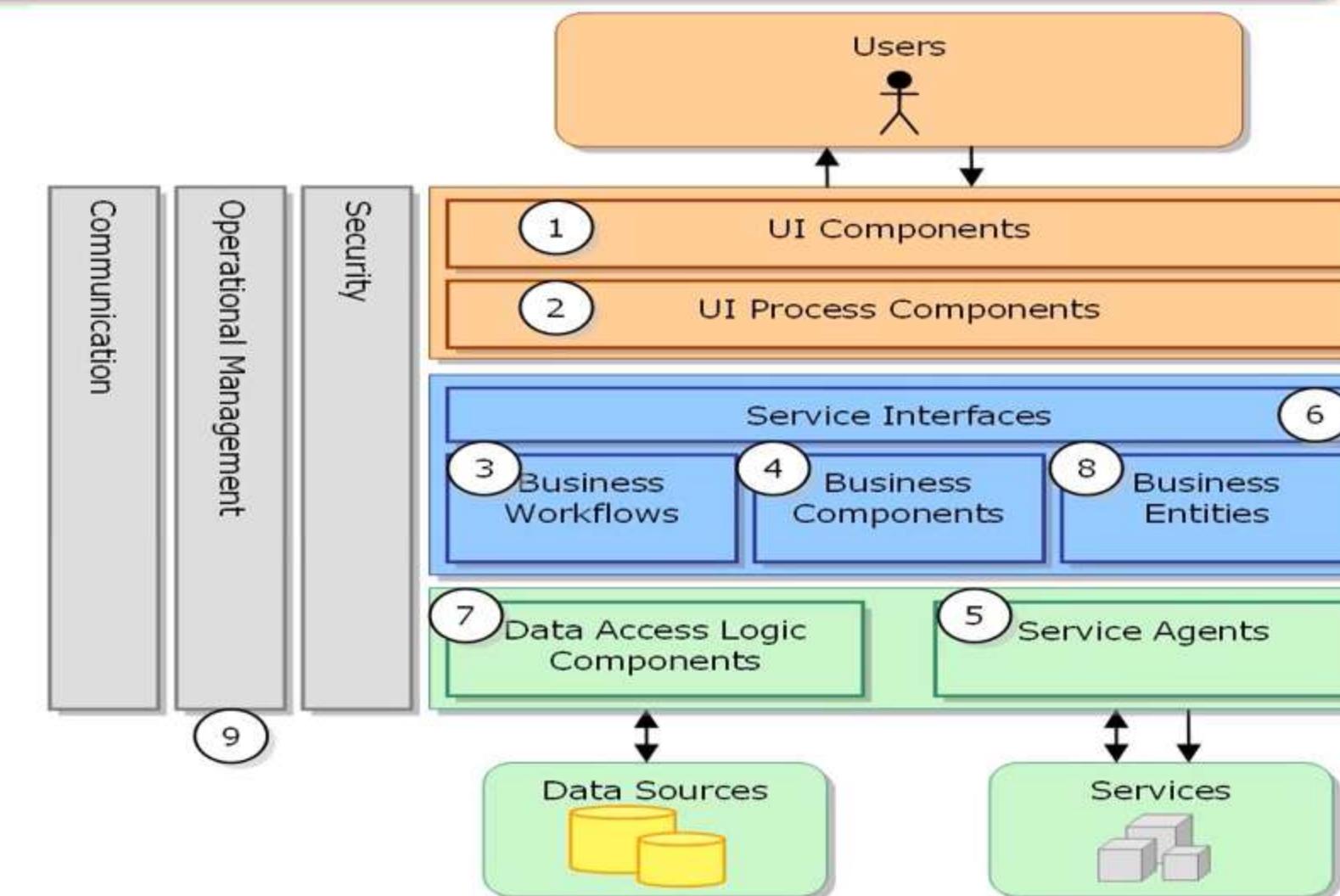
Tier Architecture introduction

What is Software Architecture?

It depicts the system's organization or structure, and provides an explanation of how it behaves

The high level breakdown of a system into parts (structures).
 Each structure/parts comprises software elements, relationships among them, and its properties

Logical or physical grouping of components into separate **layers** or **tiers** that communicate with each other and with other clients and applications.



Layers

Layers

- *Layer* is a logical structuring mechanism for software elements
- Layers are concerned with the logical division of software components and functionalities
- Used for software development purpose
- Helps to maximize maintainability of the code



Types of layers:

Three major layers involved in an application:

Presentation Layer

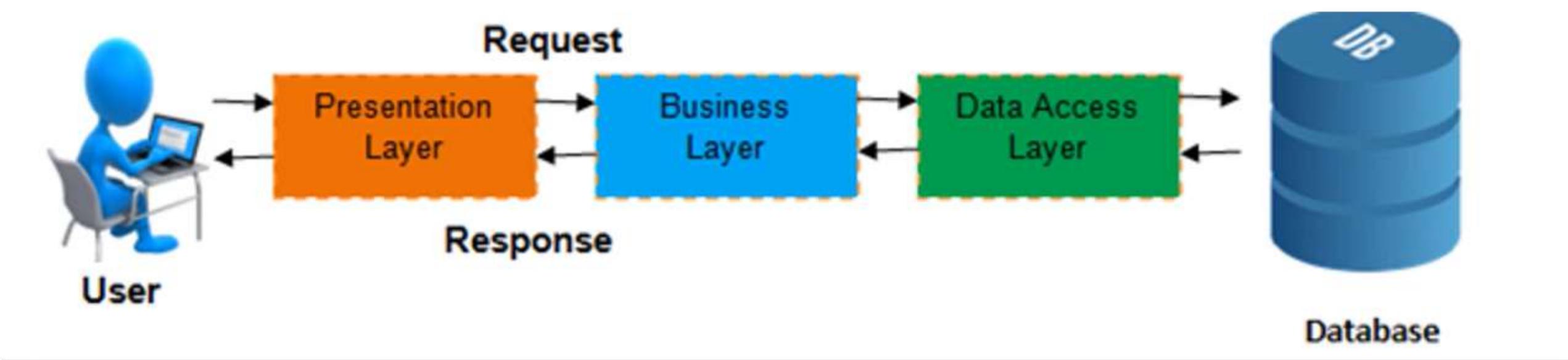
UI, Web Page or Mobile App

business/Application Layer

Business logic and data validation

Data Layer

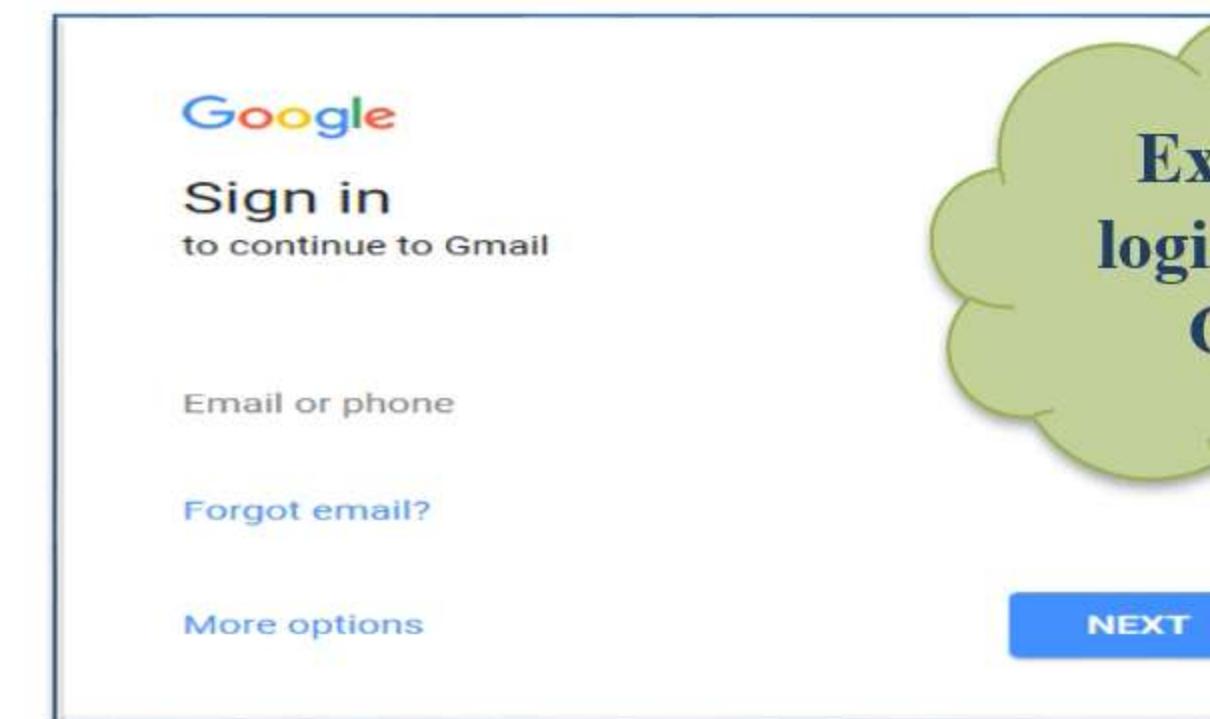
External data sources like SQL server, oracle, etc.



Presentation Layer

- *Top most layer of an application.*
- *It is also known as Client layer*
- *This is the layer we see when we use a software*
- *Using this layer users can access a web page, or an operating systems GUI*
- *The main functionality of this layer is to communicate with Application layer*

**In simple words:
This layer is to view
the application**



Presentation Layer Components



Presentation layer components implement the functionality required to allow users to interact with the application

User interface (UI) components

- Provide the mechanism for users to interact with the application
- Format data and render it for display, acquire and validate data entered by users

UI process components

- Help to synchronize and order user interactions
- This prevents the process flow and state management logic from being hard-coded into the UI elements themselves

Business Layer

- *It is also known as Application layer or middle layer*
- *This layer implements the core functionality of the system, and encapsulate the relevant business logic*
- *It controls an application's functionality by performing detailed processing*
- *This layer acts as a mediator between the Presentation and the Data layer*

In simple words:
This layer is to perform operations on the application.

In Gmail login page:
Once user clicks on the login button, this layer takes the information and interacts with Data layer and sends required information to the Presentation layer

Business Layer Components



These components implement the core functionality of the system, and encapsulate the relevant business logic

Business components

- Implement the business logic of the application

Business workflows.

- These components define and coordinate long-running, multi-step business processes, and can be implemented using business process management tools

Business entity components

- These components are used to pass data between components

Data Layer

- *This Layer consists of database servers and other resources where information is stored and retrieved*
- *It provides access to data (back-end) that is hosted within the boundaries of the system, and data exposed by other back-end systems*
- *This layer keeps data neutral and independent from application servers or business logic*
- *Application layer communicates with Data layer to retrieve the data.*

In simple words:
This layer is to share and retrieve the data

In Gmail login page:
Once data layer retrieves necessary login information from the database and returns back to the Application layer

Data Layer Components

Data layer components implement the logic to access data regardless of the storage mechanism

Data access components

- These components abstract the logic required to access the underlying data stores

Data helper and utility components

- This helps to reduce the complexity of the data access components and centralizes the logic, which simplifies maintenance

Service agents

- Service agents implement data access components that isolate the varying requirements for calling services from your application

Service Layer

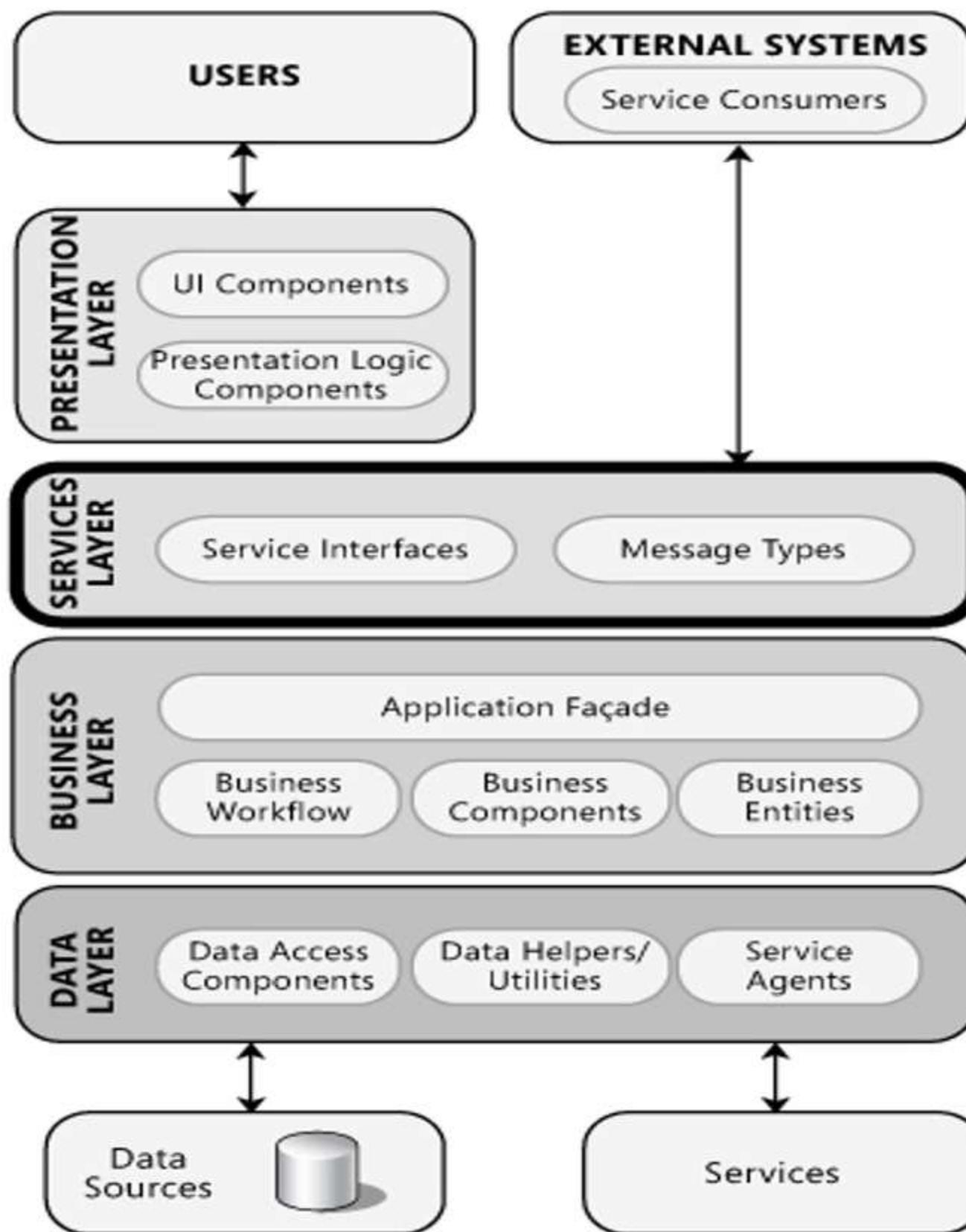


- This layer provides other clients and applications with a way to access business logic in the application
- It defines and implements the service interface and the service contracts (or message types)
- Service layer should never expose details of the internal processes or the business entities used within the application

Benefit:

It defines a common set of application operations available to many kinds of clients and it coordinates an application's response in each operation

Service Layer Components



Services layer components provide other clients and applications with a way to access business logic in the application. They are:

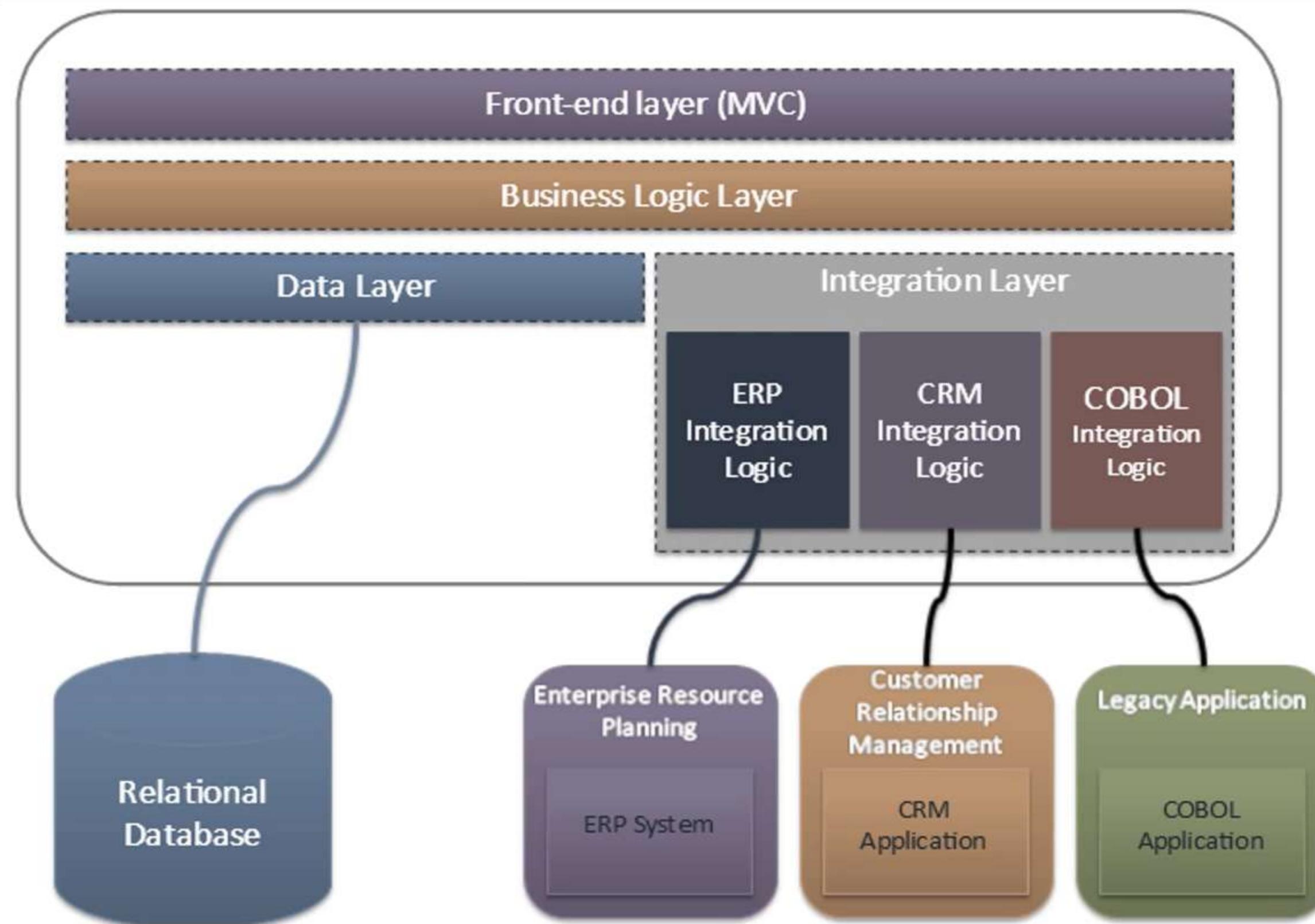
1. Service interfaces

2. Message types

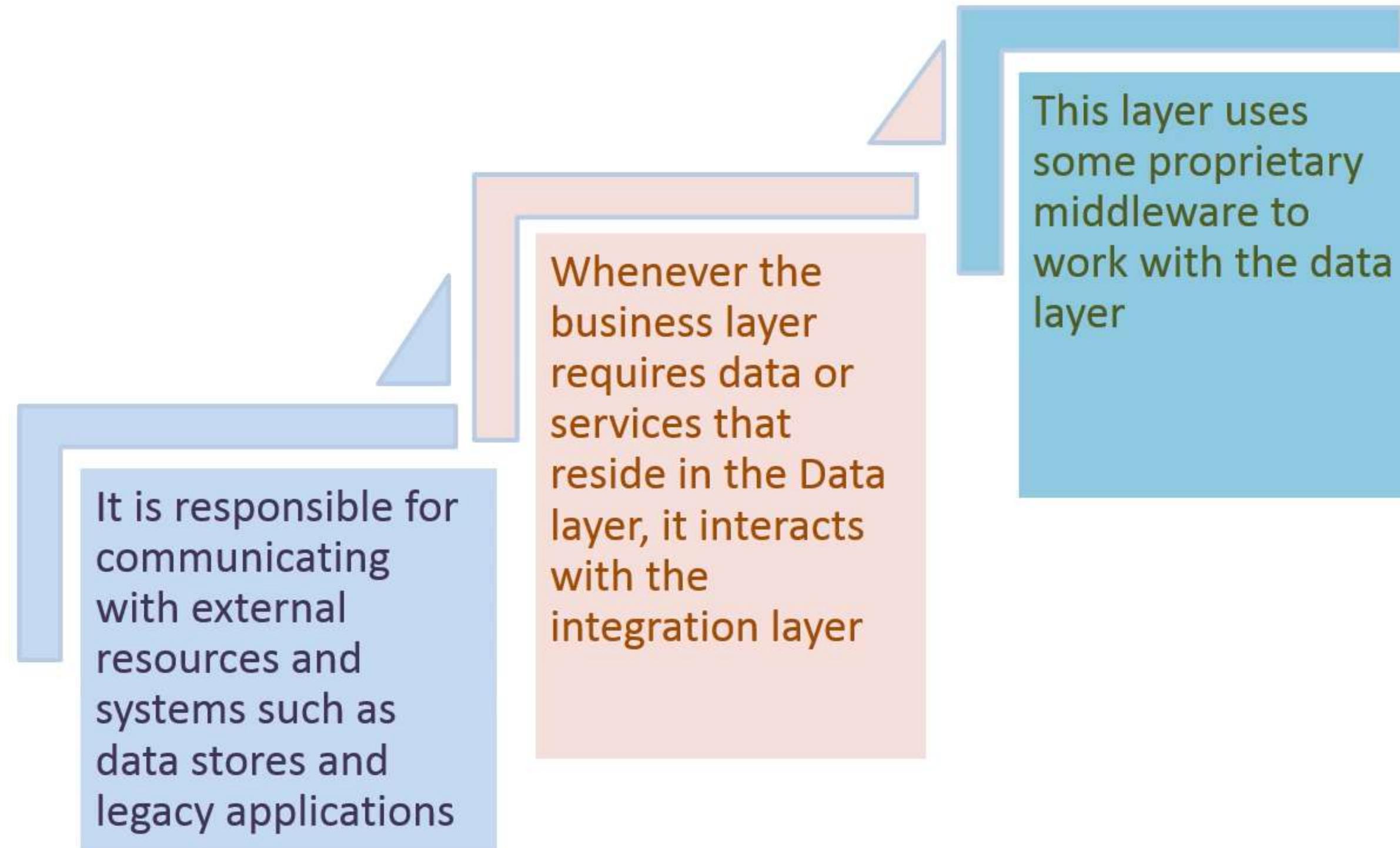
Advantages of layers

- 
- 1 • Individual layers can be managed independently
 - 2 • Easier to maintain
 - 3 • Components are reusable
 - 4 • Faster development (Division of work)
 - 5 • Security setting of each layer can be adapted to its use and requirements

Adding Integration Layer



Integration Layer

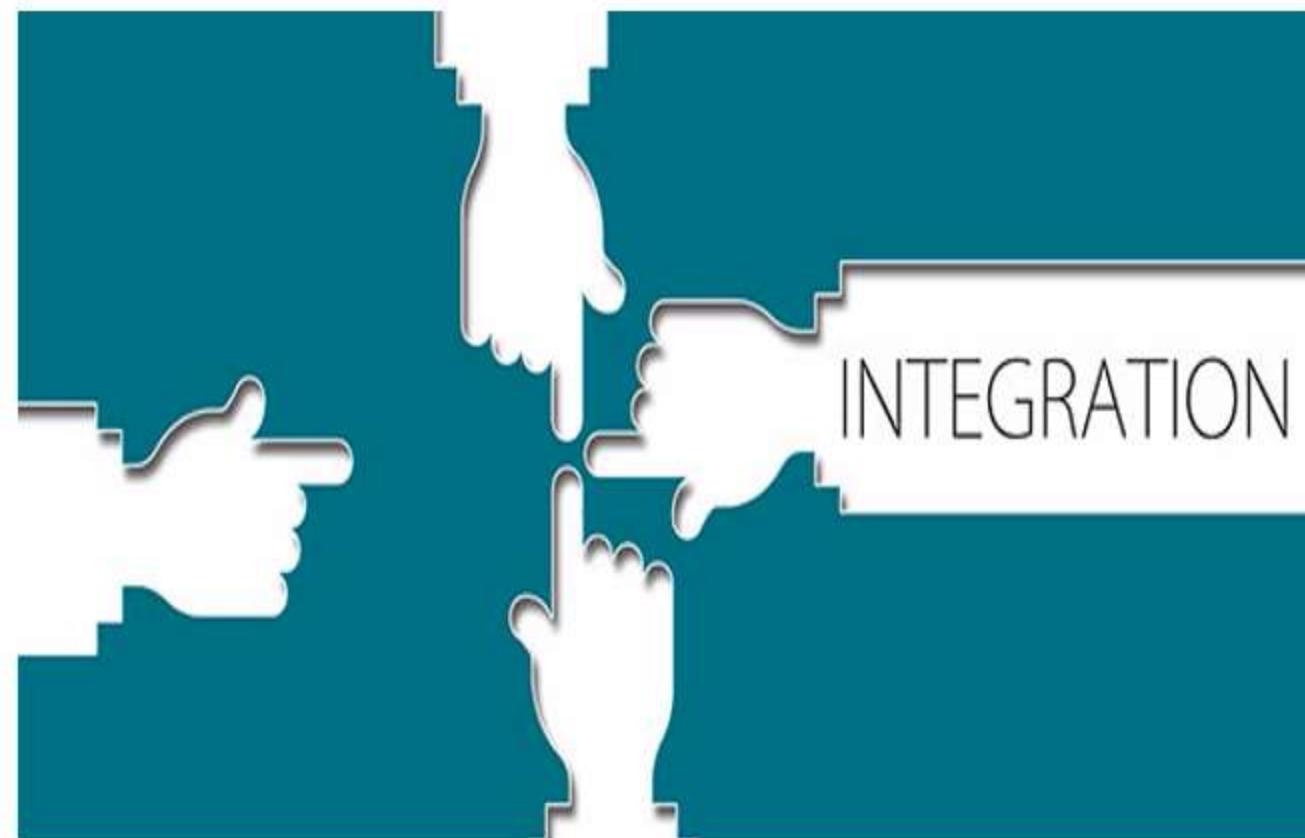


Integration frameworks

Integration frameworks provide a model for interaction and communication between mutually interacting software applications in service-oriented architecture (SOA).

Integration frameworks help to :

- ❖ Integrate with an existing application
- ❖ Connect to a third party web service
- ❖ Build a complex transaction-processing platform.



Integration
of different
Technologies



Integration frameworks



spring
Integration

Popular open-
source
Integration
frameworks:

Mule ESB

Integration frameworks



Spring Integration

- Supports lightweight messaging
- Allows integrations with external systems via declarative adapters

Apache Camel

- A light-weight Java framework that makes integration a lot easier for developers
- Easily exchange, route, and transform data through various protocols

Mule ESB

- Enterprise Service Bus facilitates communication between two mutually interacting software applications
- Enables the developers to easily connect and integrate diverse distributive systems

Layer Implementation

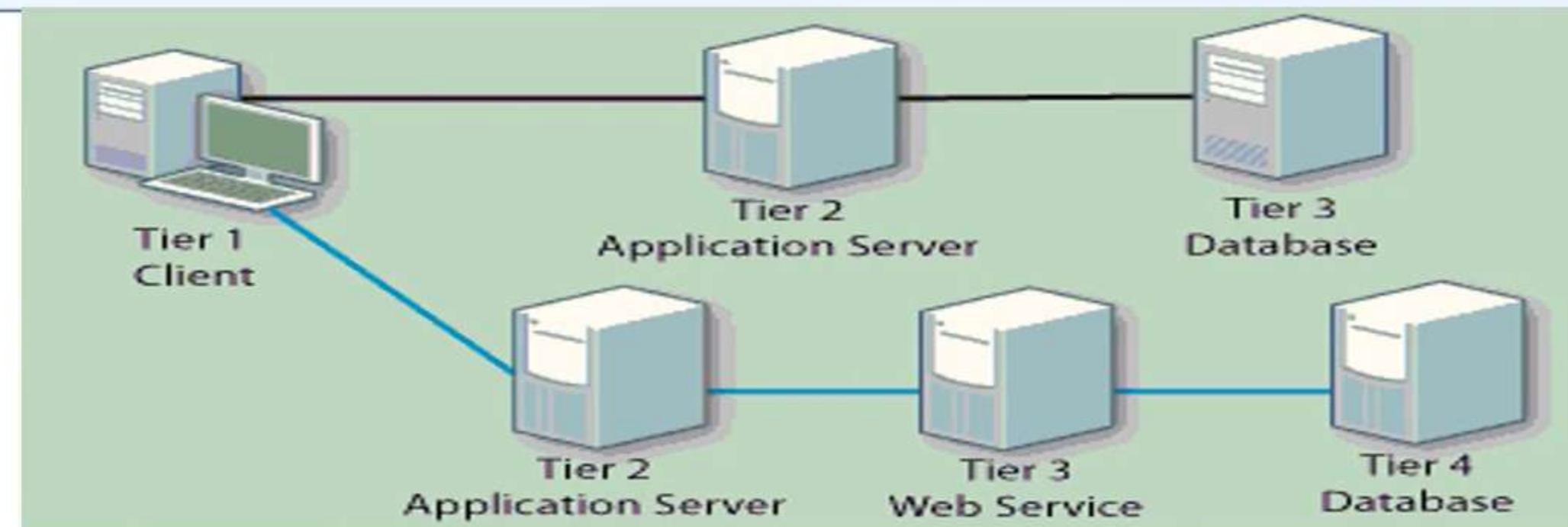


Tiers

A tier is a physical structuring mechanism for the system

Tiers represent the physical separation of the presentation, business and data functionality of your design across separate computers and systems

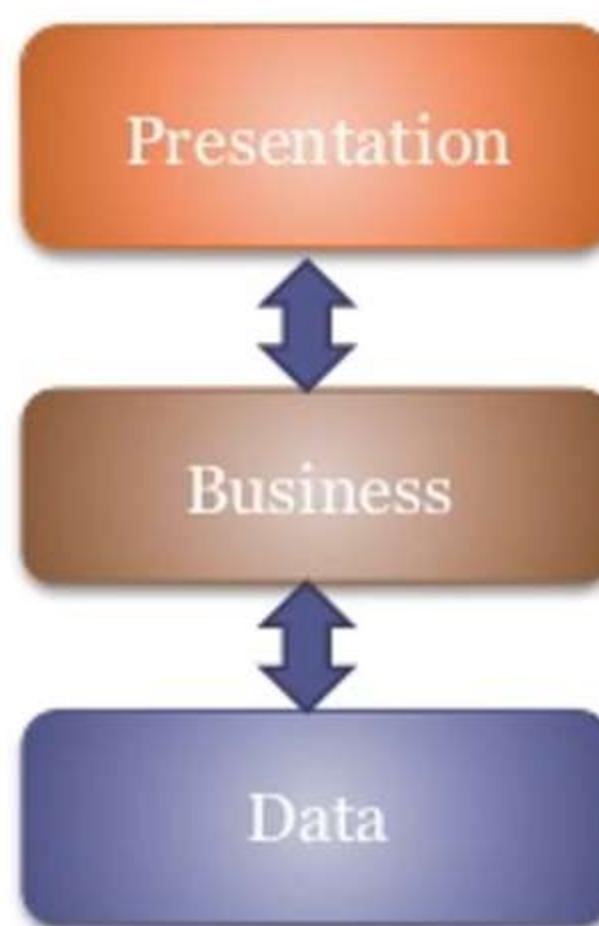
Usually an individual running server is one tier. Several servers may also be counted as one tier



Layers Vs. Tiers

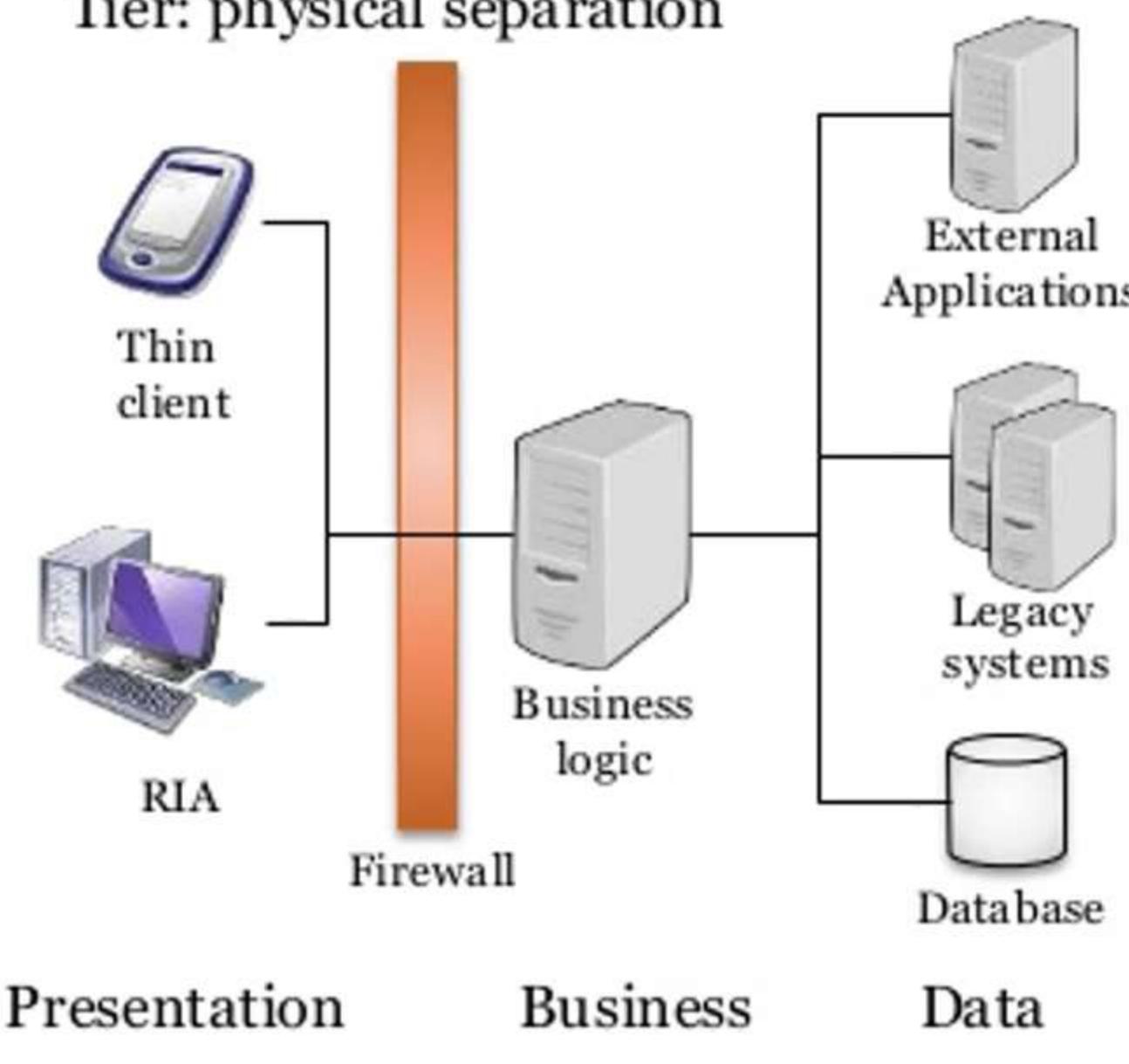
Layers ≠ Tiers

Layer: conceptual separation



3-Layers

Tier: physical separation

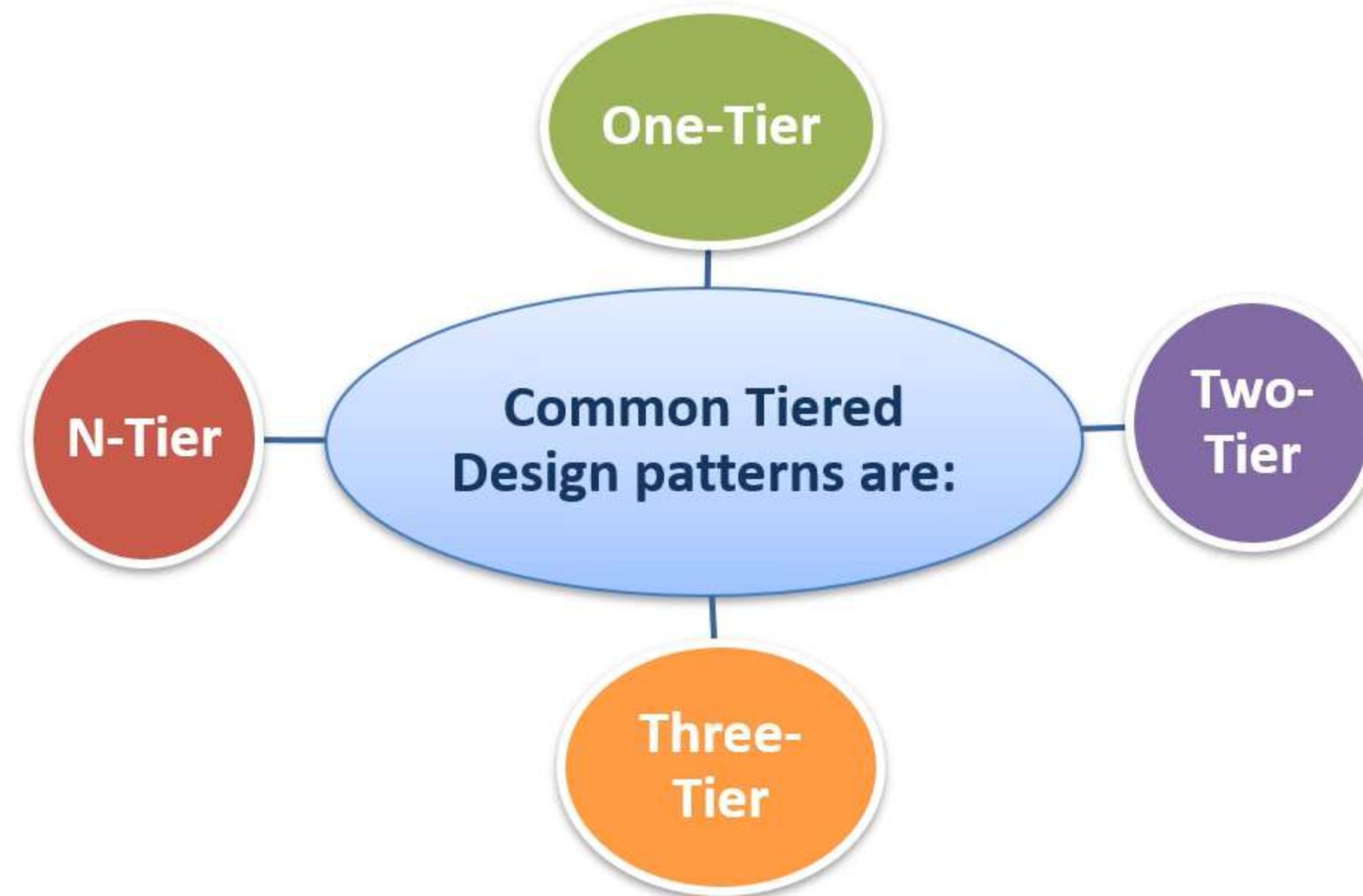


3-tiers

Tiers

Note

- Each layer may run in an individual tier
- Multiple layers may also be able to run in one tier
- A layer may also be able to run in multiple tiers



One-tier Architecture



It involves putting all of the required components of a software application on a single runtime environment.

Presentation, Services and data storage were merged into a single tier .

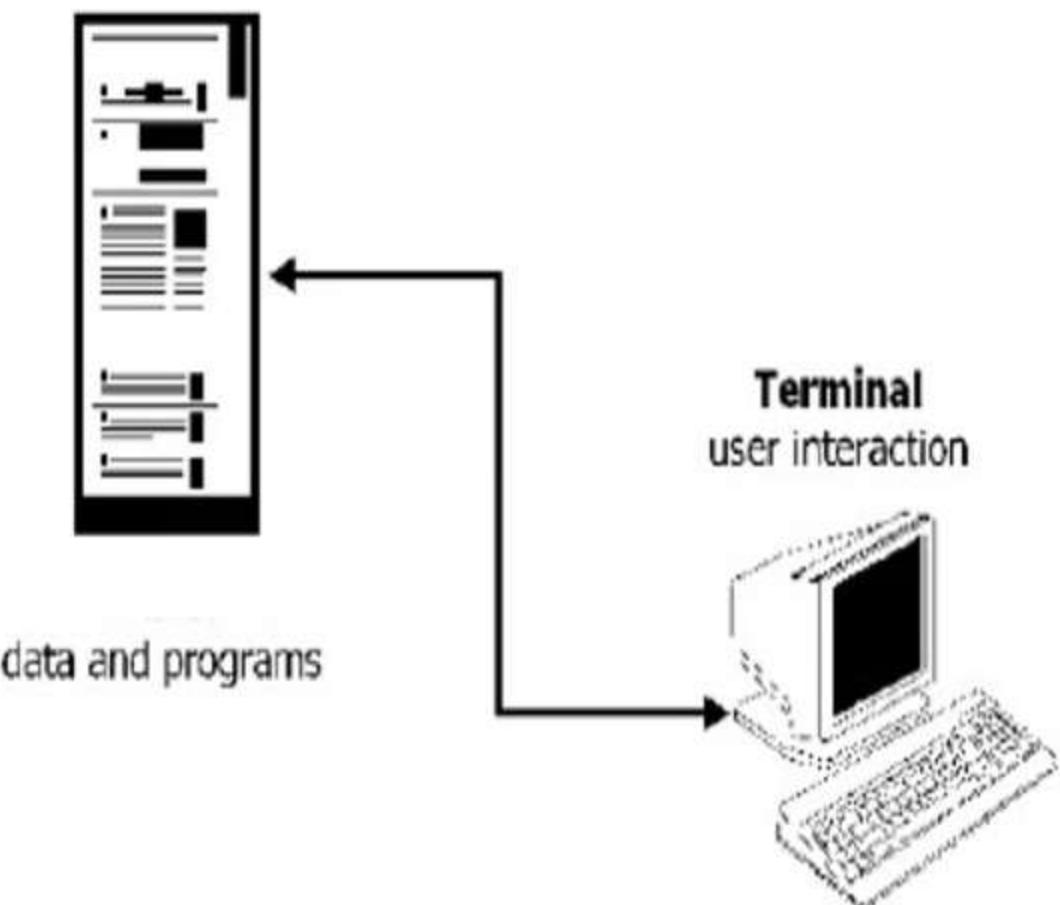
Well suited for simple applications and mostly used by small organizations.

EXAMPLE

MP3 player, MS Office, etc.

One-tier Architecture

Pros.	Cons.
<ul style="list-style-type: none">• Easy to optimize performance• Efficient• Uncomplicated	<ul style="list-style-type: none">• Low Scalability• Less portability• Less Maintenance

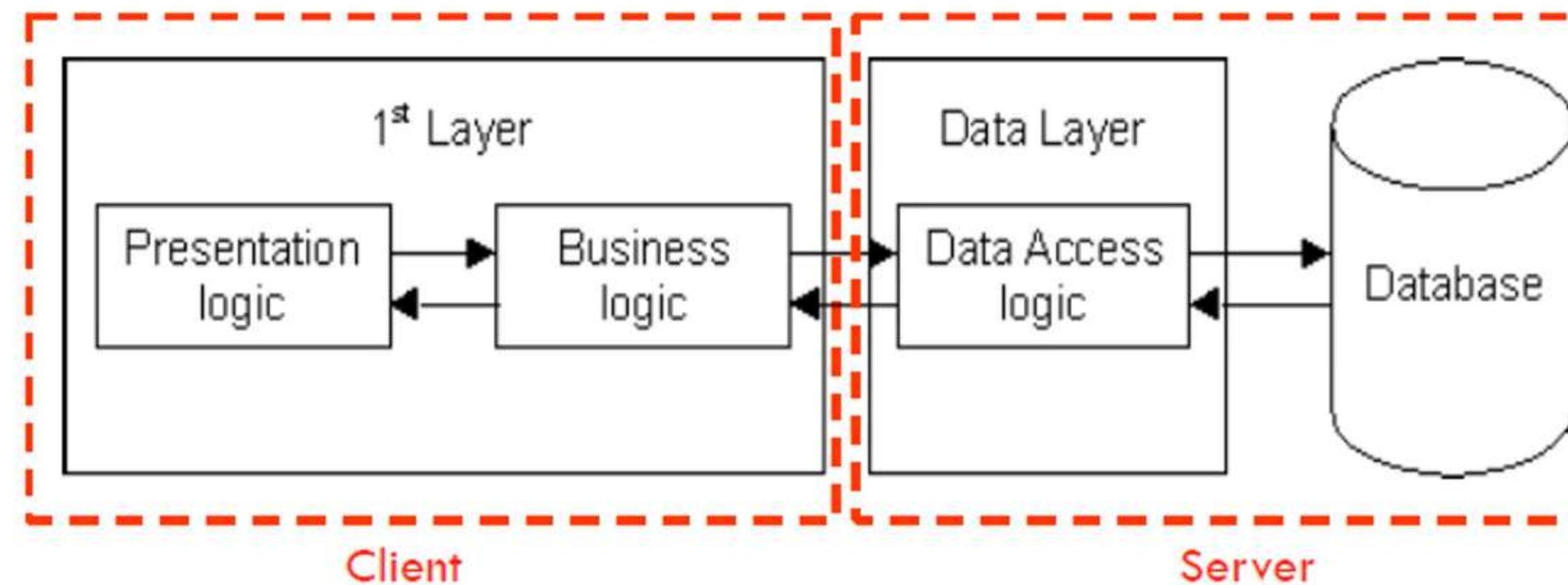


Two-tier Architecture

The Two-tier architecture is divided into two parts:

1. Client Application (Client Tier)
2. Database (Data Tier)

It is also known as **client-server** application



Two-tier Architecture



In this architecture, either presentation layers and application layer will run on one computer or application layer and data layer will run on one computer

Database or data source runs on Server

- Separated from client
- Easy to switch to a different database

Presentation and logic layers tightly connected

- Heavy load on server
- Potential congestion on network
- Presentation still tied to business logic

Three-tier Architecture

The Three-tier architecture is divided into three parts:

1. Presentation layer (Client Tier)
2. Application layer (Business Tier)
3. Database layer (Data Tier)

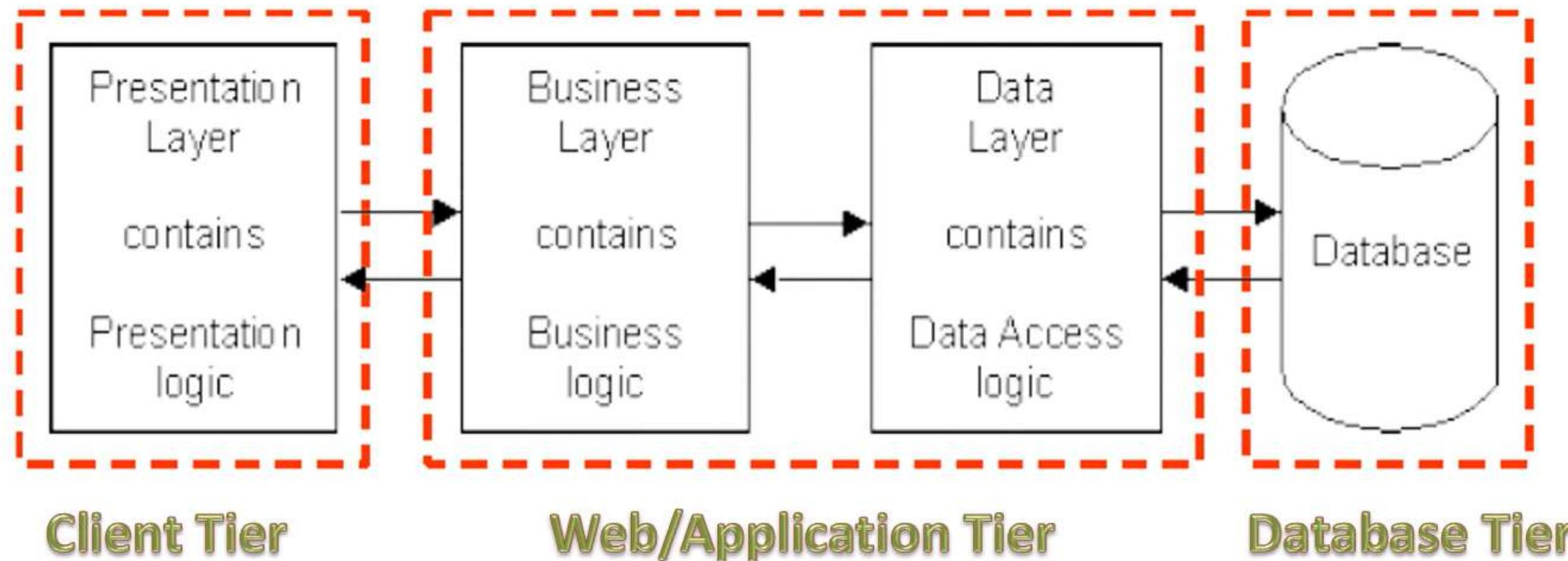
- Client system handles Presentation layer
- Application server handles Application layer
- Server system handles Data layer.

EXAMPLE

Web Based applications

Three-tier Architecture

All three layers will run in three separate computers



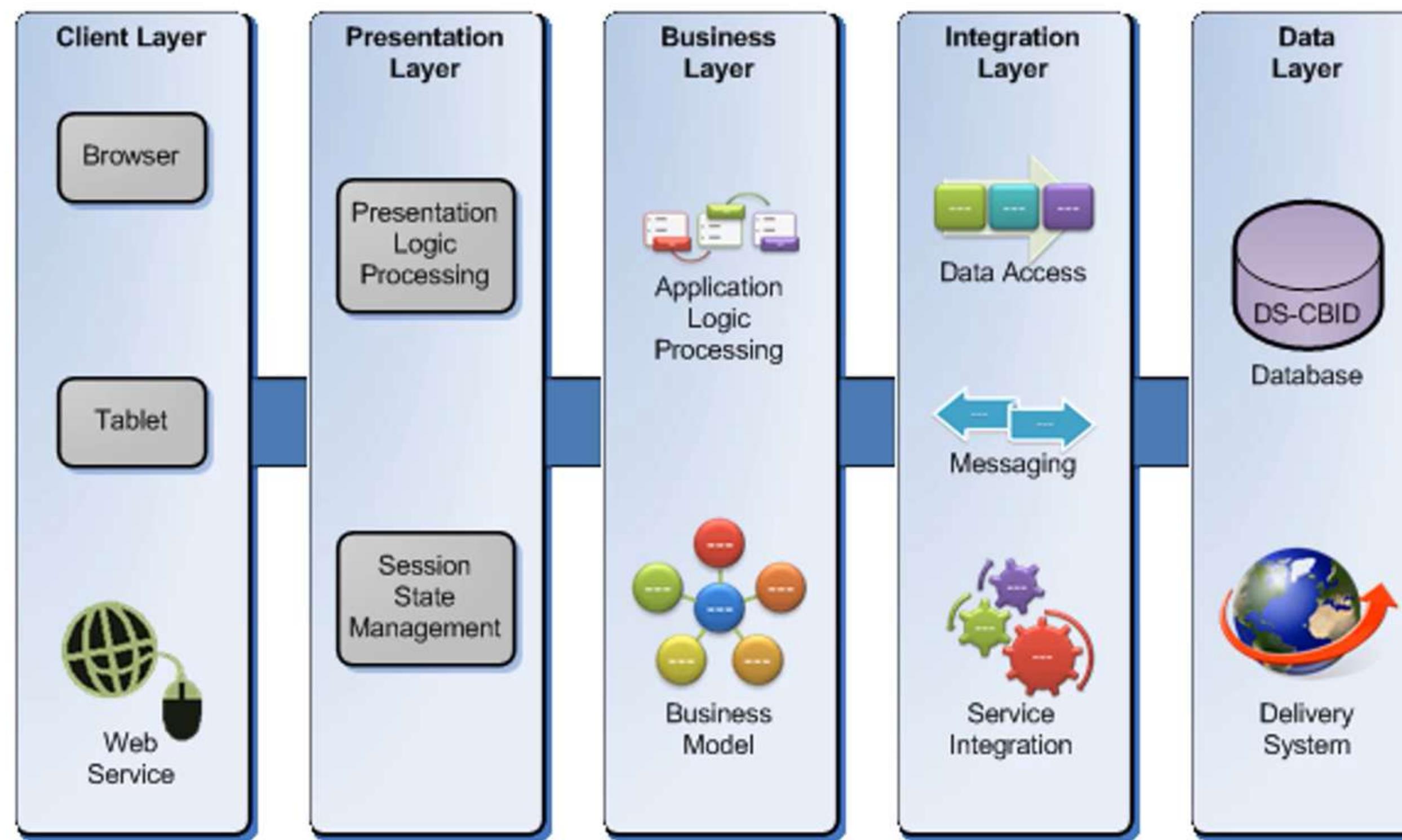
N-tier Architecture

- It is an industry -proven software architecture model
- Three or more tiers are involved. Some layers in 3-layer architecture can be broken to many layers
- Number of application servers are increased and represented in individual tiers in order to distribute the business logic, So the logic will be distributed

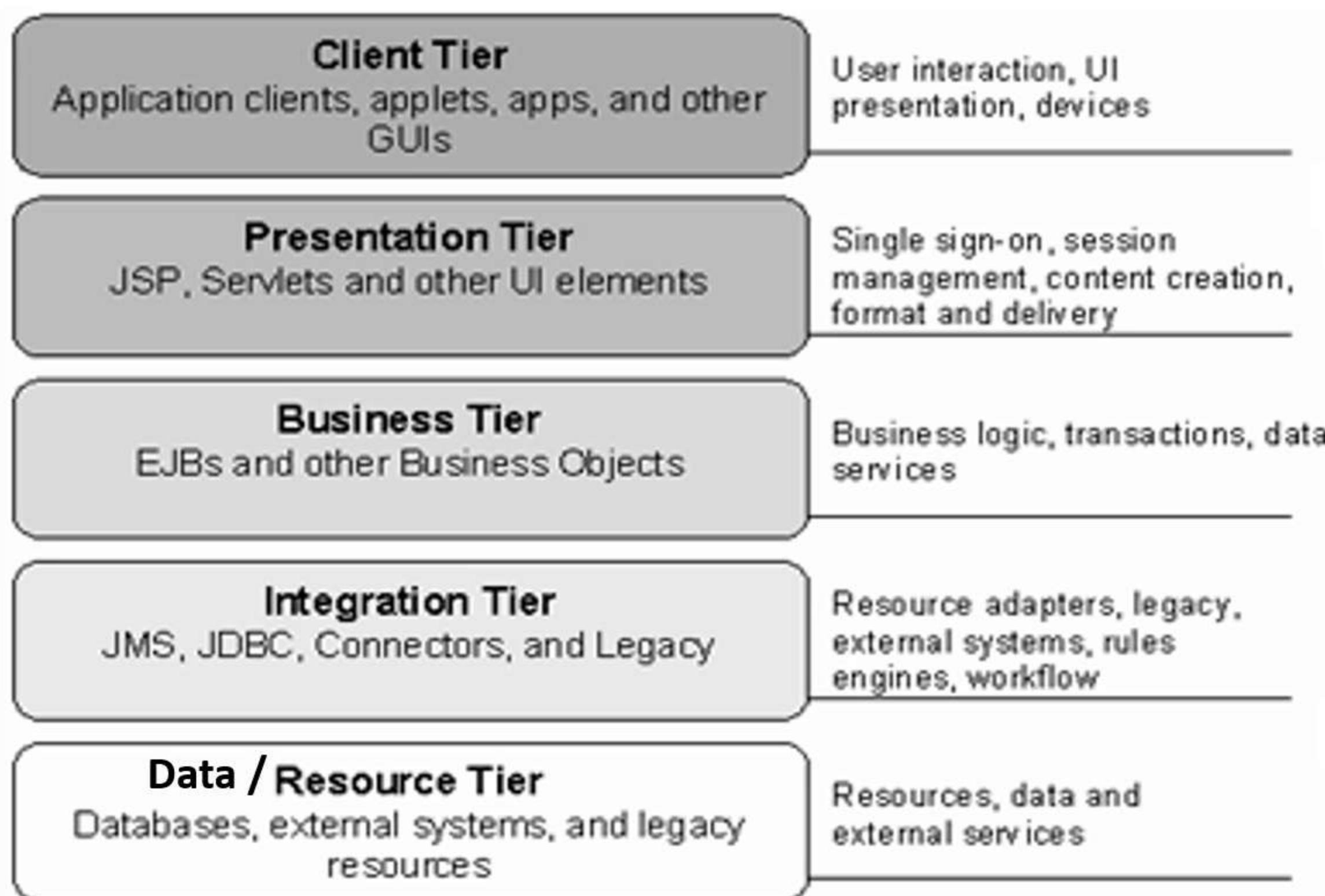
Similar to three tier architecture

N-Tier application is also known as Distributed application and using for web based enterprise applications

Layers in N-tier Architecture



N-tier Architecture Implementation



Overview of each Tier:

1. Client Tier:

- This layer is involved with users directly
- There may be several different types of clients coexisting, such as: Window Form, HTML pages, Mobile app, etc.

2. Client presentation Tier:

- Contains Presentation logic needed by clients. Like JSP in apache web server

Overview of each Tier:

3. Business Tier

- Handles and encapsulates all business data and logics; also called domain layer

4. Integration Tier:

- This tier is responsible for communicating with external resources and systems such as data stores and legacy applications

5. Data Tier:

- Contains external data source such as a sql server, db2 oracle, mysql database, etc.

pros and cons of N-tier Architecture



Pros

- Better scalability and fault tolerance ability
- Easy maintenance and better reusability
- Finer security control
- Independent tier upgrading without affecting other tiers
- Easy and efficient development

Cons

- Slow performance of the whole application
- More cost for hardware, network, maintenance and deployment

Summary

- Tiered Architecture
- Presentation Layer
- Services Layer
- Data Layer and implementation
- Integration Layers



SOFTWARE ENGINEERING FUNDAMENTALS



In this module you will learn

- Process
- Software Development Life Cycle
- Waterfall model
- V model
- Prototype model
- RAD
- Spiral Model
- ETVX



Terms

Program

- A computer program is a sequence of instructions written to perform a specified task in a computer

Software

- A software is a set of programs, procedures and its documentation concerned with the operation of a data processing system

Software Process

- A Process is a series of definable, repeatable, and measurable tasks leading to a useful result

Software Development

Ad-hoc Software Development (till 1960's)

- The Software was developed on a Trial & Error basis
- No Specific Process was followed during the development of the Product
- Defects were detected only after the product was delivered to the external Users

Terms

Software Engineering is defined as

- Software engineering is the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software
- An establishment and use of sound engineering principles in order to obtain an economical software that is reliable and works efficiently on real machines

Software Crisis

Software fails to meet the user requirements

Software crashes frequently

Development of Software became expensive

Difficult to alter, debug, and enhance the software

Software was often delivered late

Software used resources non-optimally



Process



How to make coffee.
What are the steps??



Process



How to make
coffee. What are
the steps??



A Process is a series of definable, repeatable, and measurable tasks leading to a useful result.

Software Development Process

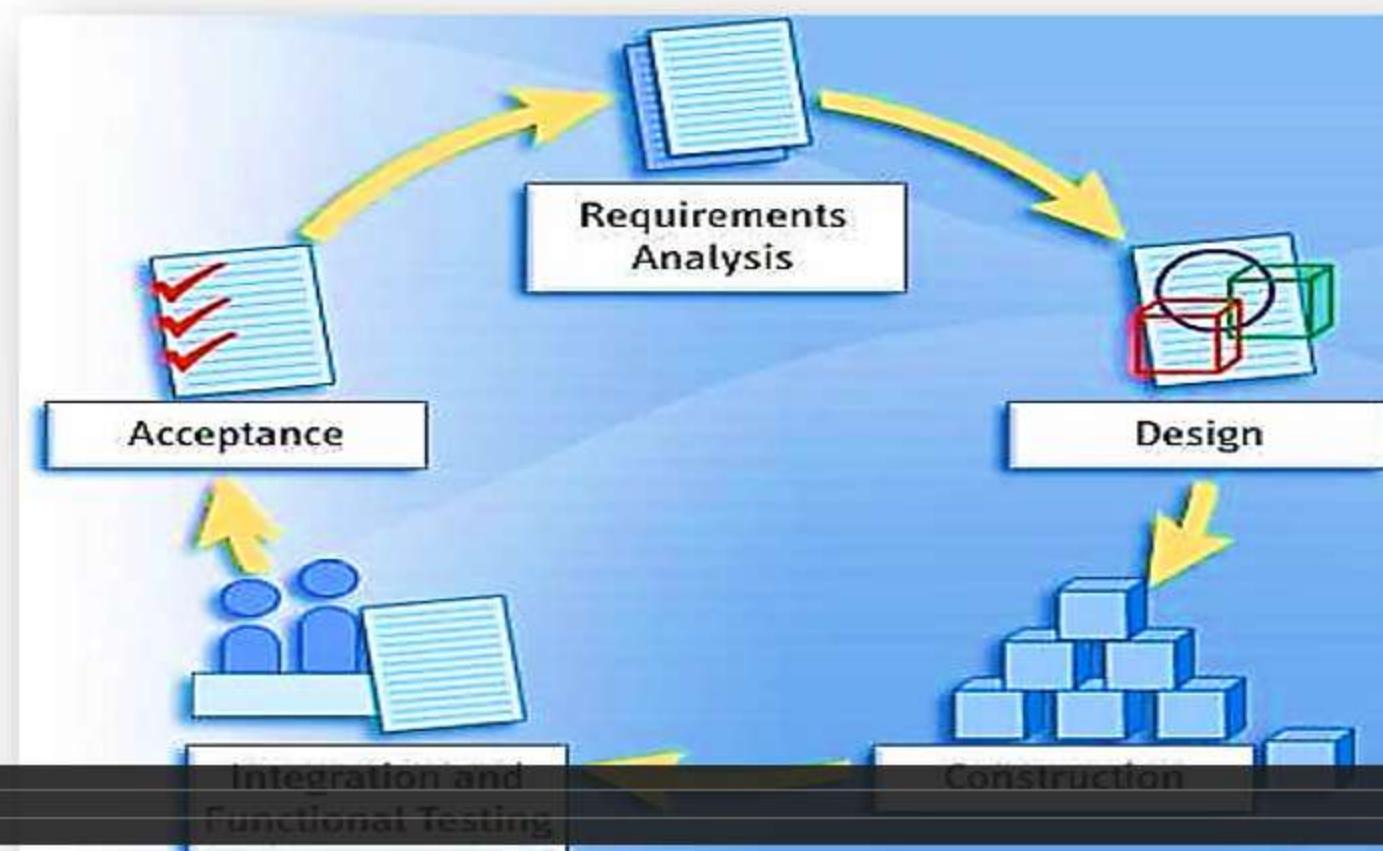


Software Development process involves transformation of user needs into an effective software solution.

SDLC

Software Development Life Cycle is the process used in a project to develop a software product

It describes how the development activities are performed and how the development phases follow each other



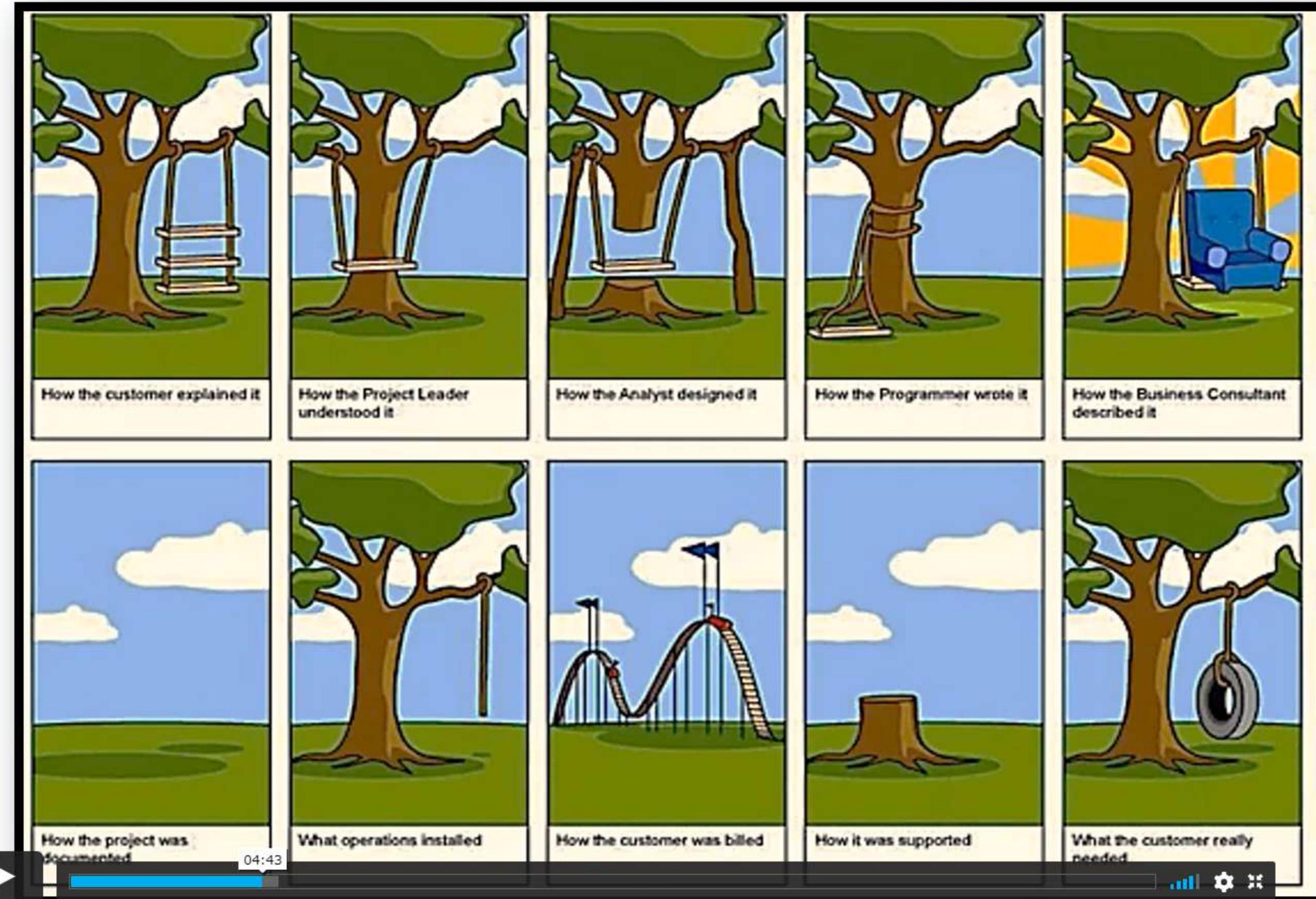
04:18

Integration and Functional Testing

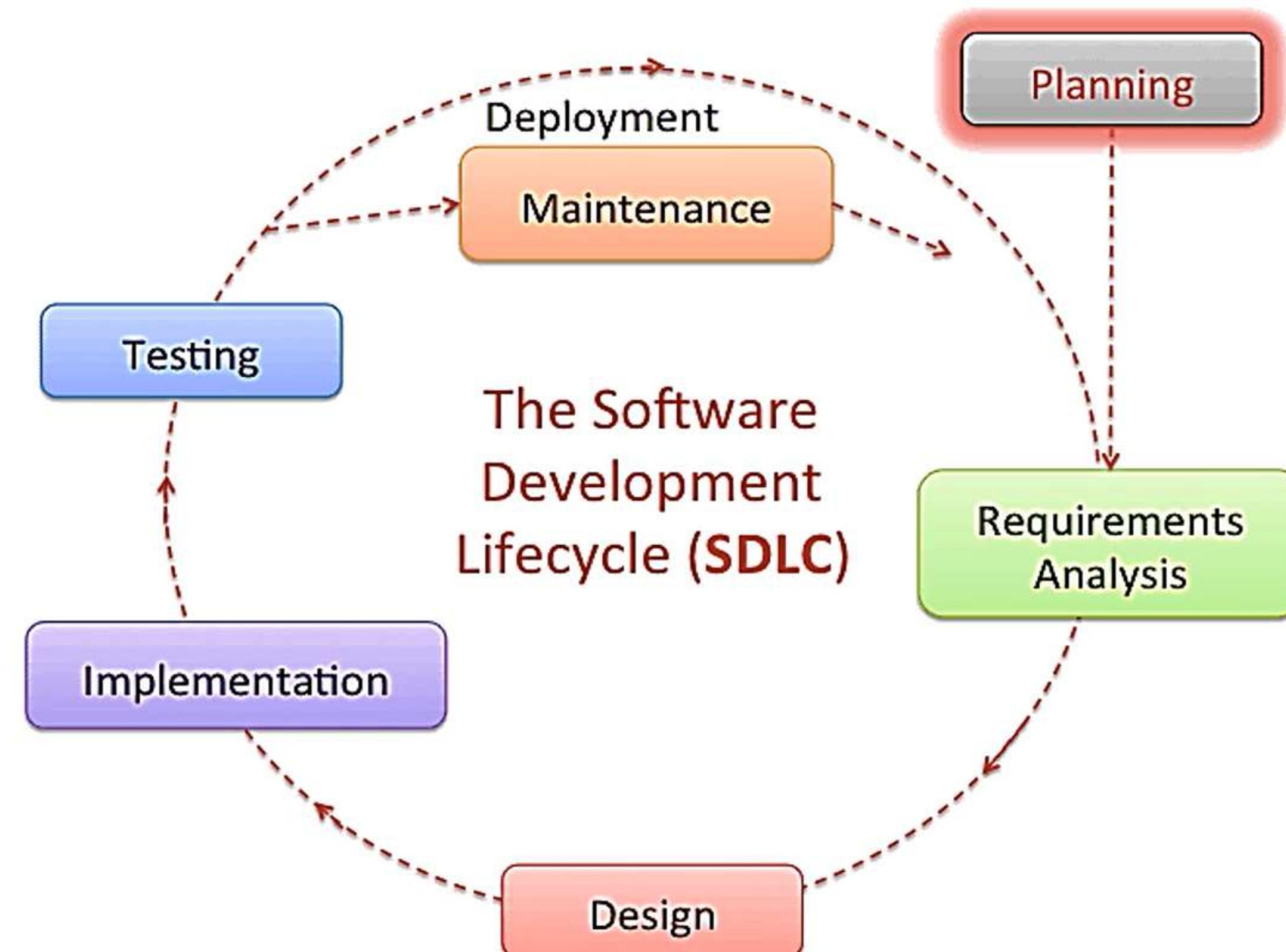
Construction



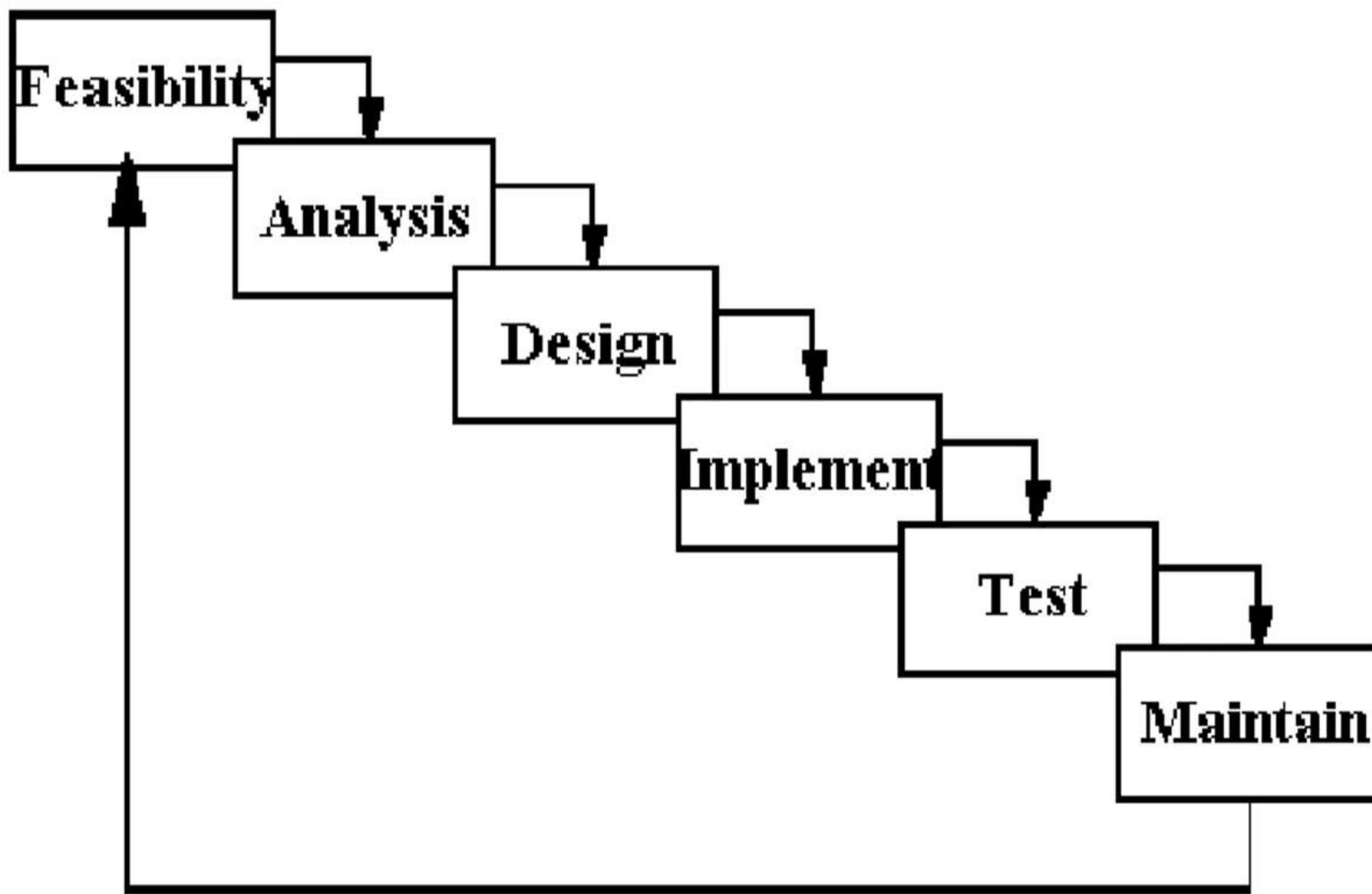
Importance of SDLC



Software Development Life Cycle



Phases in SDLC



Analysis

The goal of the system analysis is to define the requirements of the system

Requirement gathering requires client as well as the service provider to get the detailed and accurate requirements

SRS(Software Requirement Specification) is the primary artifact of Analysis phase

Activities in Analysis Phase

Requirements gathering and analysis

Preparing Requirements Specification(SRS)

Design

Software design deals with transforming the customer requirements into a set of documents that is suitable for implementation in a programming language

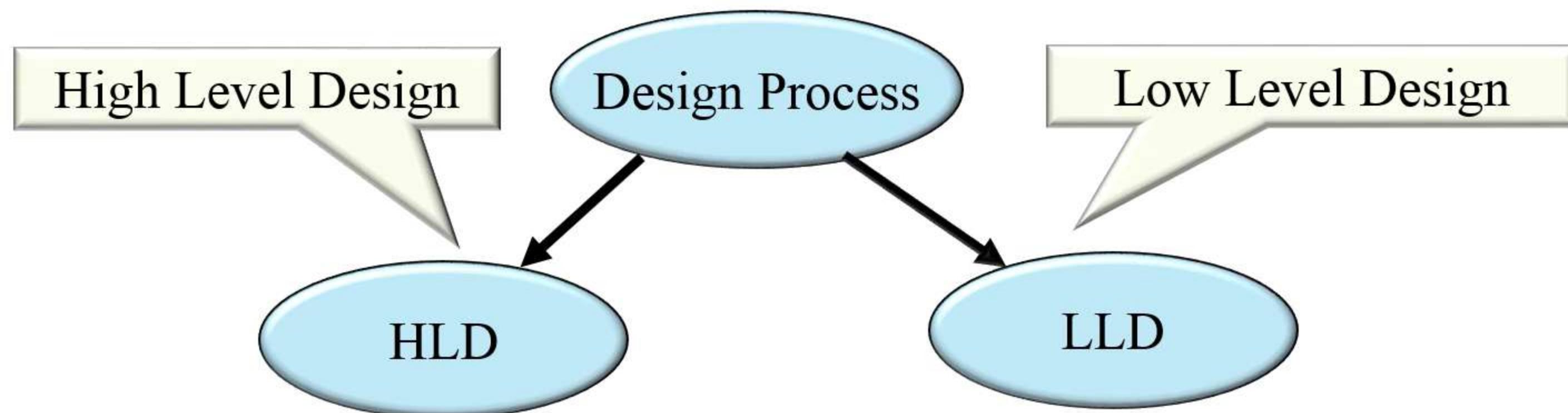
It is the process of defining the architecture, interface, component and other characteristics of a system

The design stage takes the requirements identified in the approved requirements document (SRS) as its initial input

Levels of a Design

Decompose the entire project into units / modules and identify the system architecture, data structure and processing logic

- DD(Design Document)= HLD + LLD



Construction(Code + Unit Testing)



Modular and subsystem programming code will be accomplished
during this stage

Unit testing /module testing is done in this stage by the developers

This stage produces the source code, executable, and databases
applicable



Testing

Testing is the process of executing the program with the intent of finding errors

Software testing is a process of verifying and validating that a software application or program meets the business and technical requirements

Levels of Testing

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing



Testing

Software testing includes

Verification

- Confirms that the software meets its technical specifications

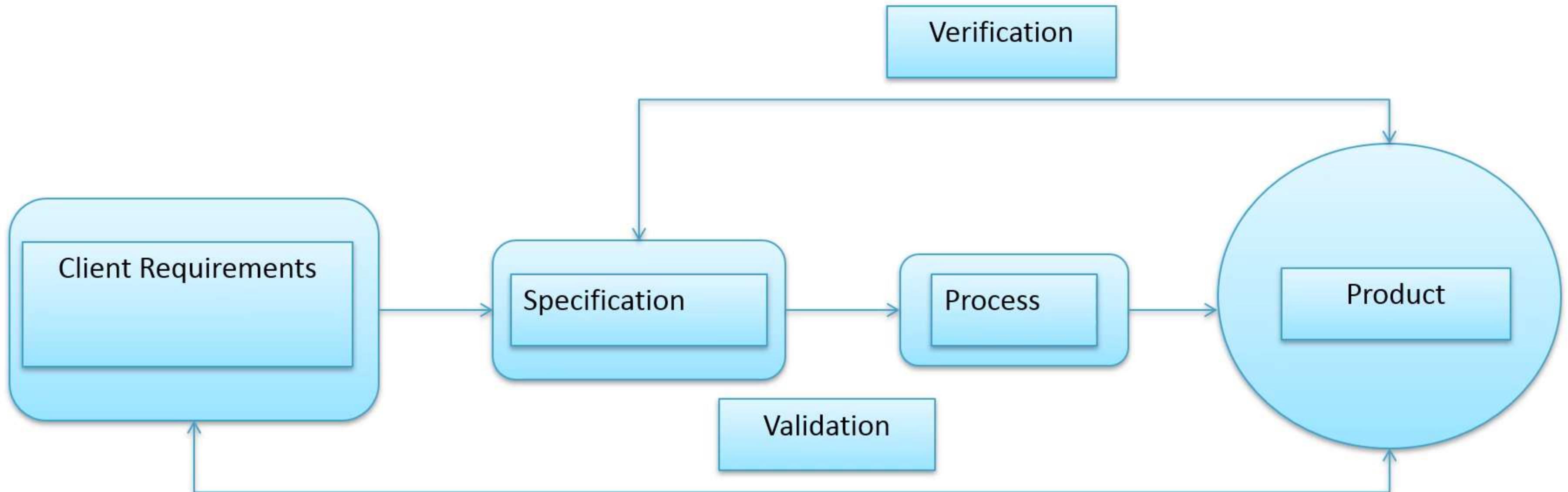
Validation

- Confirms that the software meets the business requirements

Defect

- Variance between the expected and actual result

Verification, Validation and Defect Finding



Levels of Testing

Unit testing

System testing

Integration Testing

Acceptance Testing

Maintenance

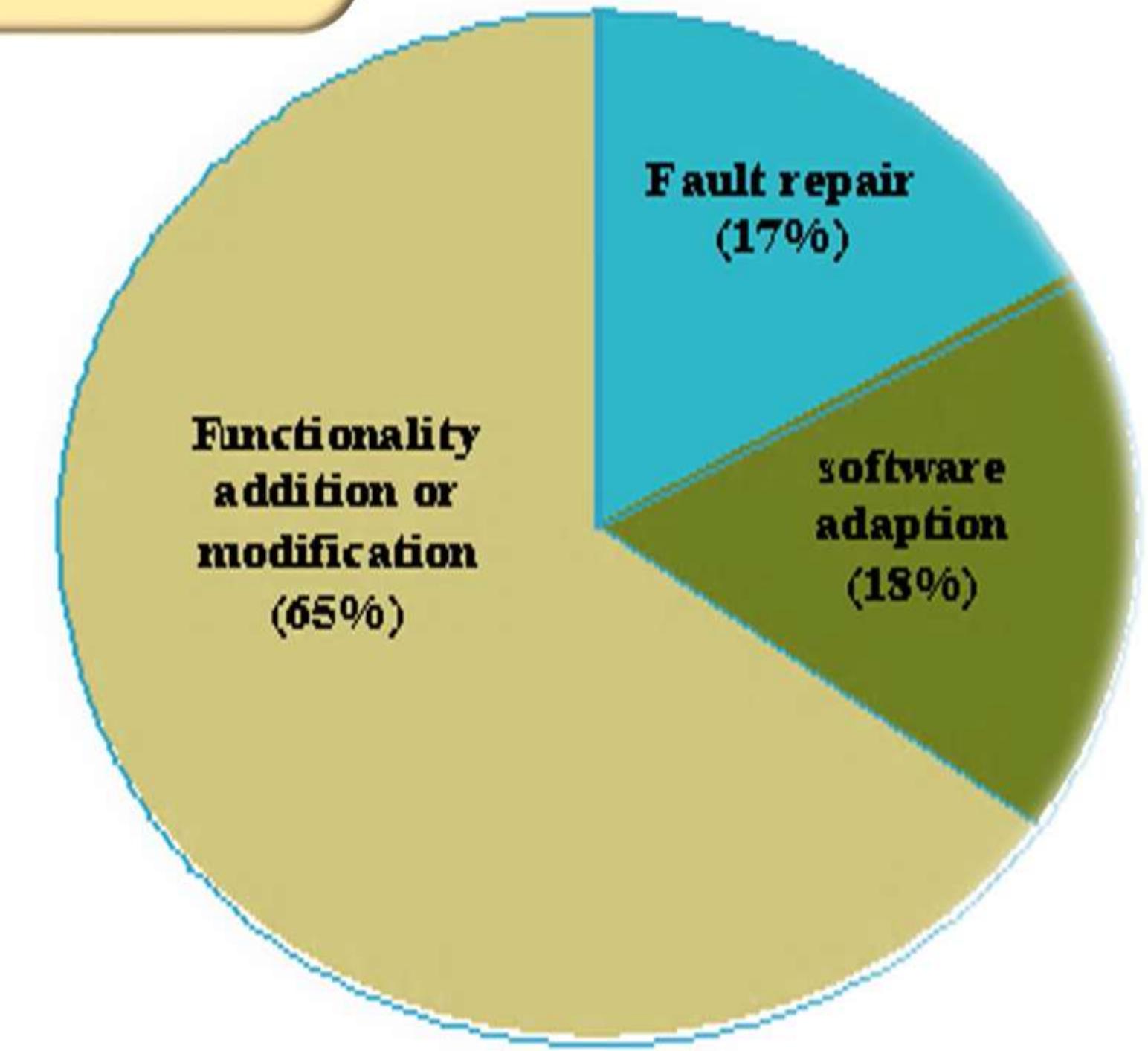


My house is leaking. It's a 5 year old house. I think it is time to do patch work.

Functionality addition or modification (65%)

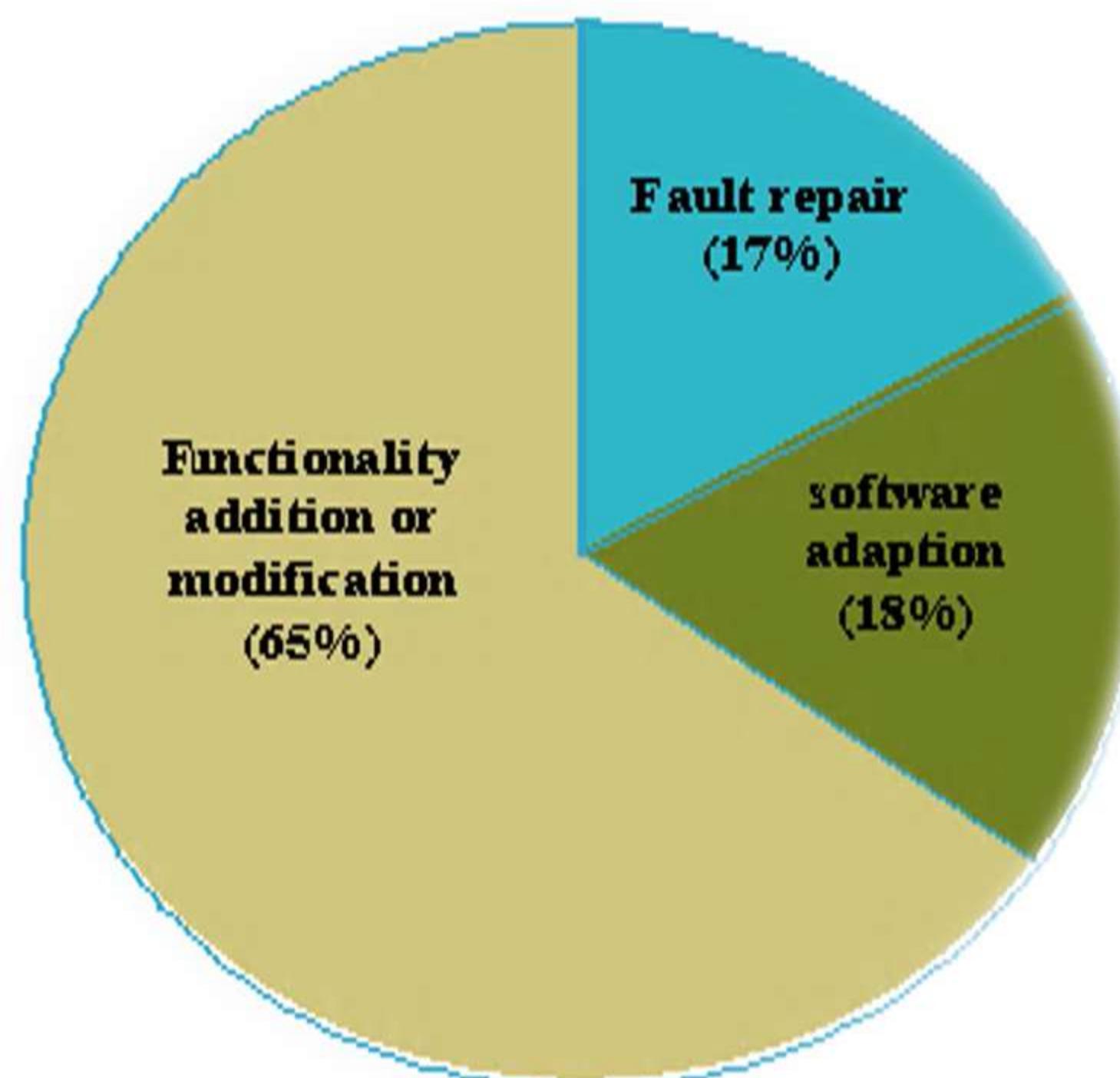
Fault repair (17%)

software adaption (18%)



Maintenance

Changes or enhancements happen everywhere. Software is no exemption. Any change that is made to the software after it is deployed is known as maintenance.



Functionality addition or modification (65%)

Fault repair (17%)

software adaption (18%)

Software Development Life Cycle Models

Waterfall model

V-model

Prototype model

RAD (Rapid Application Development)

Incremental model

Spiral model

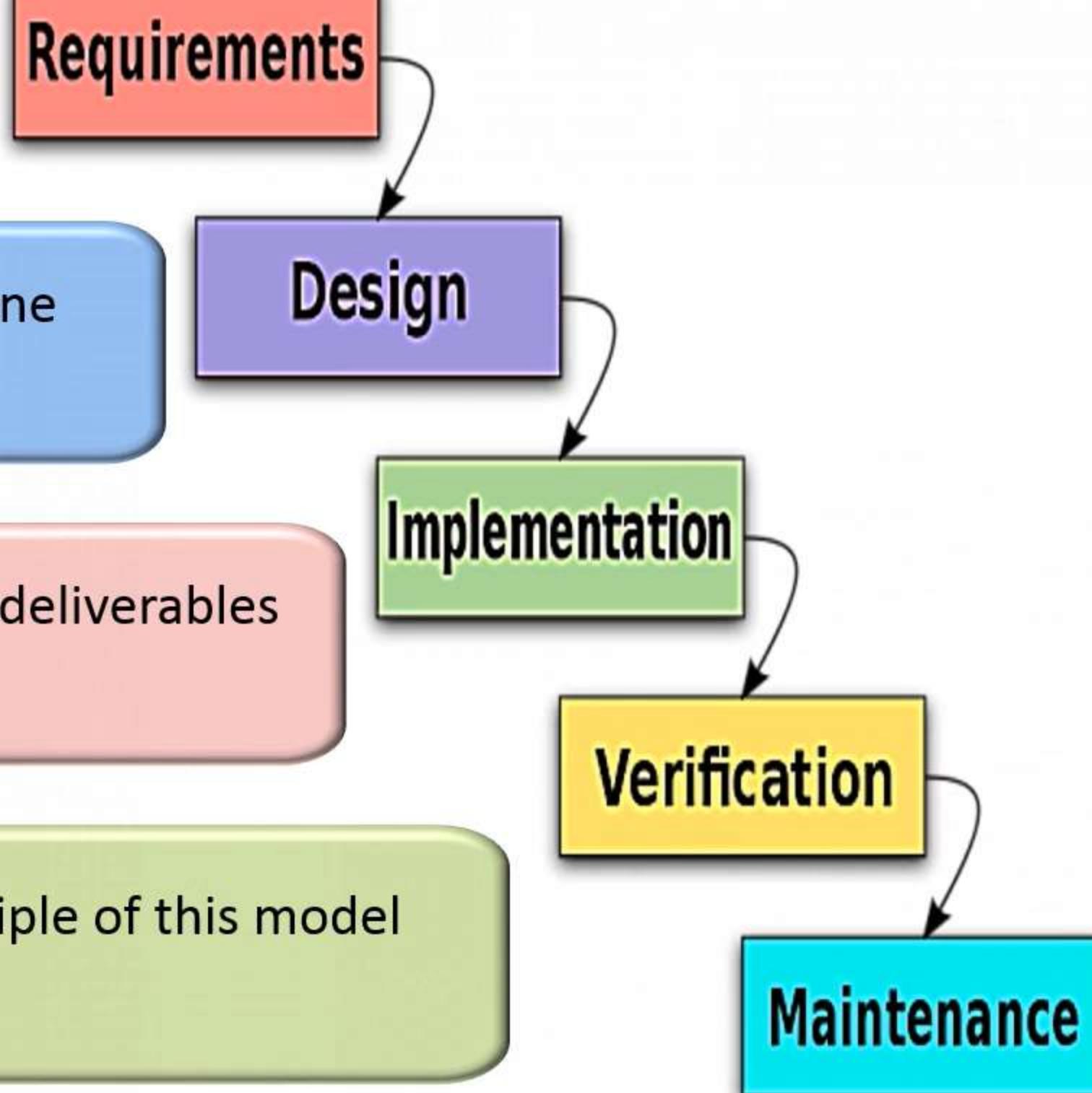
Waterfall Model

This model was proposed by Winston Royce in 1970

Waterfall model derives its name due to the cascading effect from one phase to the other as depicted in the diagram

Each phase has a well defined start and end point, with identifiable deliverables to the next phase

It is a linear sequential model, systematic in approach and the principle of this model suggests - "Can't retrieve to previous phase"



Advantages of Waterfall Model

- Simple and easy to use.
- Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process
- Phases are processed and completed one at a time
- Works well for smaller projects where requirements are very well understood
- Linear approach
- Equivalent importance to all the phases
- Contract Related issues can be addressed effectively

Limitations of Waterfall Model

- This model is suitable if the requirements are well-defined and stable
- User gets a feel of the system only at the later stages of development
- Backtracking cost is high in case of a problem
- Increased development of time and cost
- Systems must be defined up front
- Rigidity
- Hard to estimate costs & project overruns

Waterfall Model

**Suitable
when**

- Software requirements are clearly defined and known
- Product definition is stable
- Software development technologies and tools are well known
- New version of the existing software system is created

V-Model

Verification and Validation Model commonly known as V-Model evolved from waterfall Model

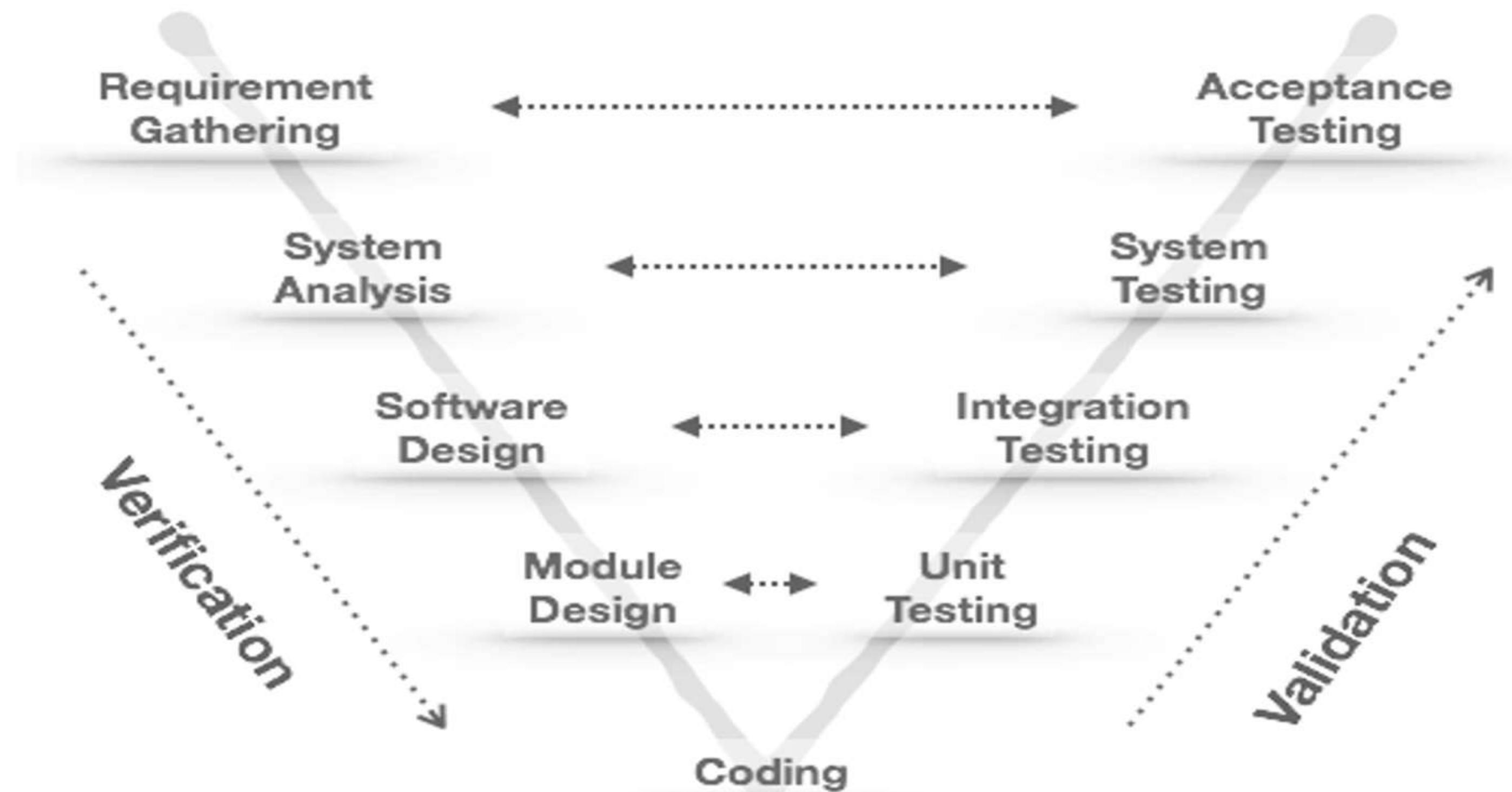
Each phase must be completed before the next phase begins

Testing is emphasized in this model more than in the waterfall model

It is a structured approach to testing

Testing is done from the earlier stage thereby bringing high quality into the development of our products

V-Model



Advantages of V-Model



Validation and Verification at each level of the stage containment

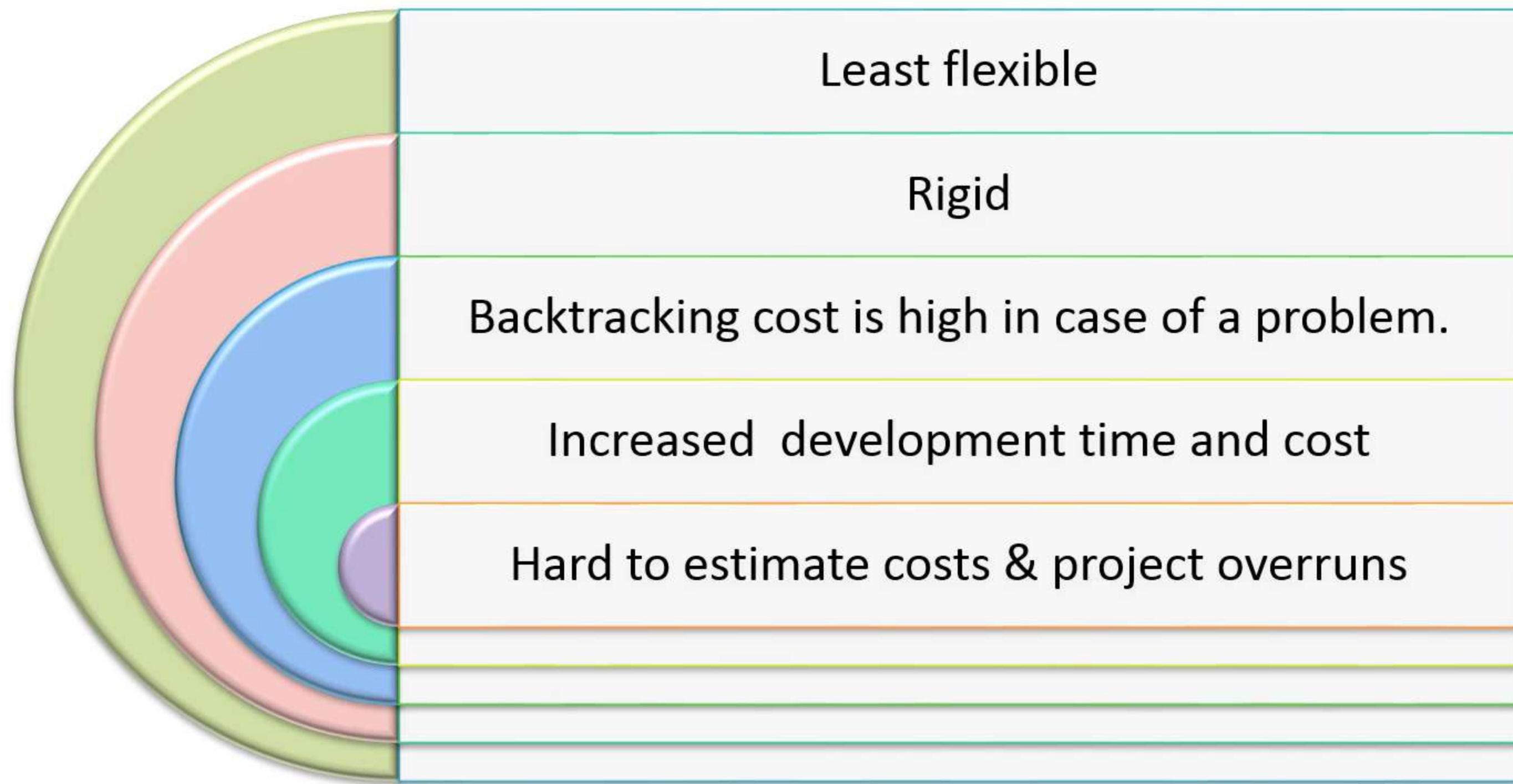
Stage containment mechanism

Avoids the downward flow of defect

Lower defect Resolution cost due to earlier detection

Allows testers to be active from the initial state of the project life cycle

Limitations of V-Model



Prototype Model

Creates prototypes, which is an incomplete version of the software program being developed.

Simulates only few aspects of the features of the System to be built



Prototype Model

Prototyping can also be used by the end users to describe and prove requirements that the developers have not considered.



Developers build a prototype during the requirements phase.



Prototype is evaluated by the end users to provide corrective feedback



Developers further refine the prototype based on feedback.



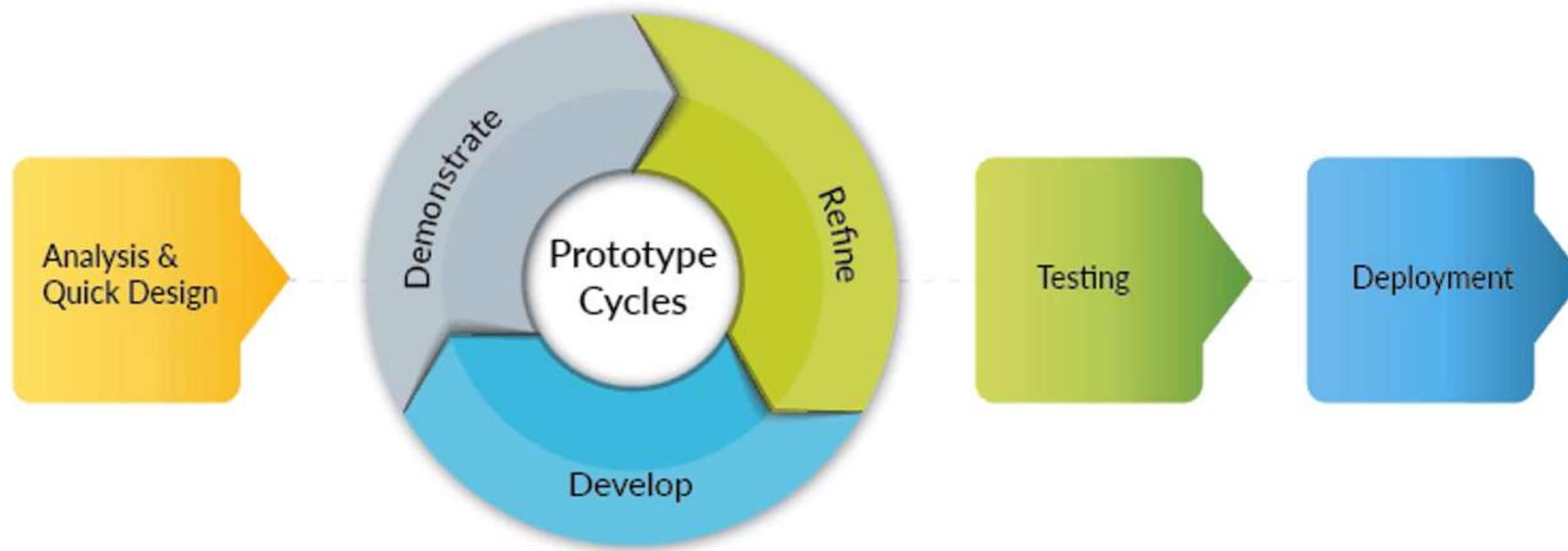
When the user is satisfied, the prototype code is brought up to the standards needed for the final product.

Prototype Model

The process of prototyping involves the following steps

- Identify basic requirements
- Develop Initial Prototype
- Review
- Revise and Enhance the Prototype

Prototype Model



What are the types of prototypes?

- Throw away
- Evolutionary

Throw away Prototype Model

This prototyping model is a ‘quick and dirty’ approach involving - Quick requirements assessment, analysis, design

Focuses on rapid construction

Ad-hoc development approach

Discards prototype after the objective is met

Throw away prototyping-Steps

Write preliminary requirements

Design the prototype

User experiences/uses the prototype, specifies new requirements.

Writing final requirements

Rapid Construction

Evolutionary Prototype Model

Requirements are prioritized and the code is developed initially for stable requirements, with an eye on quality

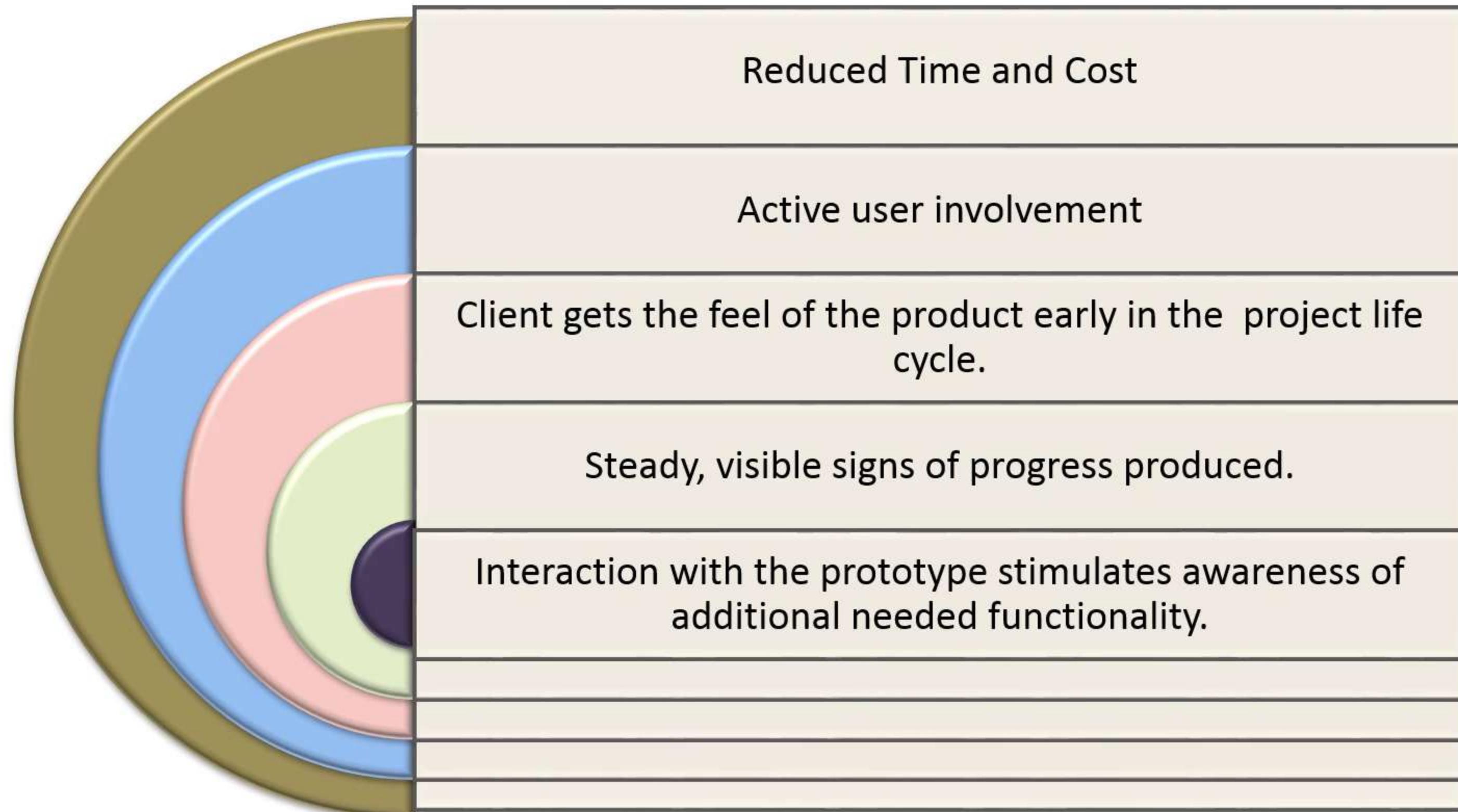
Software is continuously refined and augmented in close collaboration with the client

Build the software incrementally

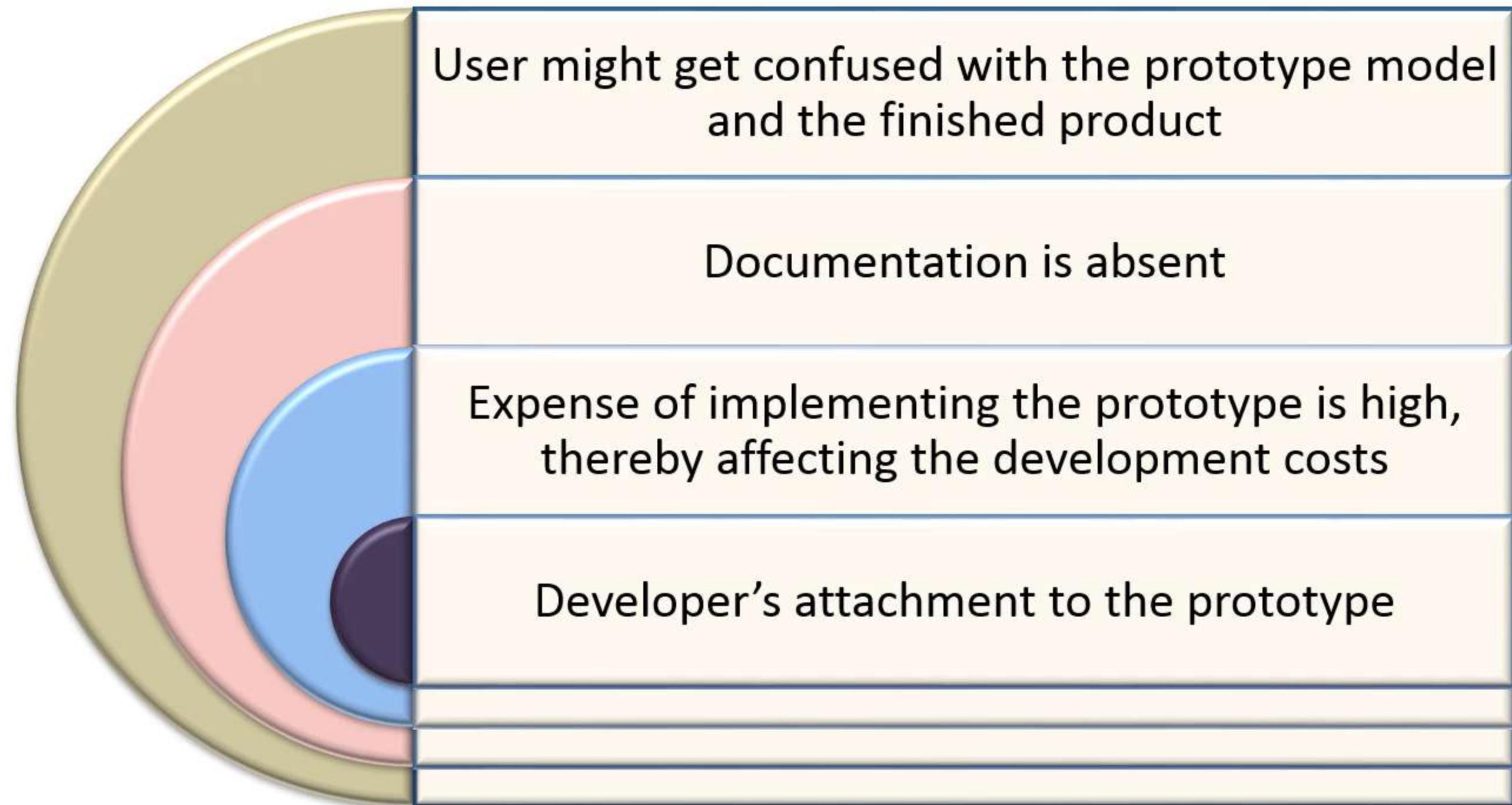
Adopt a rigorous, systematic approach

Iterative model

Advantages of Prototype Model



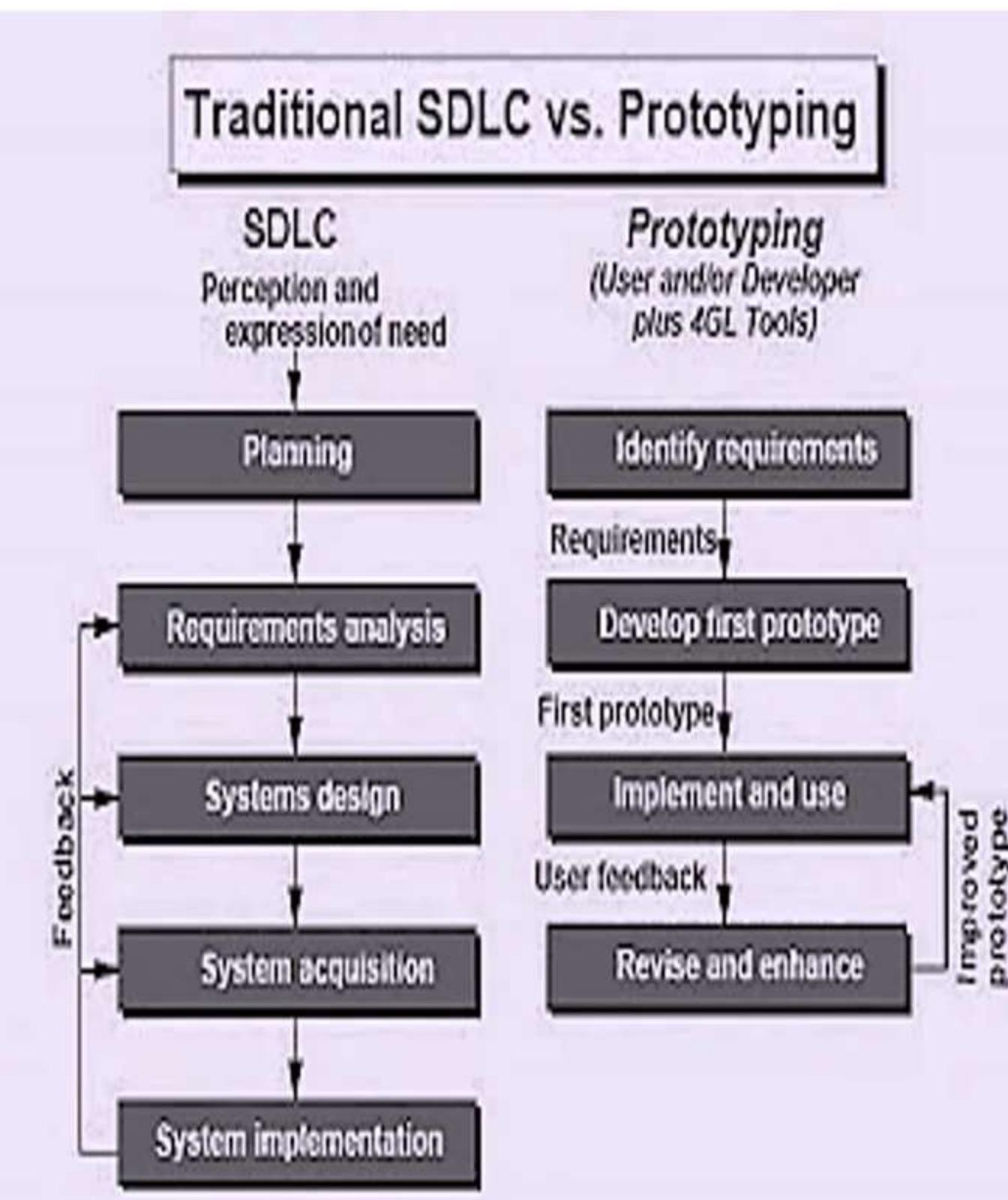
Limitations of Prototype Model



When to use Prototype Model?

When requirements
are unclear, use
"Throwaway
Prototype Model"

When requirements
are unstable, use
"Evolutionary
Prototype Model"



Rapid Application Development

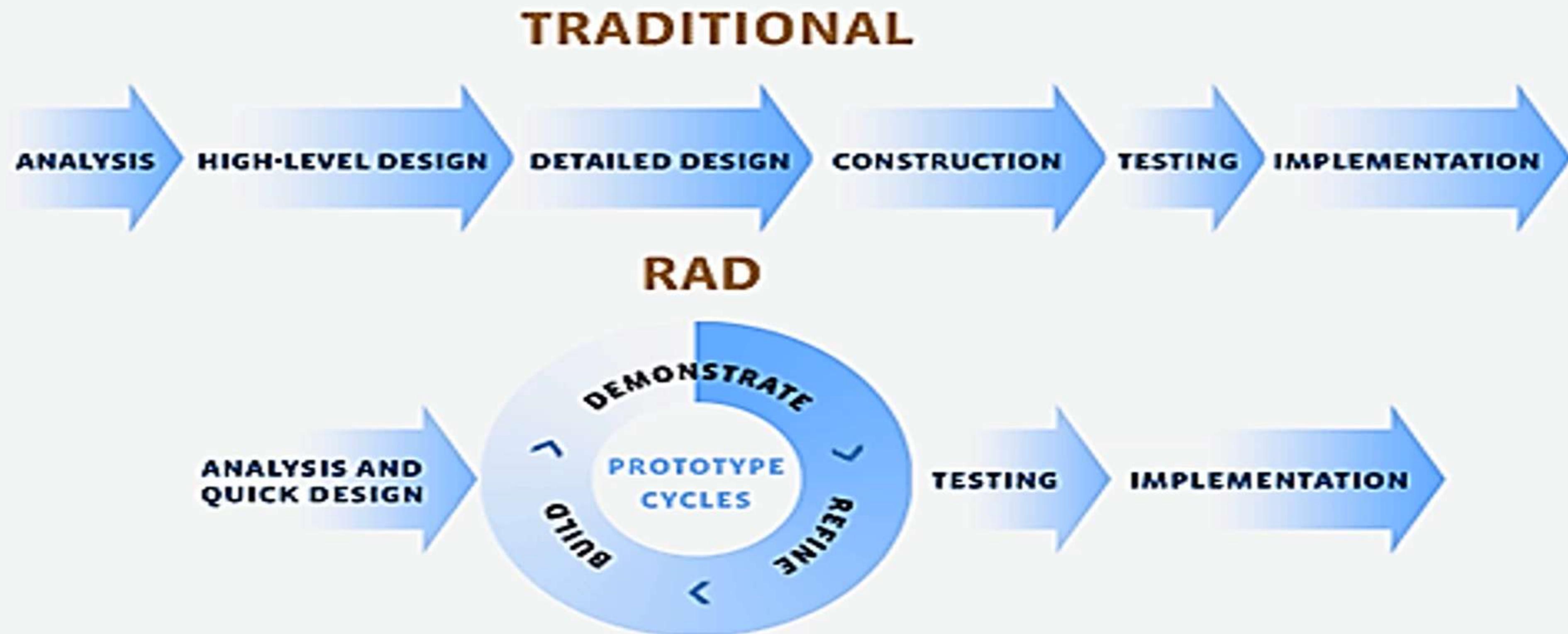
RAD is a high speed version of the linear sequential model

Characterized by a very short development life cycle

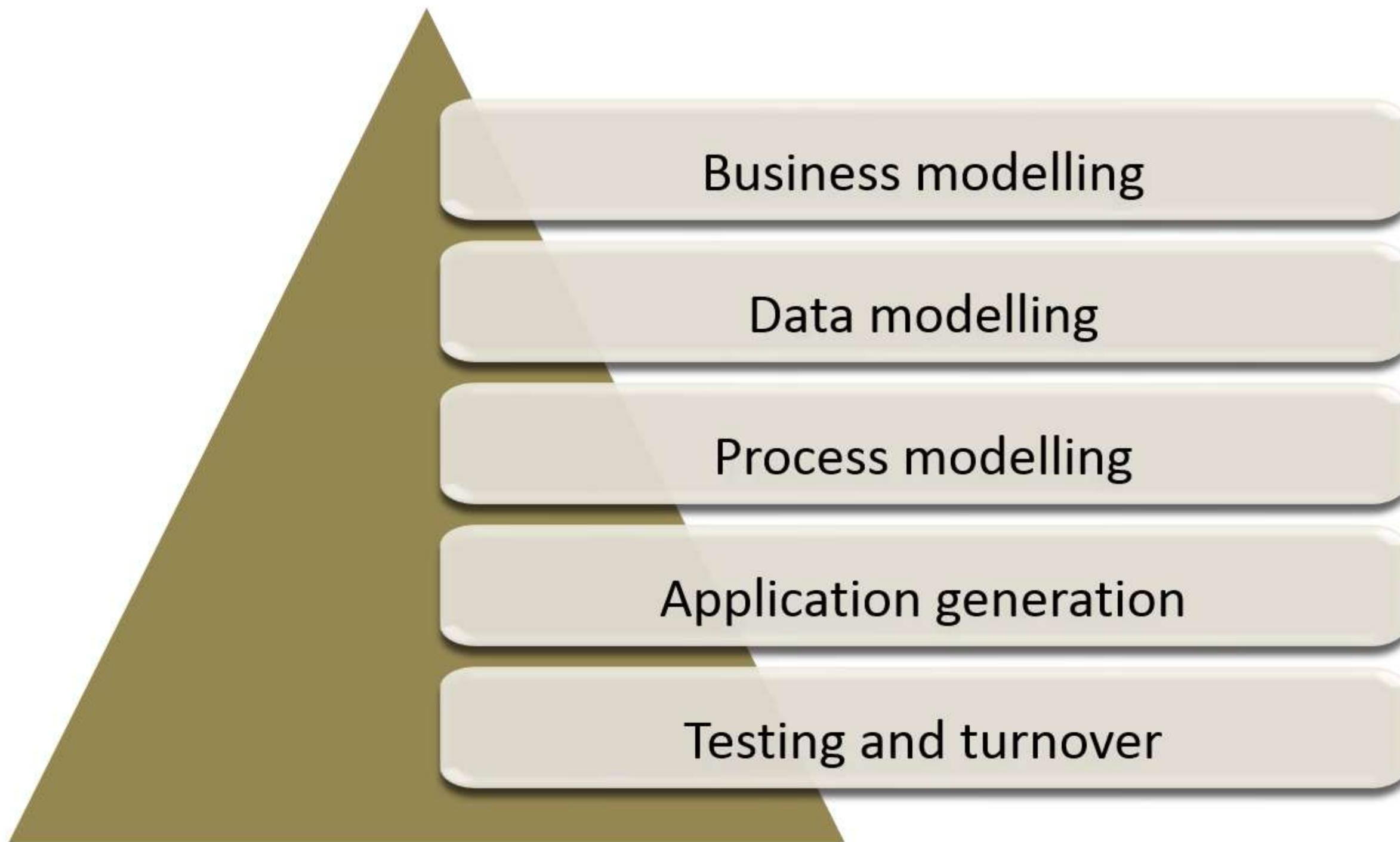
The RAD model follows a component based approach

Individual components are developed by different people and assembled to develop a large software system

Traditional vs RAD



Rapid Application Development phases



Advantages and Limitations of RAD

Advantages:

- Due to the emphasis on rapid development, it results in the delivery of a fully functional project in short period.
- Facilitates Parallel Development.

Limitations:

- Developers and clients must be committed to rapid-fire activities in an abbreviated time frame.
- If either party is indifferent in needs of other, the project will run into serious problem.
- It is not suitable for large projects.

When to use RAD?

When requirements are not
fully understood

User is involved throughout
the life cycle

System can be modularized

Incremental Model

The incremental model prioritizes requirements of the system and implements them in groups

Each subsequent release of the system adds function to the previous release, until all designed functionalities have been implemented

Advantages of Incremental Model

- 
- Uses "divide and conquer" breakdown of tasks
 - High-risk or major functions are addressed in the first increment cycles
 - Each release delivers an operational product
 - Customer can respond to each build
 - Customers get important functionality early

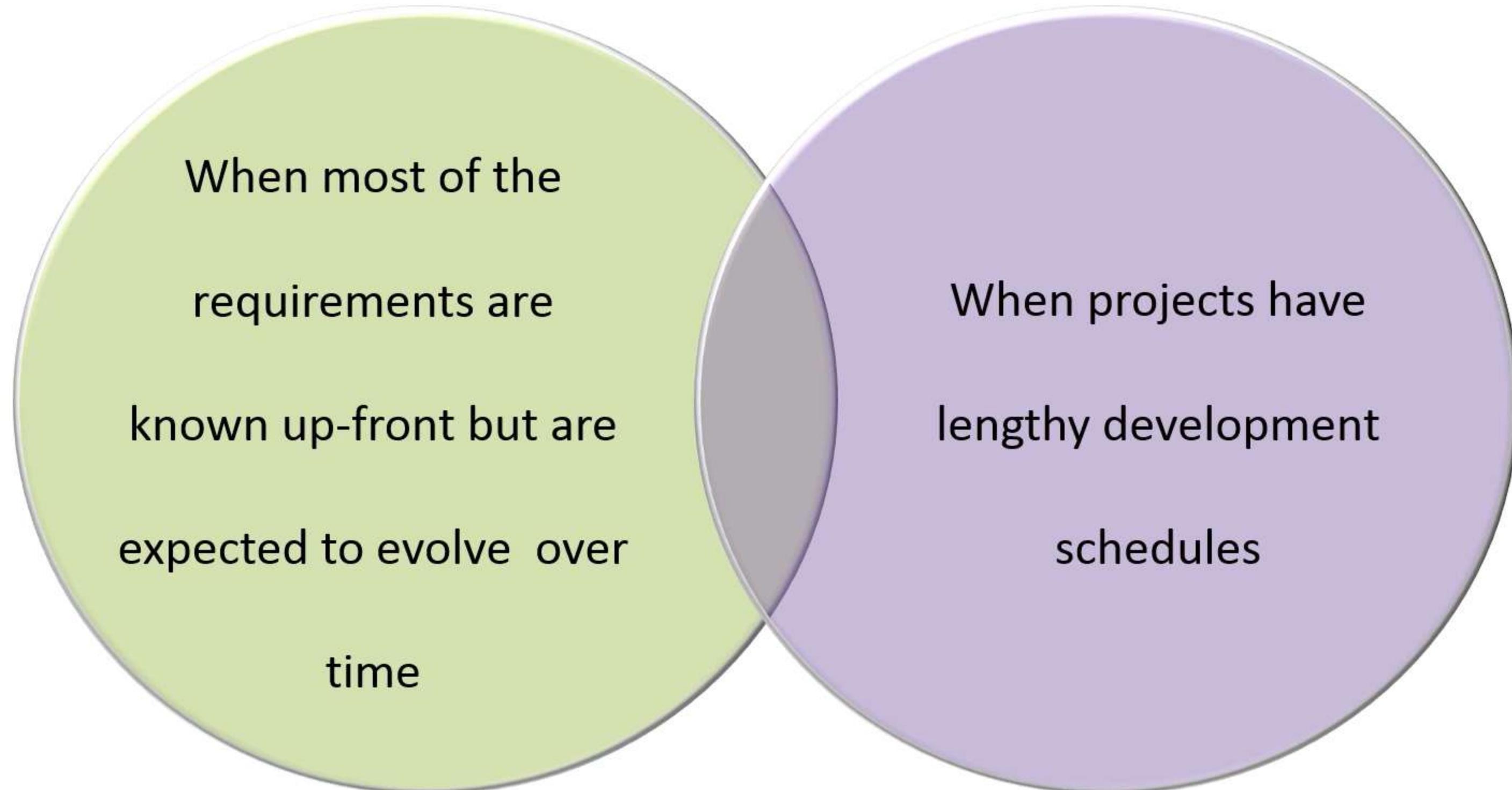
Limitations of Incremental Model

Requires early definition of a complete and fully functional system to allow for the definition of increments

Requires good planning and design as basis for the system

Absence of a well-defined module interface is a major obstacle for this model of development

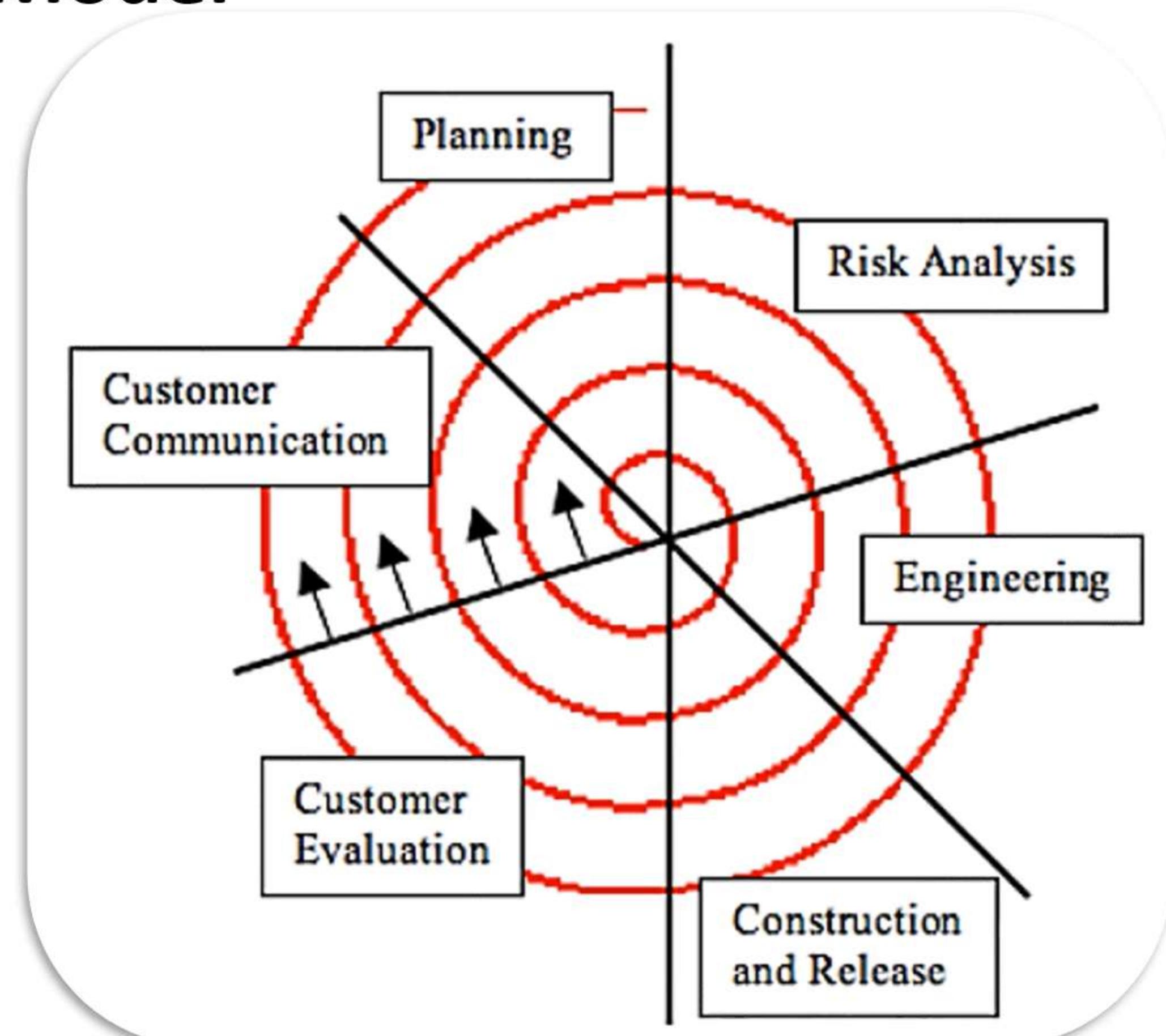
When to use Incremental Model?



Spiral Model

- Proposed by Barry Boehm in 1986
- Diagrammatic representation of this model appears like a spiral with many loops
- Suitable for technically challenging software products that are prone to several kinds of risks
- Accommodates prototyping. This model combines the features of the prototyping model and the waterfall model
- It is favoured for large, expensive, and complicated models
- Suggested for High-Risk Scenarios based projects

Spiral Model



Advantages of Spiral Model

Provides early indication risk.

Users see the system early because of rapid prototyping tools.

Critical high-risk functions are developed first.

Early and frequent feedback from users.

Limitations of Spiral Model

Time spent for evaluating risks are too large for small or low-risk projects and may not prove cost-worthy.

Time spent on planning, resetting objectives, doing risk analysis and prototyping may be excessive.

Relies on Risk assessment expertise.

When to use Spiral Model?

Risk
perceived is
very high

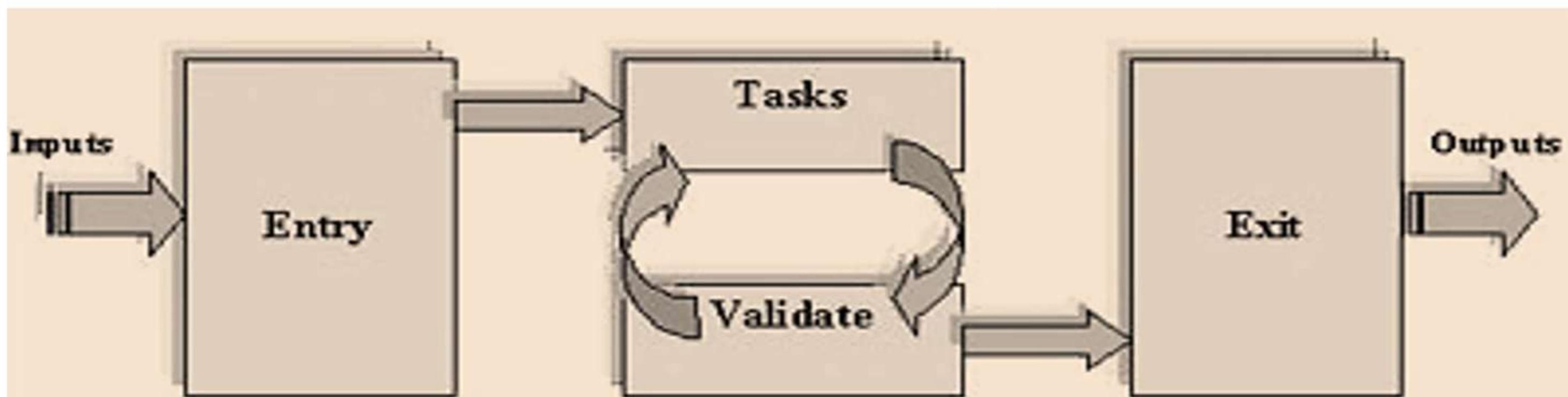
Requirements
are complex

Significant
changes are
expected

Process Representation Technique – ETVX Model

IBM introduced the ETVX model to document their process.

Each phase in a process performs a well-defined task and generally produces an output.



ETVX Model



Entry

Task

Validation

Exit

ETVX Model Example

Entry

- Hall Ticket

Task

- Show Hall Ticket
- Get Question Paper
- Get Answer Sheet
- Write Answer for Respective questions

Verification

- Verify whether questions are written for appropriate questions
- Review the answers written

Exit

- Submission of the answer sheet to the invigilator

ETVX Model Example for Analysis Phase

Entry

- Feasibility Report

Task

- Collect requirement
- Analyze requirement
- Write SRS

Verification

- Review SRS

Exit

- SRS

Summary

- Process
- Software Development Life Cycle
- Waterfall model
- V model
- Prototype model
- RAD
- Spiral Model
- ETVX



Press **Esc** to exit full screen

REQUIREMENT ANALYSIS



Objectives

In this module you will learn,

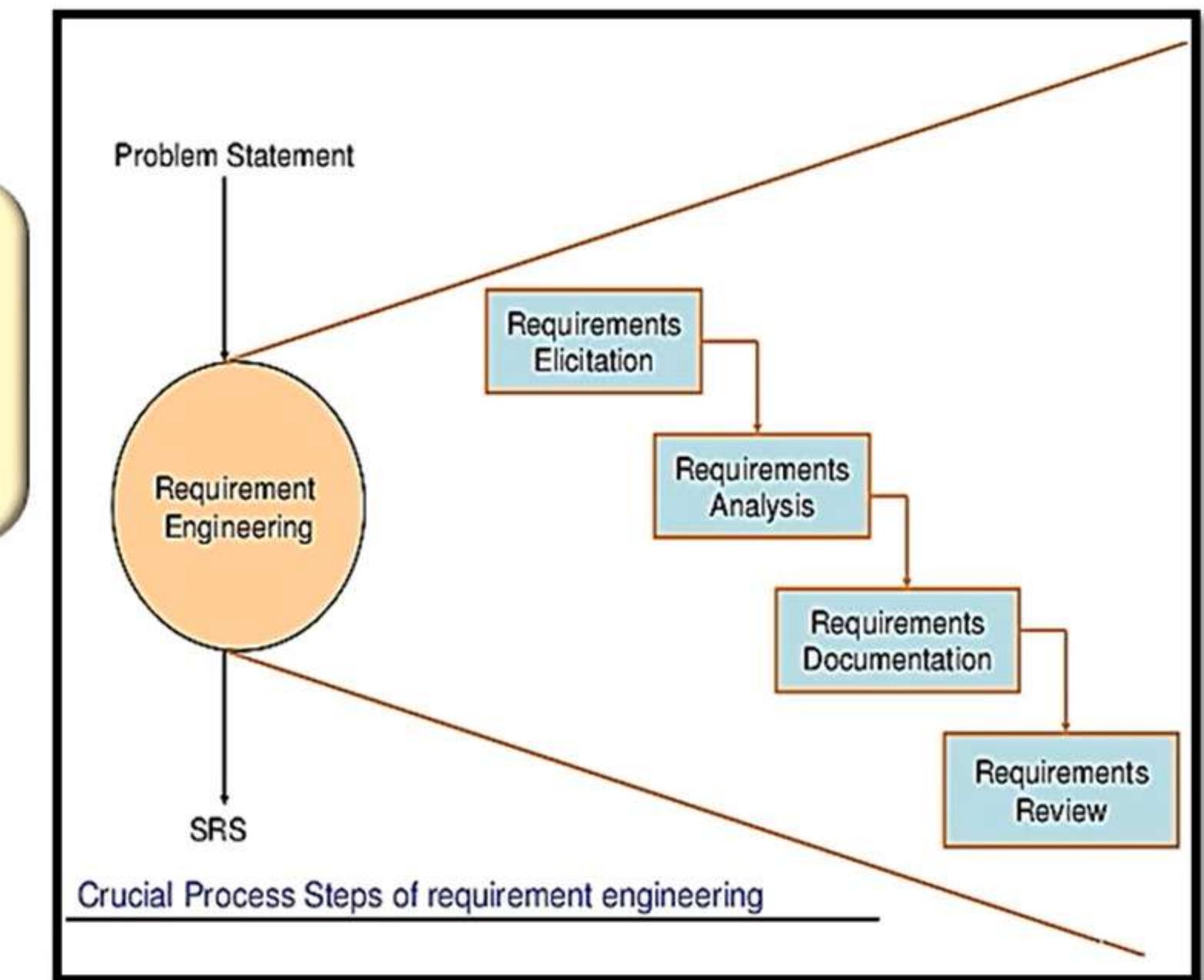
- Requirements Engineering
- SRS
- Properties of SRS
- Functional Requirements
- Non Functional Requirements
- Entity Relationship diagram and its components



Requirement Engineering



Here you go Tom! You have the requirements for a Matrimonial website in hand. This is the problem statement.



Requirement Elicitation

Elicit requirements from all stakeholders

Address problems of scope

Address problems of understanding

Address problems of volatility (changing requirements)

Requirement Analysis

The process of establishing the services the system should provide and the constraints under which it must operate.

The process of studying and analyzing the customer and the user/stakeholder to arrive at a definition of software requirements.

Requirement Analysis

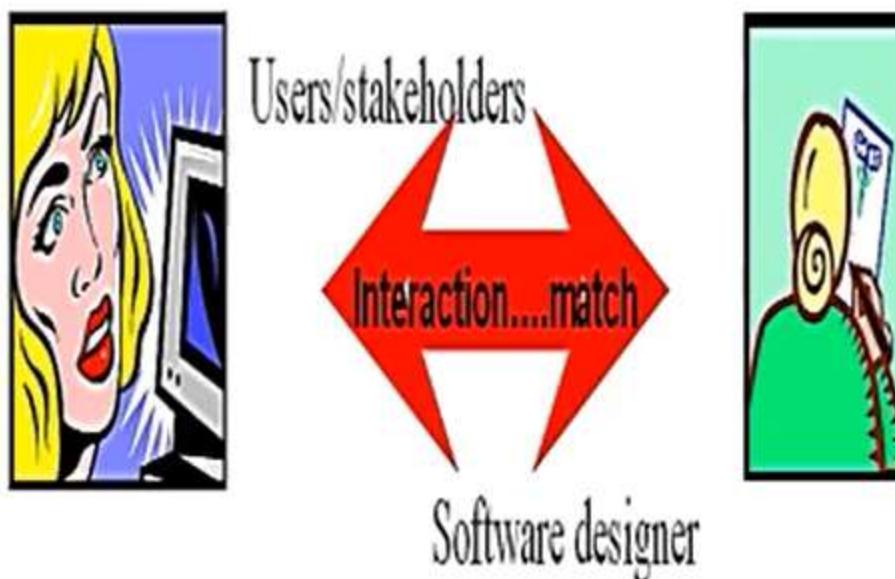
Why is Requirement Analysis difficult?

Different “worlds”

- Bridging the gap between the client and the software developer
- Knowing what should be done VS knowing what to let a computer do

Users/stakeholders are not a uniform group

- conflict between cost and usability / performance / features
- conflicting demands from different departments



Getting the good (ideal) system
Vs
possibility of building it well

Requirement Analysis

Other factors

- Expectations, the final solution is difficult to imagine by the users
- Scope of the system
 - needs well defined boundaries
- Current vs. future system
 - resistance to change
- Process of negotiation between users and designers

Requirement Analysis

Goals of requirement Analysis and specification phase

- Understand the user's requirements
- Remove inconsistencies, anomalies, etc. from requirements
- Document requirements properly in an SRS document

Requirement Analysis

A person who performs requirement analysis is called a system analyst

- Understands User requirement
- Collects data needed for the user requirement
- Writes the software requirement specification

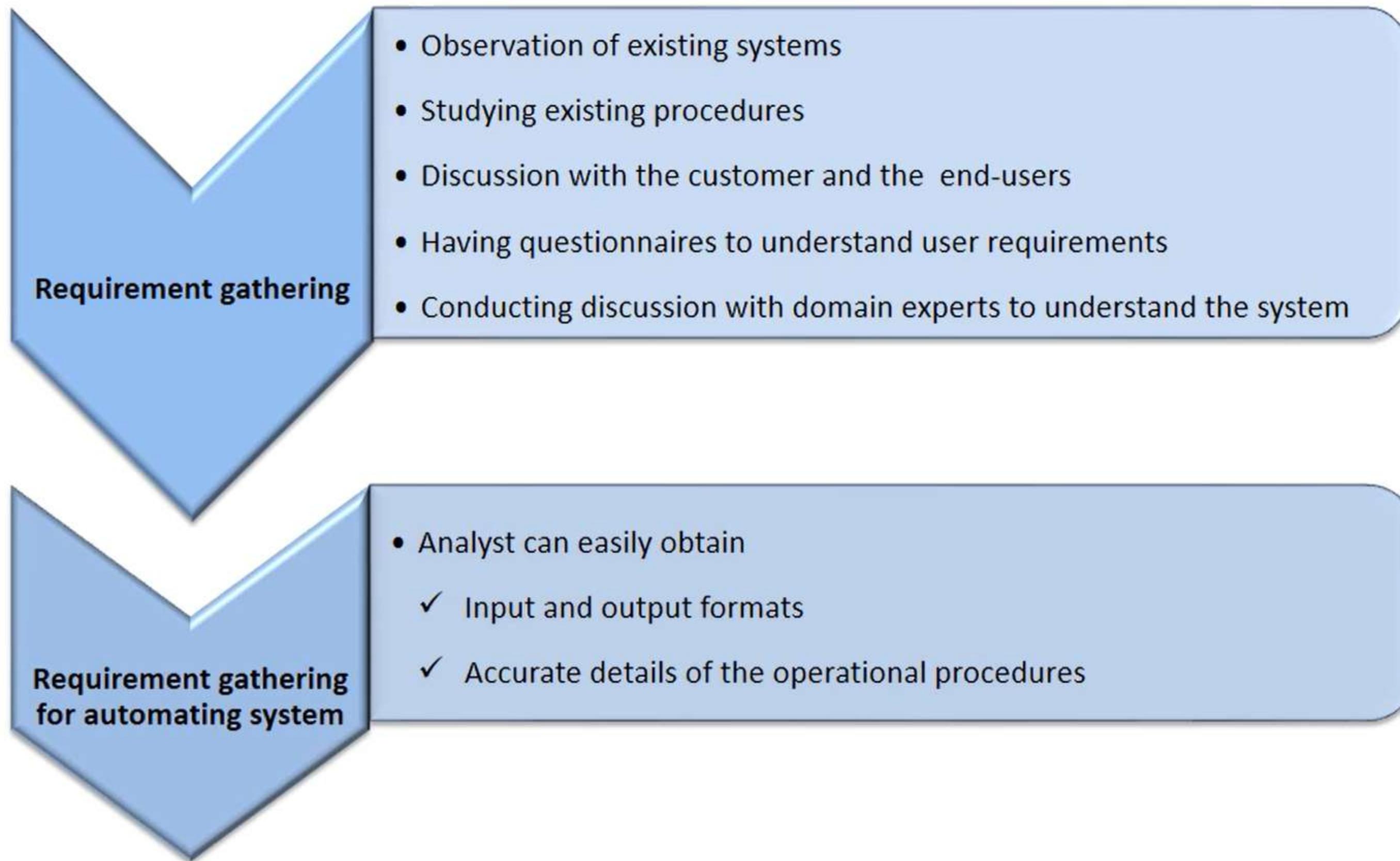
Who is a
system
analyst??

Requirements analysis consists of two main activities:

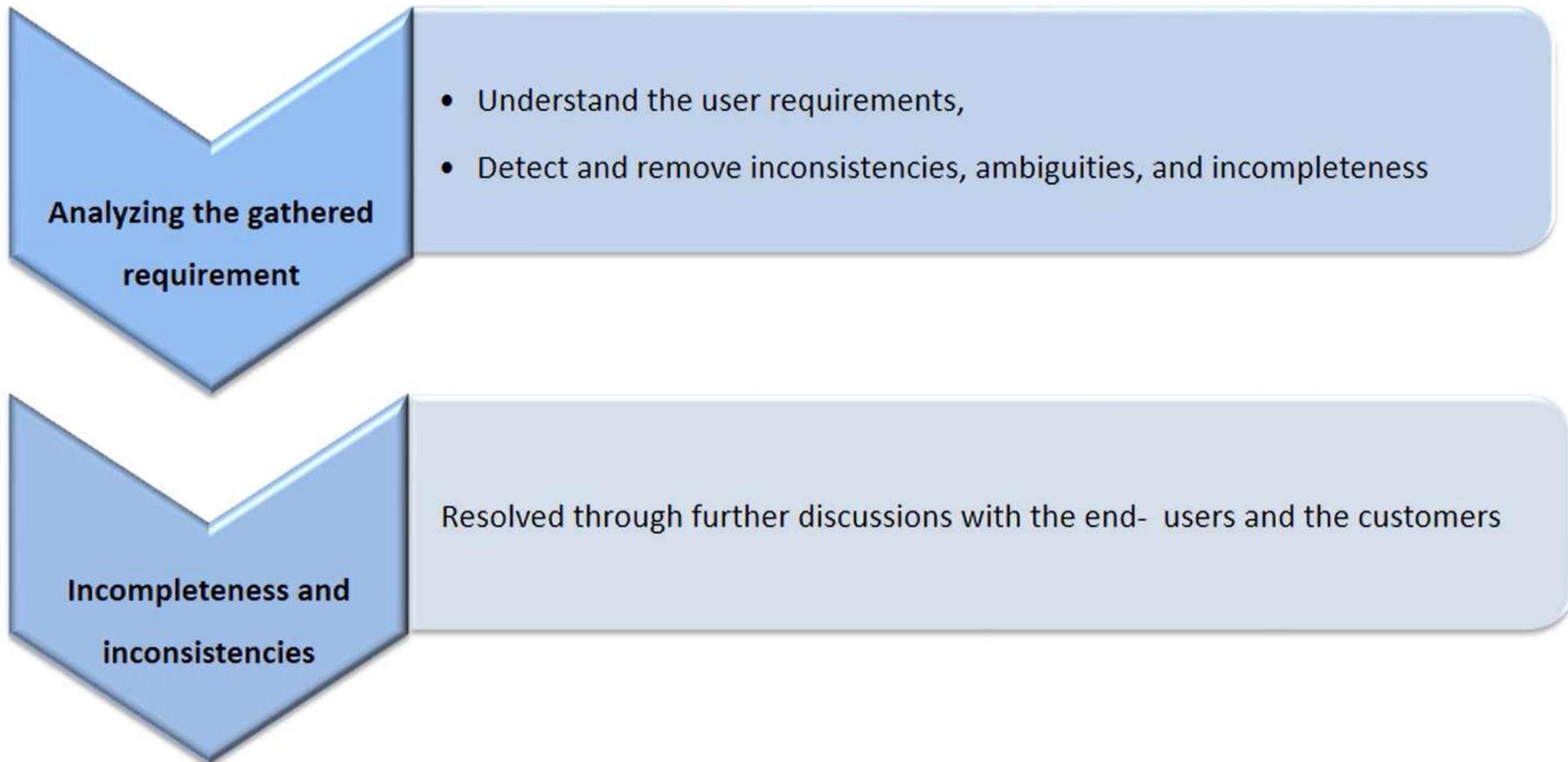
- Requirements gathering
- Analysis of the gathered requirements



Requirement Analysis



Requirement Analysis



Incomplete Requirement

Some requirements have been omitted:

- Due to oversight.

Example:

- A solar heater specifies what it will do on a 'sunny day' but it fails to specify what it will do on a 'rainy day'
- The analyst has not recorded when temperature falls below 90 degrees.
 - heater should be turned ON
 - water shower turned OFF.

Requirement Analysis

Several things about the project should be clearly understood by the analyst

- What is the problem?
- Why is it important to solve the problem?
- What are the possible solutions to the problem?
- What complexities might arise while solving the problem?

Software Requirements Specification



Outcome of the analysis phase

Contains all the information about the requirements gathered

Main aim of requirements specification:

- Systematically organizes the requirements arrived during requirements analysis
- Documents requirements properly.

SRS

SRS document concentrates on

- What needs to be done
- Carefully avoids the solution (“how to do”) aspects.

The SRS document is useful in various contexts

- Statement of user needs
- Contract document
- Reference document
- Definition for implementation

Properties of a Good SRS

Concise

Specify what the system must do

Easy to change

Consistent

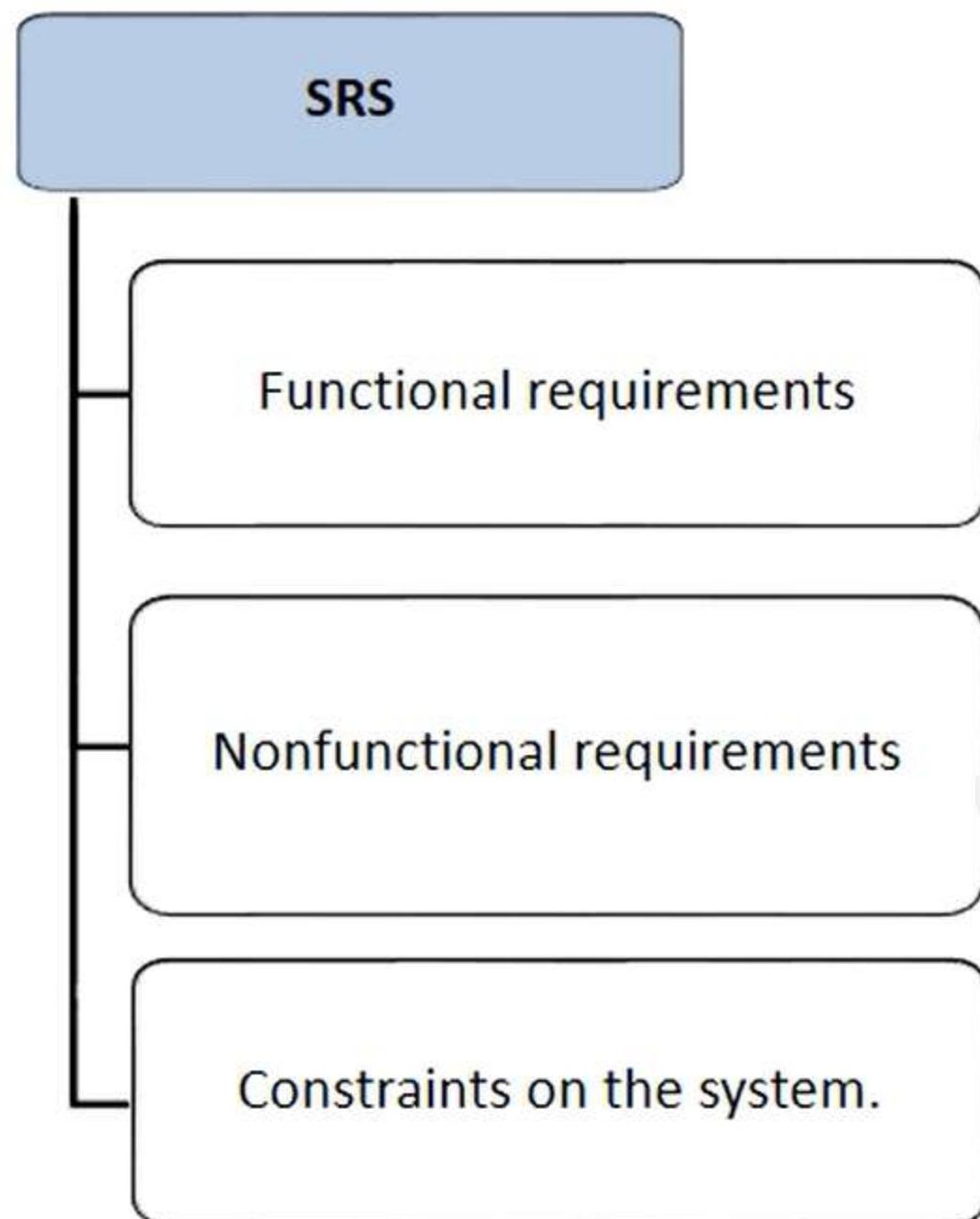
Complete

Traceable

Verifiable

SRS Document

SRS document normally contains three important parts



Functional Requirement

Functional requirements describe

- A set of high-level requirements
- Each high-level requirement
- Takes in some data from the user
- Outputs some data to the user
- Might consist of a set of identifiable functions which process the input

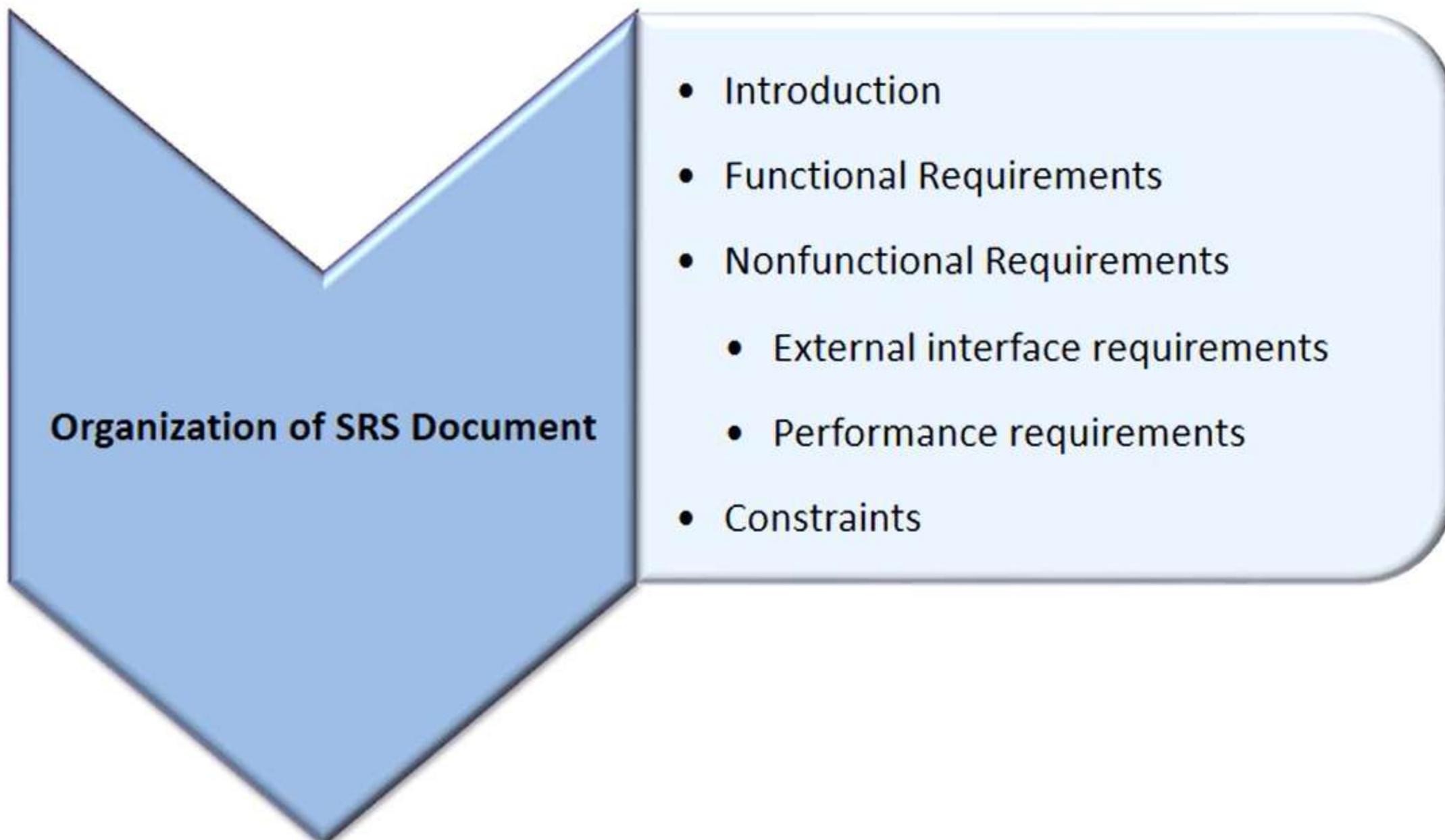
Constraints

Constraints describe things that the system should or should not do.

For example,

- Standards compliance
- How quickly the system can produce results so that it does not overload the other system to which it supplies data, etc.
- Hardware to be used
- Operating system
- DBMS to be used
- Capabilities of I/O devices
- Data representation

SRS Document



SRS Document Structure

1. Introduction	1
1.1 Purpose.....	1
1.2 Document Conventions.....	1
1.3 Intended Audience and Reading Suggestions.....	1
1.4 Project Scope	1
1.5 References.....	1
2. Overall Description	2
2.1 Product Perspective.....	2
2.2 Product Features.....	2
2.3 User Classes and Characteristics.....	2
2.4 Operating Environment.....	2
2.5 Design and Implementation Constraints	2
2.6 User Documentation	2
2.7 Assumptions and Dependencies.....	3
3. System Features.....	3
3.1 System Feature 1	3
3.2 System Feature 2 (and so on).....	4
4. External Interface Requirements.....	4
4.1 User Interfaces.....	4
4.2 Hardware Interfaces.....	4
4.3 Software Interfaces.....	4
4.4 Communications Interfaces	4
5. Other Nonfunctional Requirements	5
5.1 Performance Requirements.....	5
5.2 Safety Requirements.....	5
5.3 Security Requirements.....	5
5.4 Software Quality Attributes.....	5
6. Other Requirements.....	5
Appendix A: Glossary.....	5
Appendix B: Analysis Models	6
Appendix C: Issues List	6

ERD

Entity Relationship Diagram

ER diagram is widely used in database design

- Represents conceptual level of a database system
- Describes things and their relationships in high level

ERD

Entity

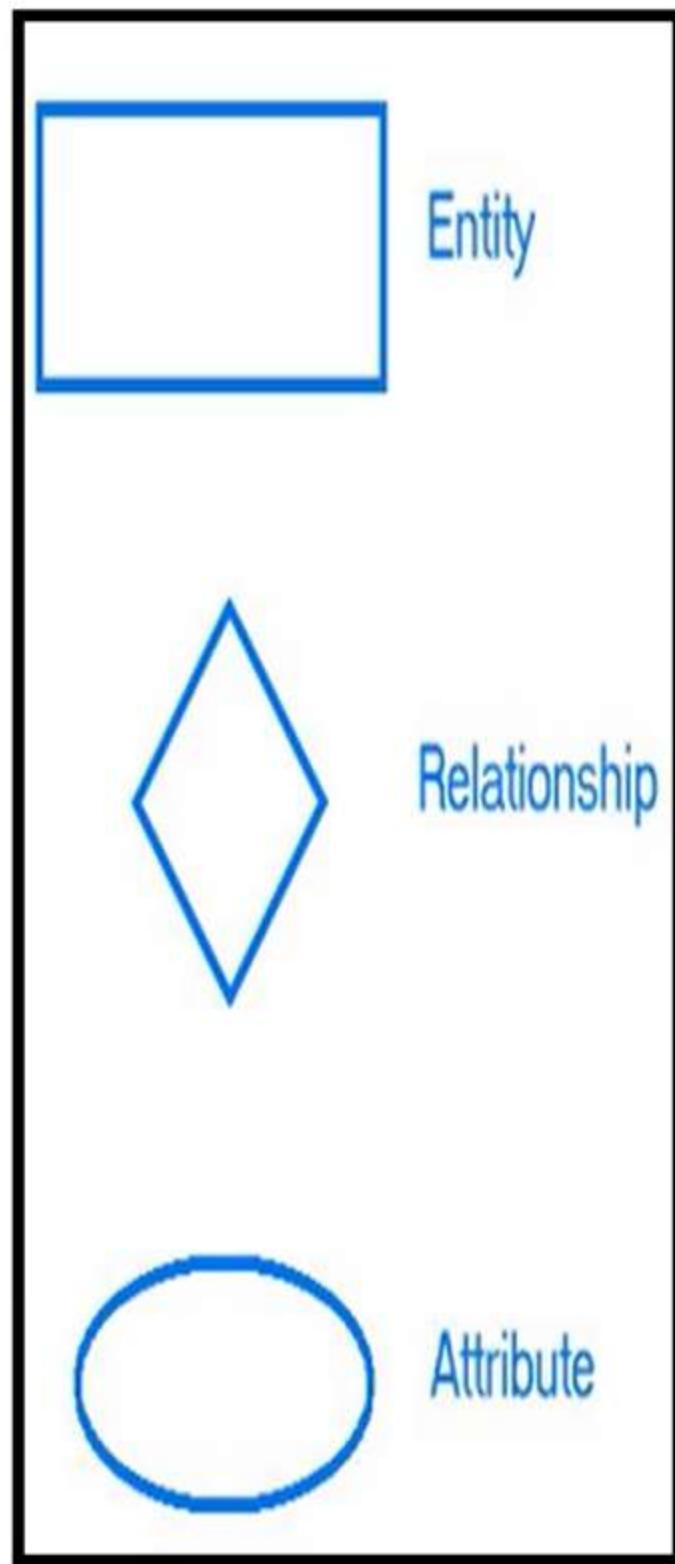
- An entity is a business object that represents a group, or a category of data.

Attribute

- Properties of an entity.

Relationship

- Specifies the relations among entities



ERD

- Relationship specifies association between two entities.

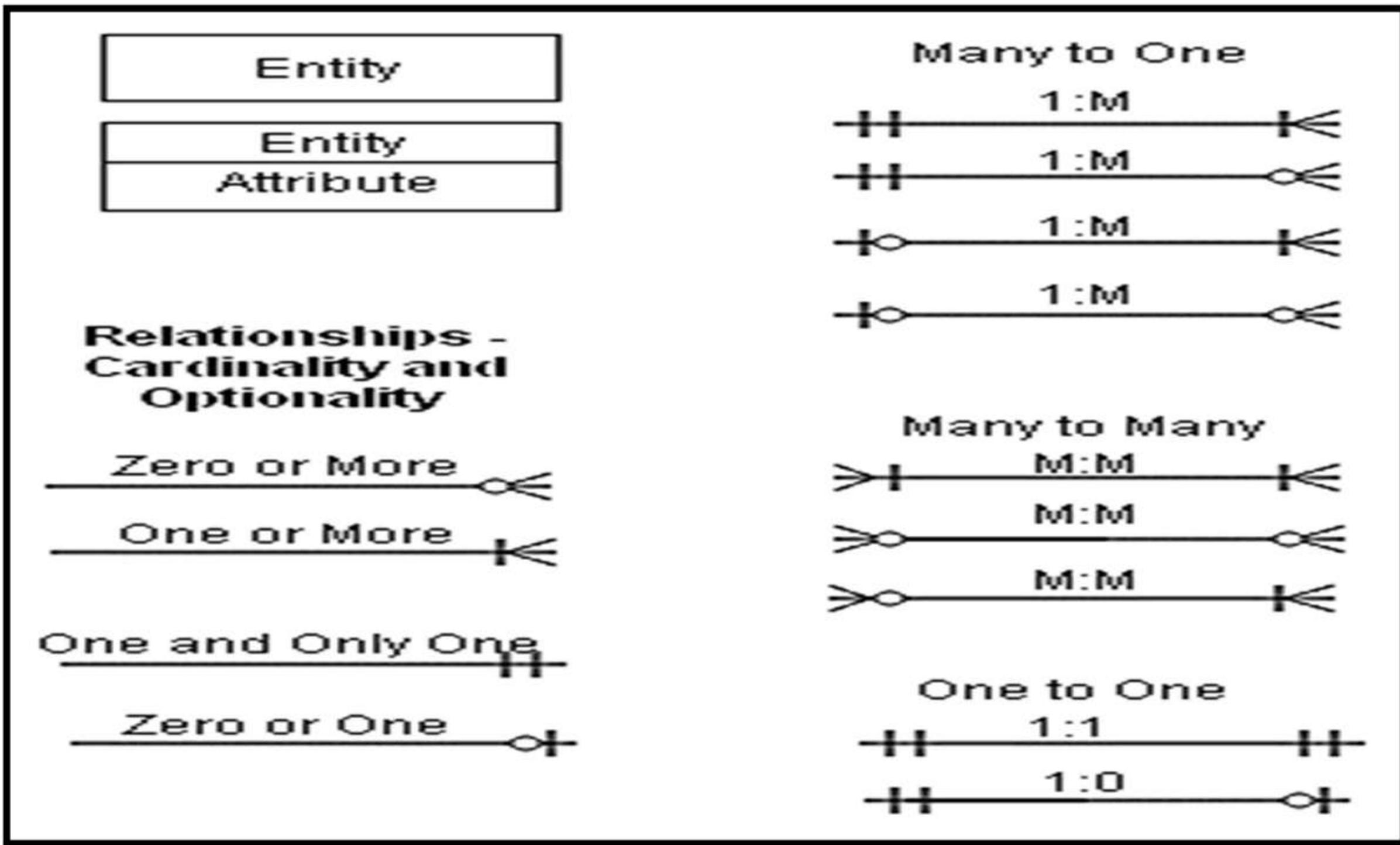
Cardinality

- One instance of an entity maps to how many instances of other entity
- Many-to-Many Relationships
- One-to-Many Relationships
- One-to-One Relationships
- Recursive Relationships

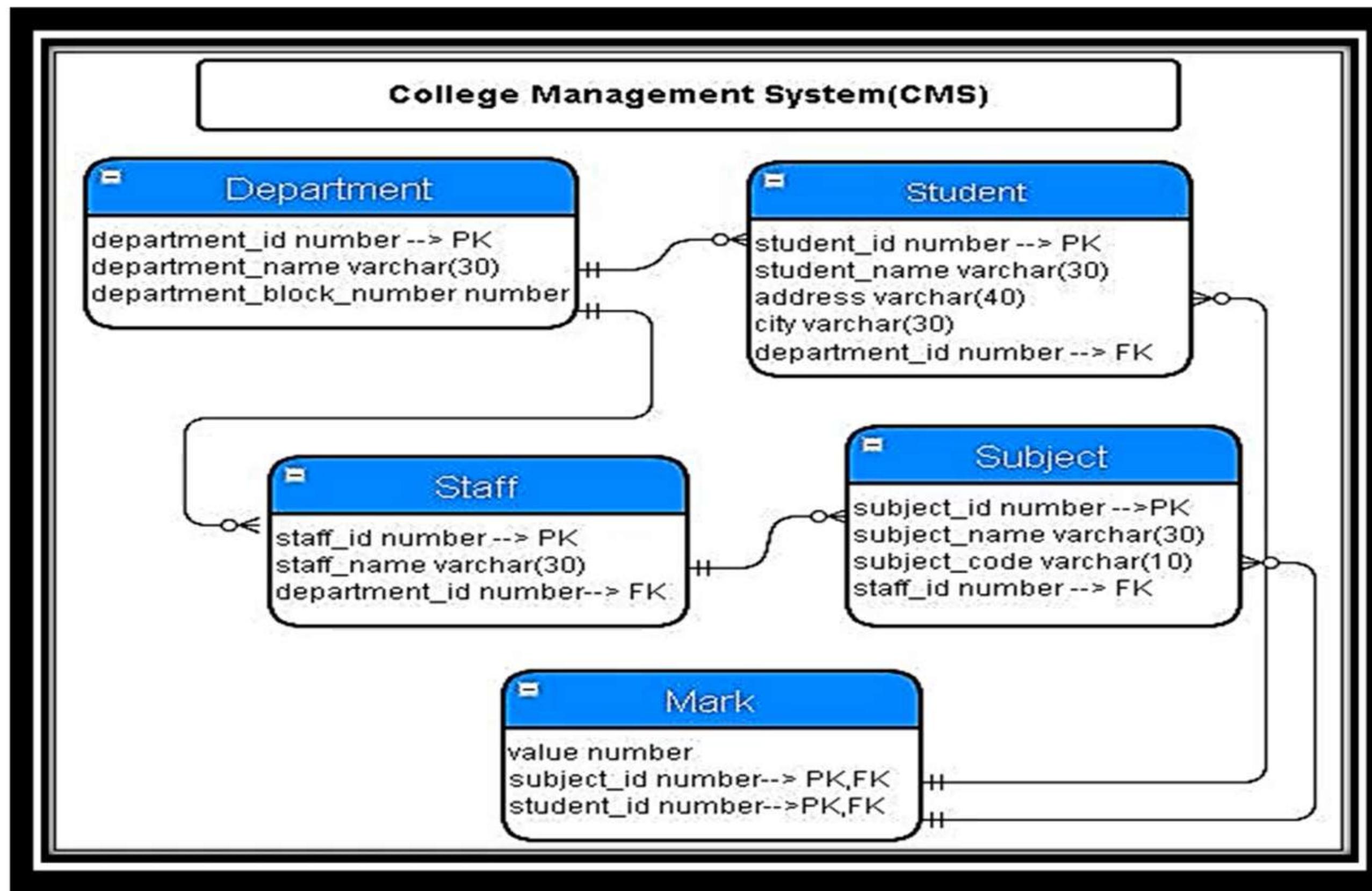
Optionality

- Is the relationship mandatory or optional
- Mandatory Relationships
- Optional Relationships

ERD Example



ERD Example



Summary

- Requirements Engineering
- SRS
- Properties of SRS
- Functional Requirements
- Non Functional Requirements
- Entity Relationship diagram and its components



SOFTWARE TESTING



Objectives

In this module you will learn,

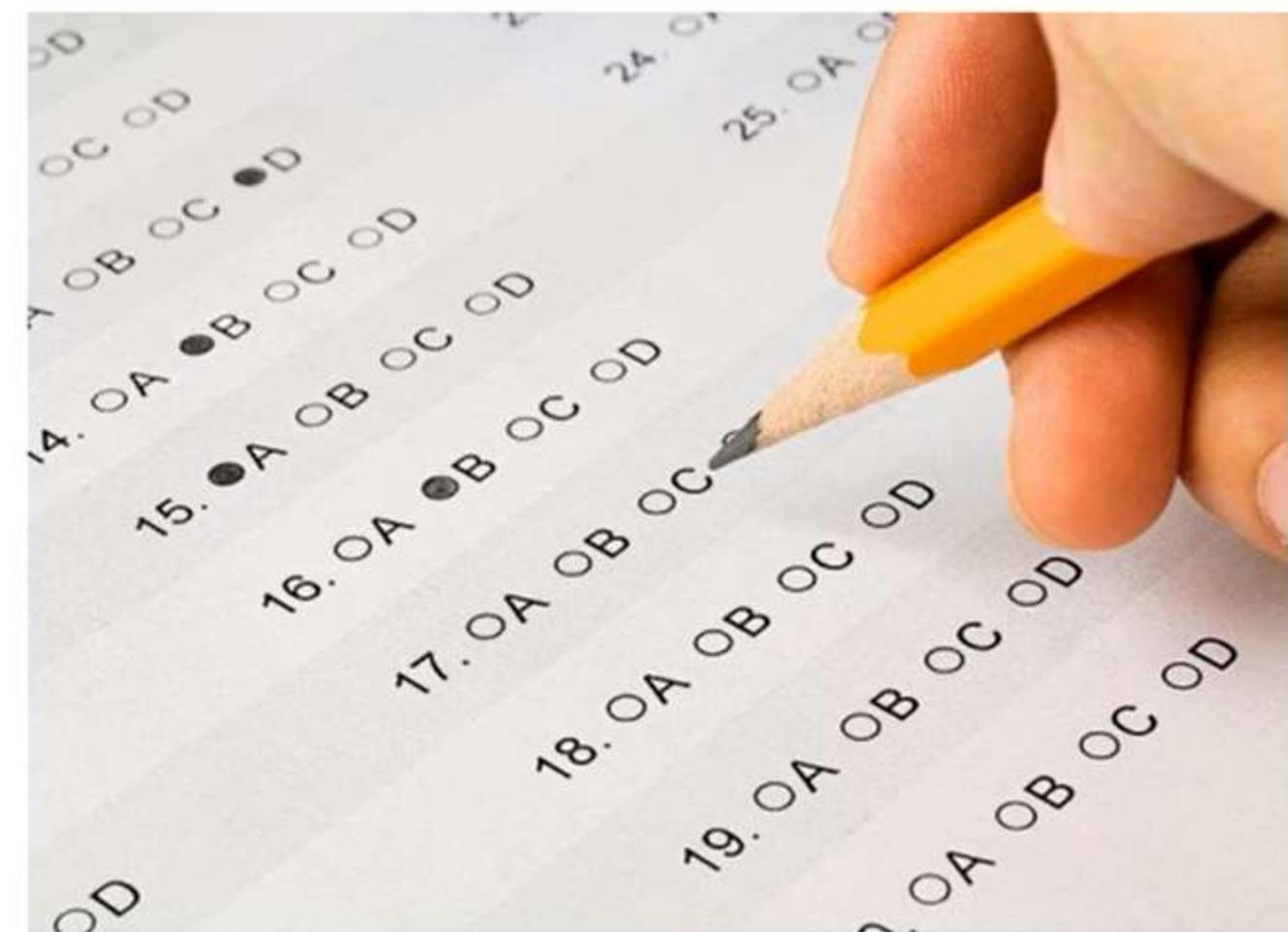
- What is Testing
- Importance of Testing
- Levels of Testing
- Types of Testing
- Debugging



Software Testing

Process of executing the program with the intent of finding errors

Process of verifying and validating so that a software application or program meets the business and technical requirements that guide its design and development work as expected.



Fundamentals of Testing

Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements, or to identify differences between expected and actual results.

Testing can be used to show the presence of defects, but never their absence!



Importance of Testing

Most cost effective approach to build confidence in most of the software systems

Detects the faults in the software

Improves the quality and reliability by removing the faults from the software

Testing Profession

The testing profession is not about just creating test related documents.

- It is understanding the application functionality and coding
- Testers are the ones who are the certifying authority for the software itself
- Testers destruct the software to give better solutions

Who will test a Software ?
What makes one a good
Tester?

Who performs testing?

- Programmers and testers

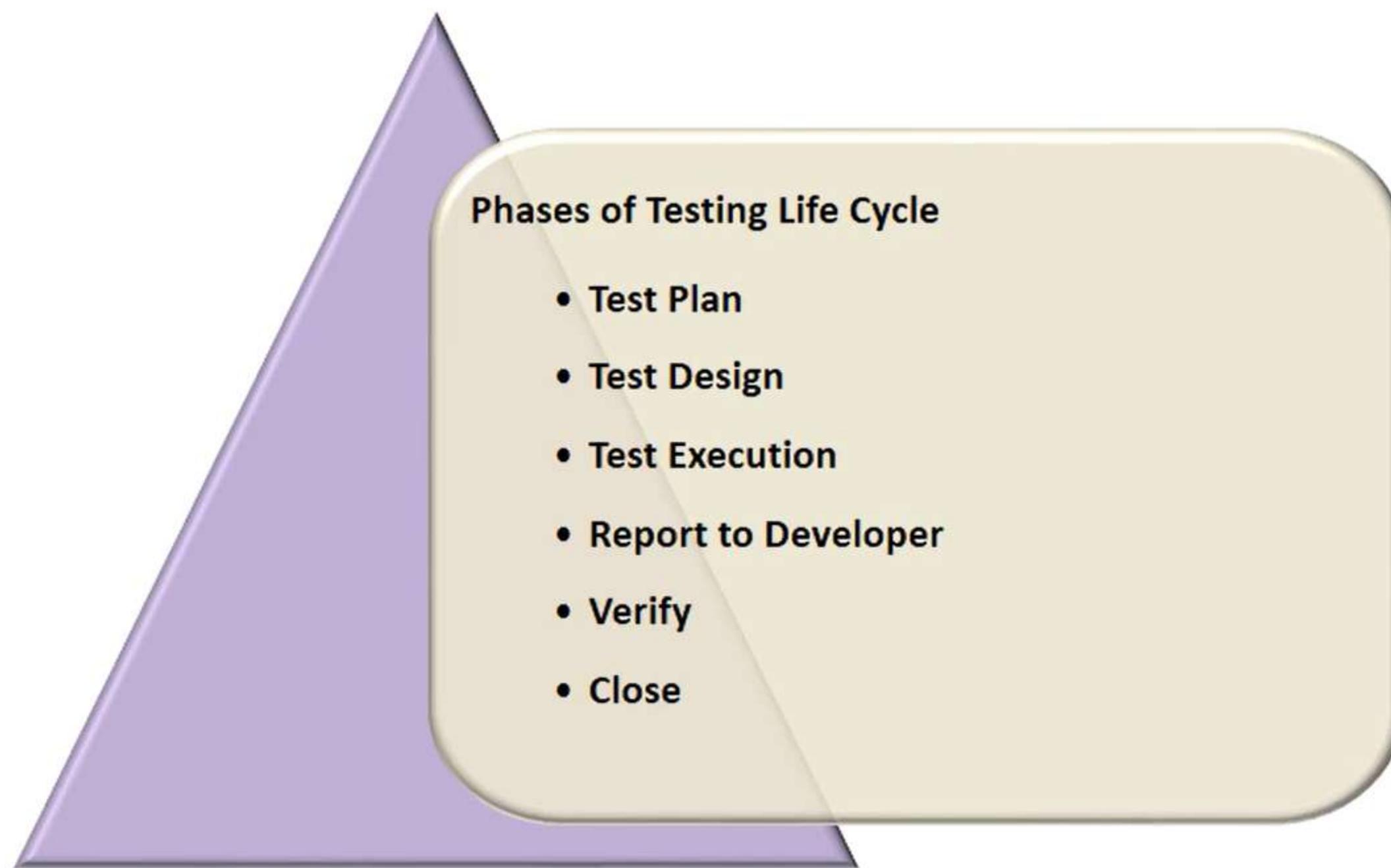
What should a good tester possess?

- Creative thinking
- Questioning skills
- Never give up attitude

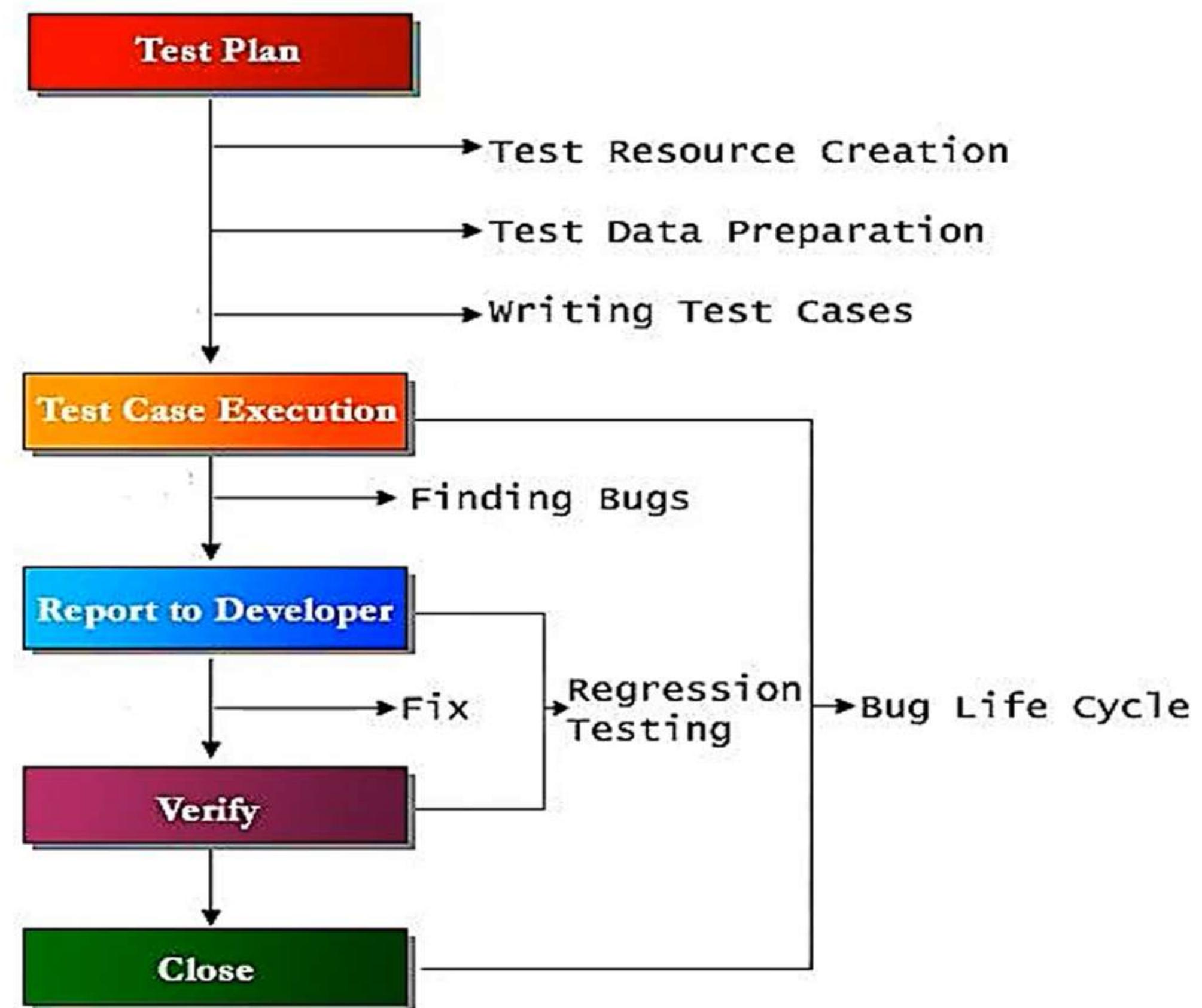


Testing Life Cycle

- Software testing life cycle identifies what test activities to carry out and when (what is the best time) to accomplish those test activities



Testing Life Cycle



Methodology of Testing

Manual Testing

Automated Testing

- Example
 - Java – JUNIT
 - Dot Net – N UNIT

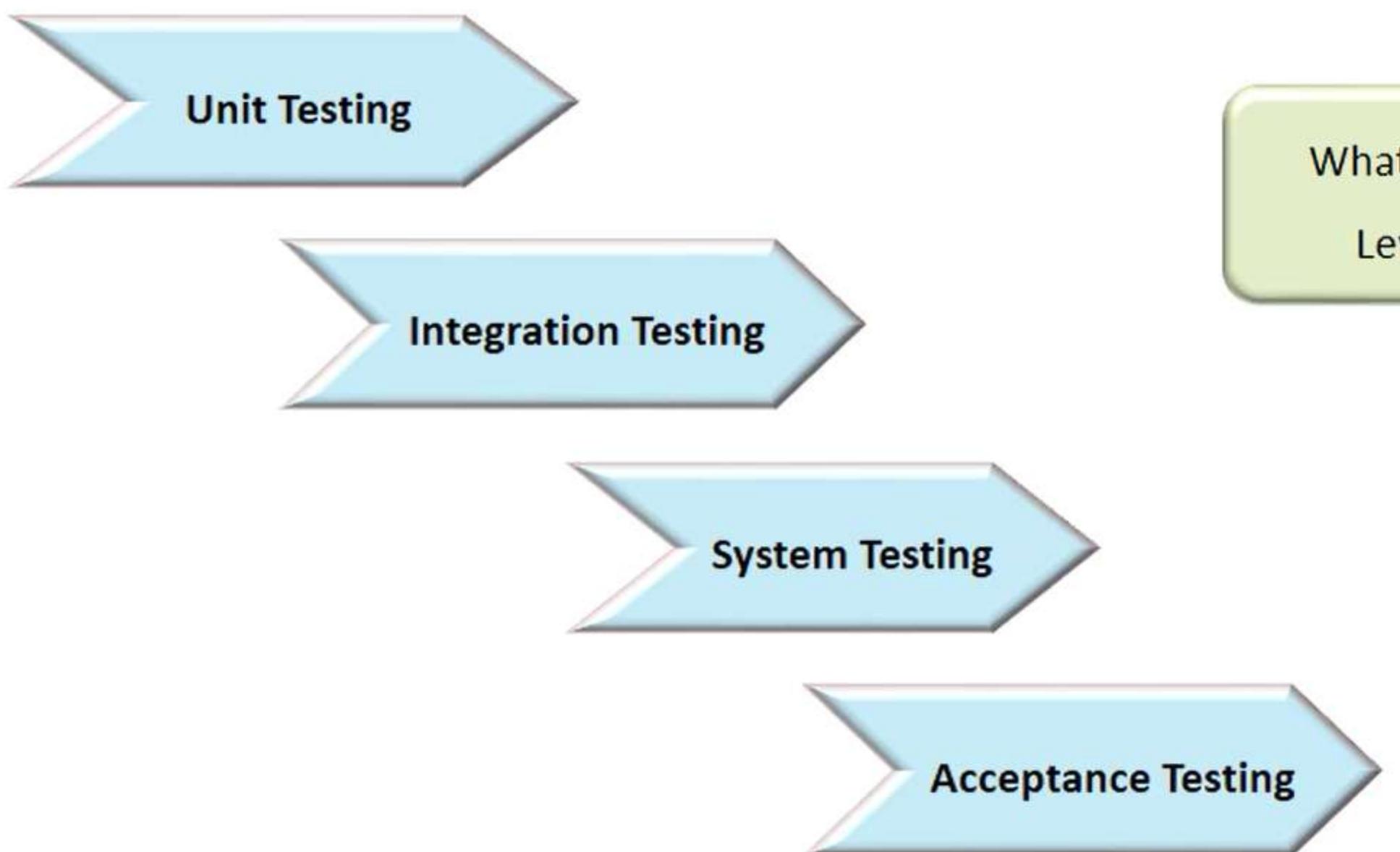
Bugzilla

- Defect Tracking Tool

What are the different methods
of Testing?



Levels of Testing



What are the different
Levels of Testing?



Unit Testing

Unit Testing is used to check whether a particular module is implementing its specification

The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.

Who performs testing?

- normally performed by software developers themselves or their peers
- In rare cases it may also be performed by independent software testers



Integration Testing



Integration Testing is a level of the software testing process where individual units are combined and tested as a group.

The purpose of Integration Testing is to expose faults in the interaction between integrated units.

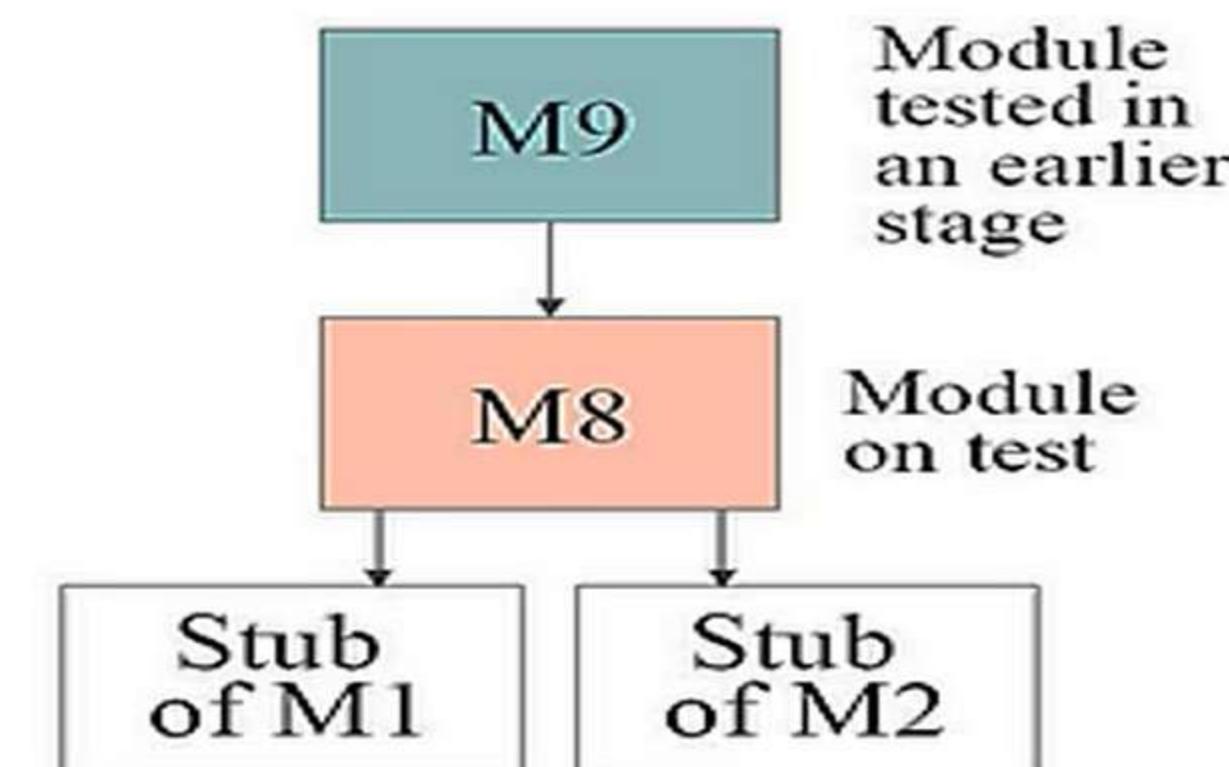
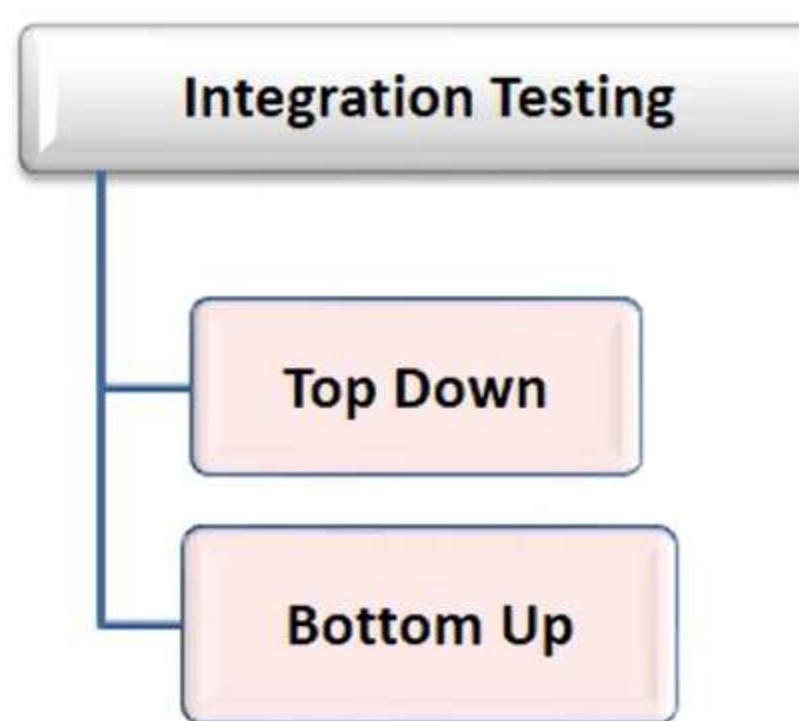
Who performs this testing?

- Either Developers themselves or independent Testers

Integration Testing Approaches

Top Down

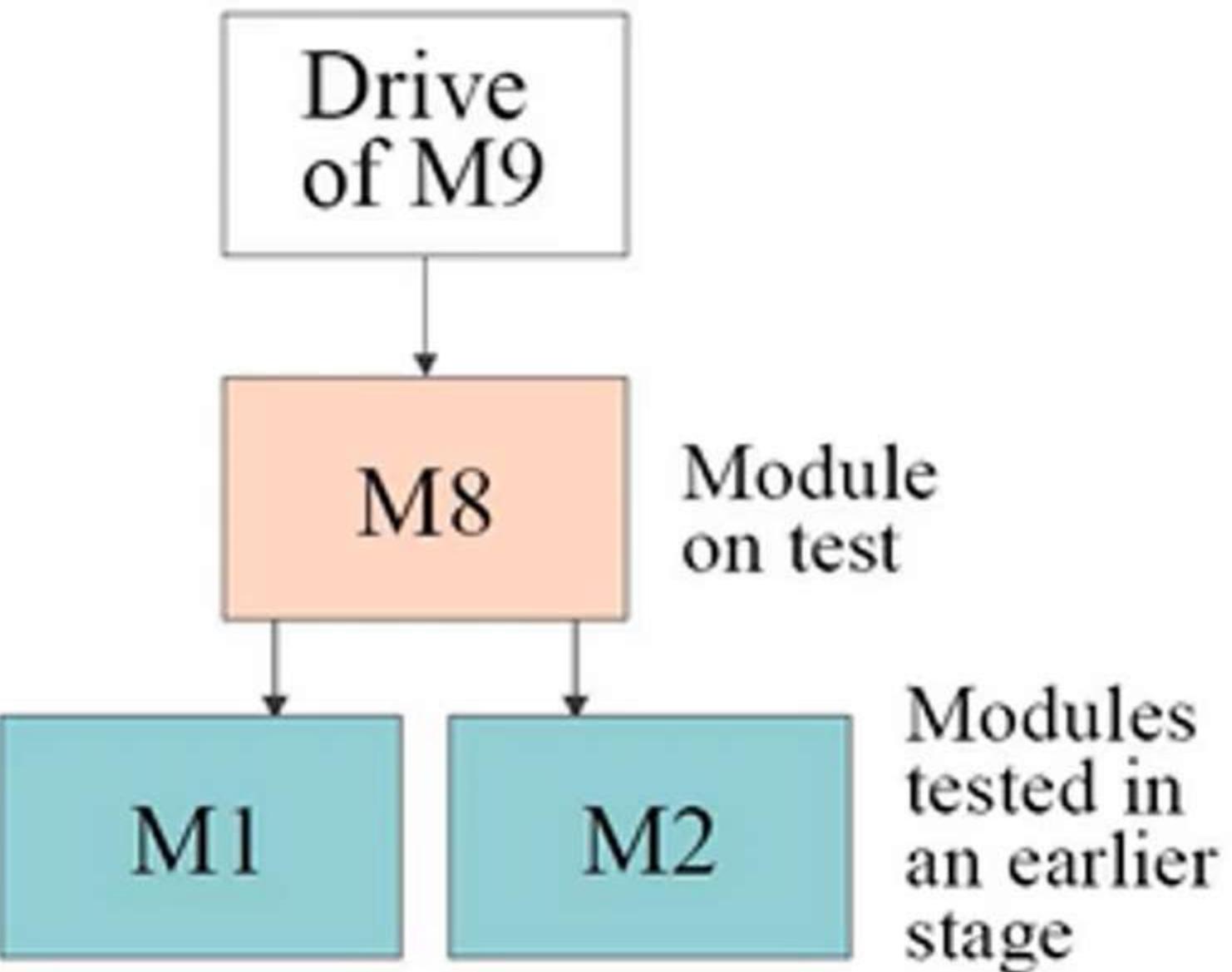
- Top level units are tested first and lower level units are tested step by step after that
- Test Stubs are needed to simulate lower level units which may not be available during the initial phases



Integration Testing Approaches

Bottom Up

- Bottom level units are tested first and upper level units step by step after that
- Test Drivers are needed to simulate higher level units which may not be available during the initial phases



System Testing



System testing finds disparities between implementation and specification

Performed by:

Testers

System testing involves

Functional testing - Test the implementation of the business needs

Performance testing - It will test all the non functional requirements of the system specified in the specification

Performance Testing - Types

Stress tests

- evaluates the system when stressed to its limits

Regression tests

- required when the system being tested is replacing an existing system

Usability test

- testing characteristics related to user friendliness

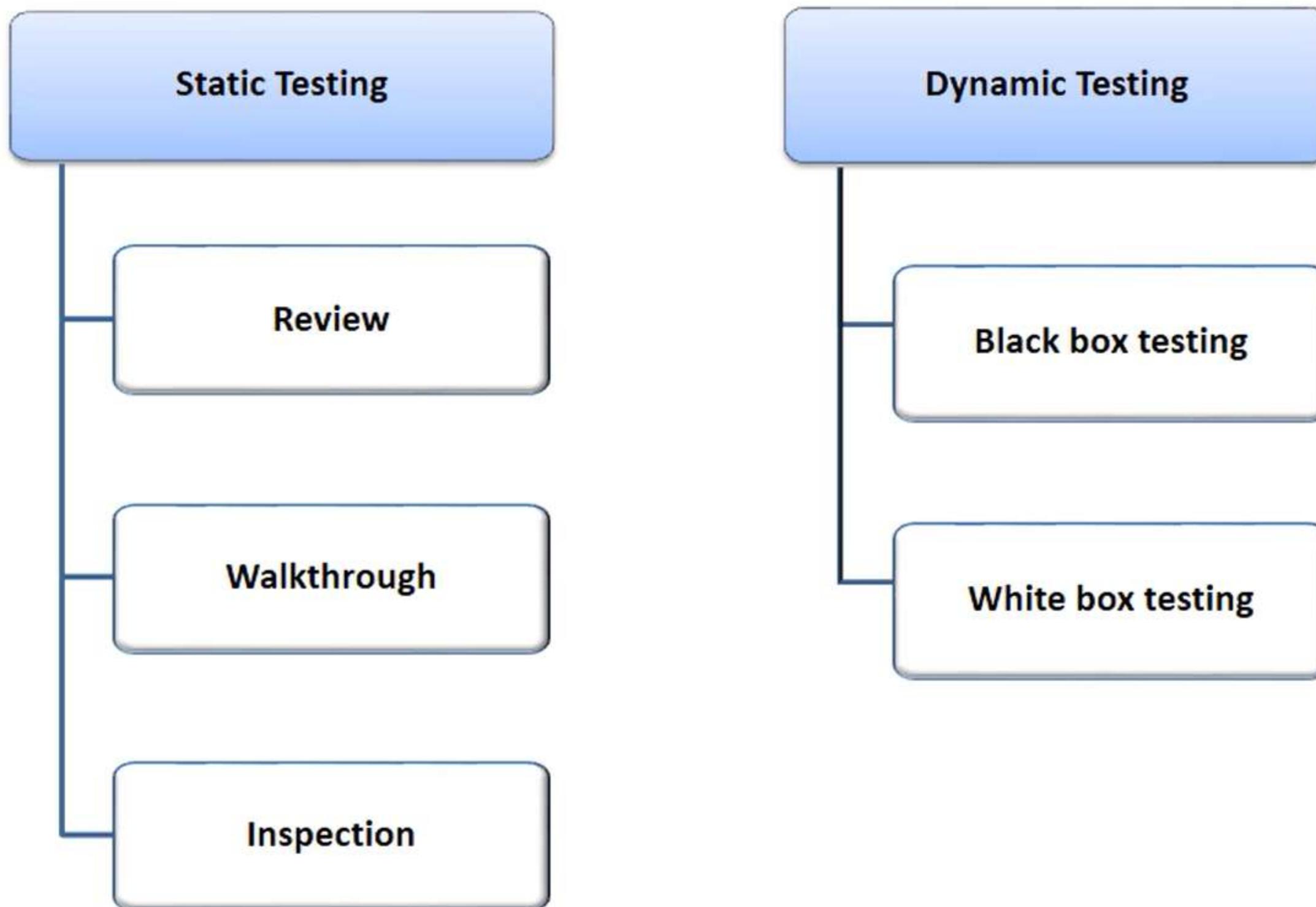
User Acceptance Testing

Done by the client to test whether the system meets the specified requirements

Types

- **Alpha testing** - Done by the client in developer's environment
- **Beta testing** - Done by the client in real world environment

Types of Testing



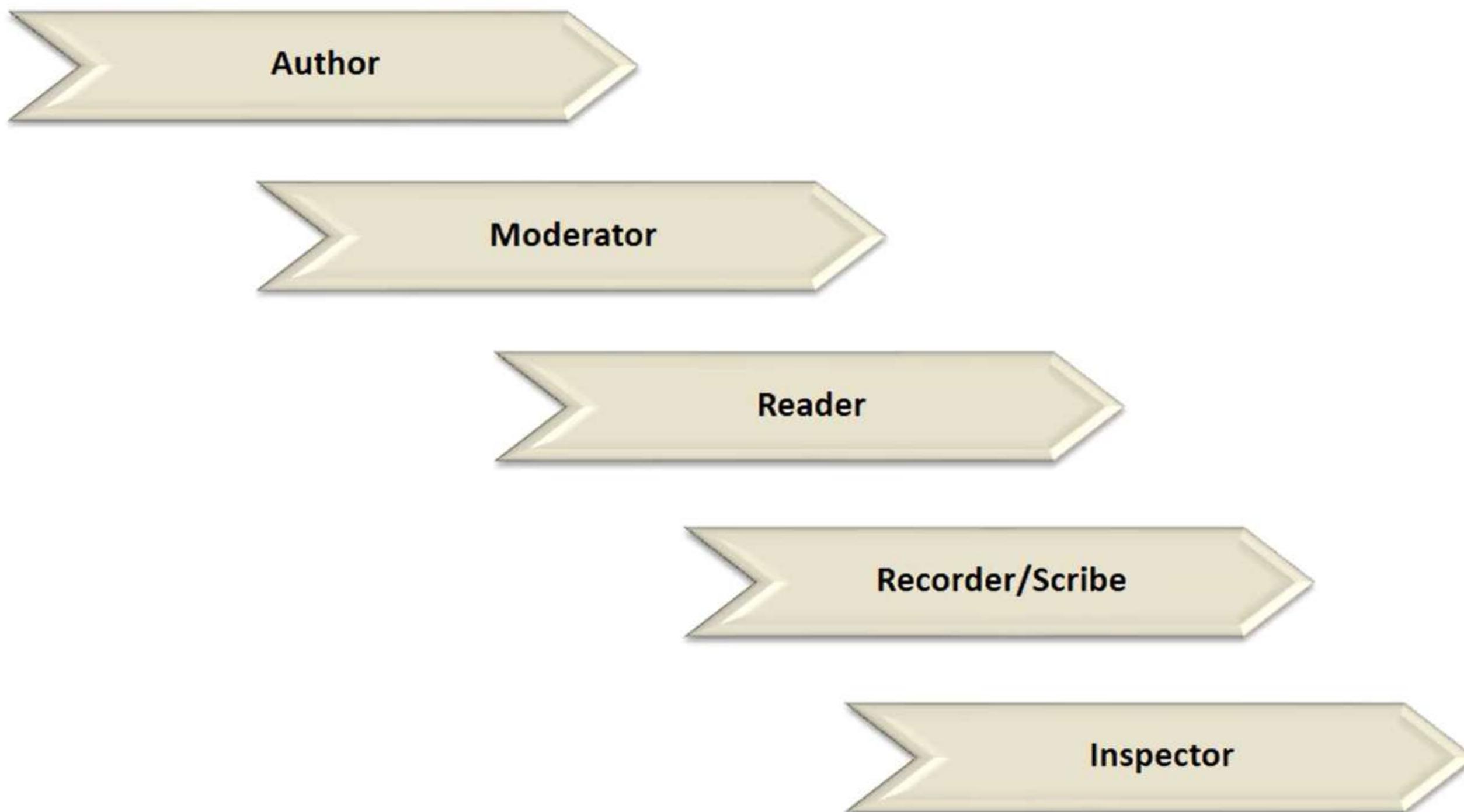
Static Testing

Static testing is the testing of the software work products manually to find errors

Work product means all the documents related to software as well as the code

Execution of the Code is not performed. Sanity of the code is checked.

Members of Static Testing



Techniques in Static Testing

- **REVIEW**
 - It is a process in which the product is re-examined or re-evaluated for possible corrections
- **WALK THROUGH**
 - Mostly done on the code that is developed
 - The author gives sample test data. The test data examined with the code and intermediate results are recorded
- **INSPECTION**
 - Inspection involves step by step reading of the product, with each step checked against a predefined list of criteria(historical common errors, standards)

Review

- **Review is a static testing technique.**
- **During review the artifacts are manually examined and the findings are recorded.**
- **Artifacts for which reviews are performed are**
 - ❖ SRS
 - ❖ Design specification
 - ❖ Source code
 - ❖ Test plans
 - ❖ Test specification
 - ❖ Test cases
 - ❖ Test scripts
 - ❖ User guides
 - ❖ web pages

Review Advantages

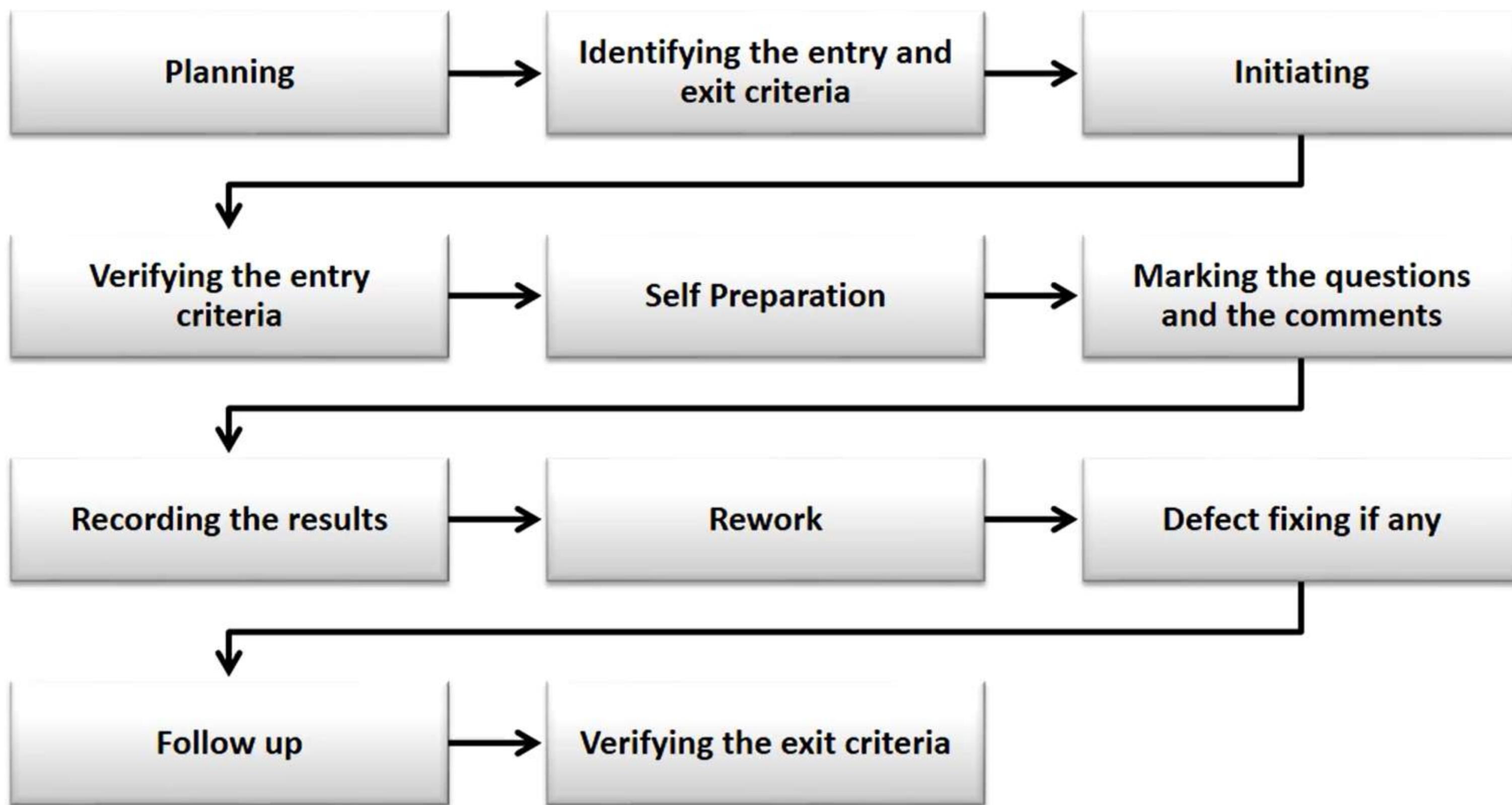
Early defect detection and correction

Fewer defects

Helps in identifying omissions

Saves testing cost and time

Review Process



Types of Review

Informal Review

Walkthrough

Technical Review

Inspection

Types of Review

Informal Review

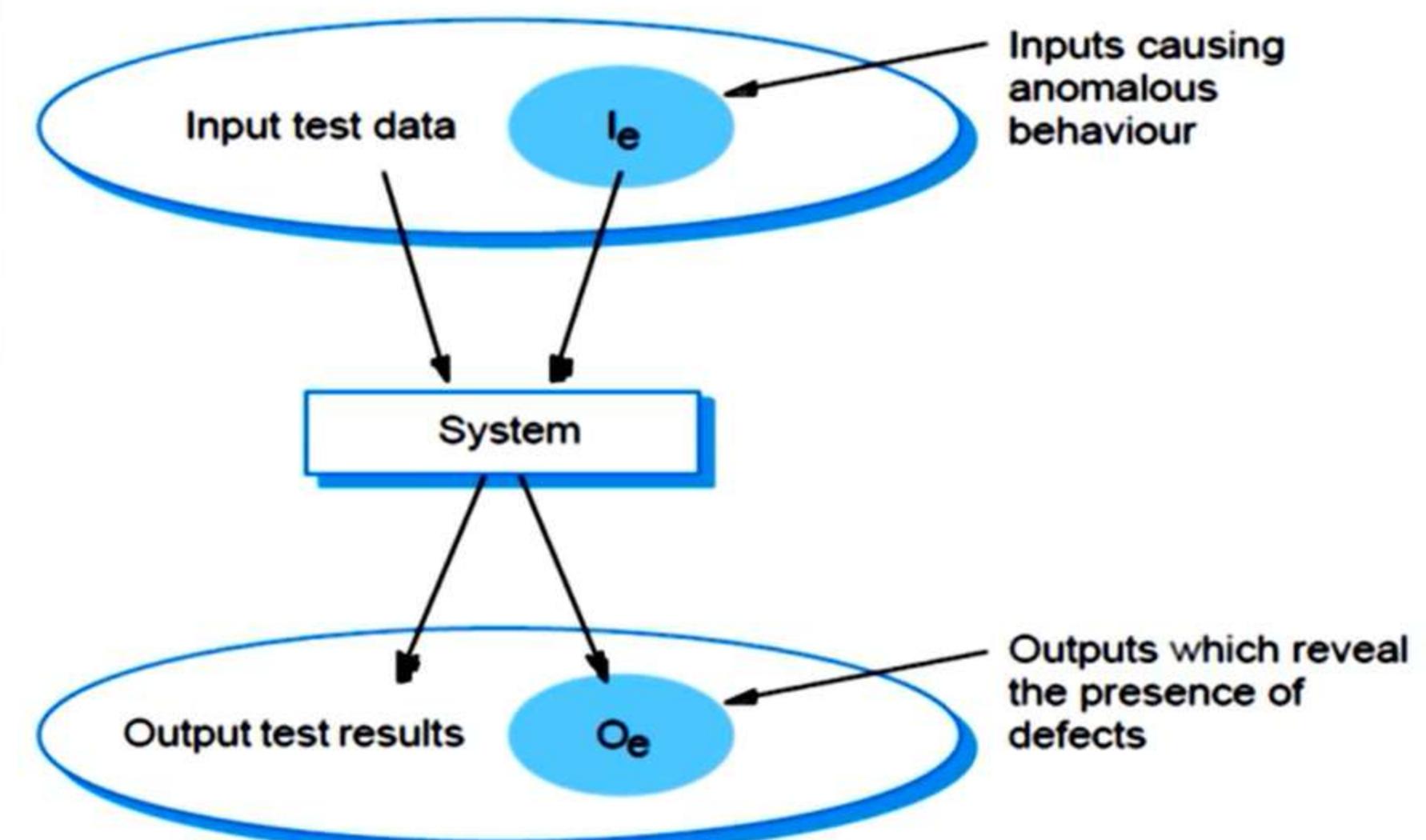
Walkthrough

Technical Review

Inspection

Dynamic Testing – Black Box Testing

It is testing that ignores the internal mechanism of a system or a component and focuses solely on the outputs generated in response to selected inputs and execution conditions.



Test Case

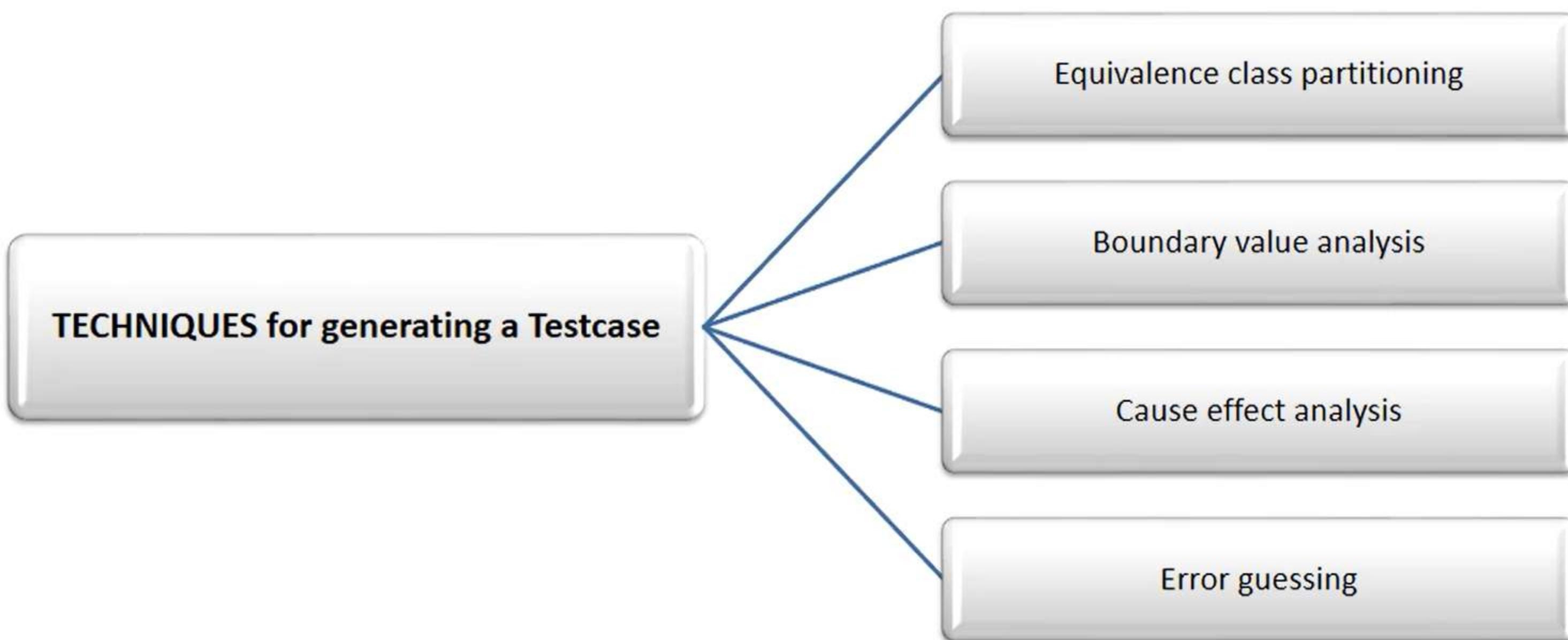


Once the test plan for the level of testing has been written, the next stage of test design is to specify a set of test cases or test path for each item to be tested at that level

Each test case will specify how the implementation of the particular requirement or design decision is to be tested and the criteria for success of the test

A critical test case uncovers a new error

Black Box Testing



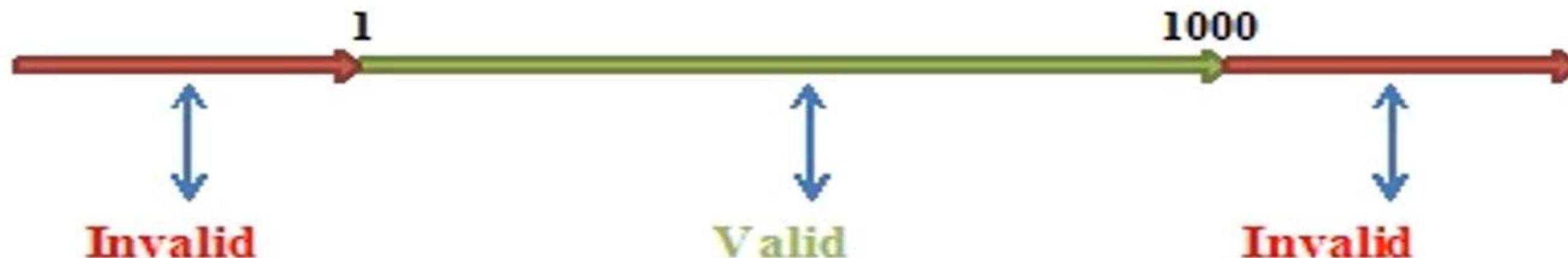
Equivalence Class Partitioning

Black-box technique divides the input domain into classes of data from which test cases can be derived.

For each of these equivalence classes, the set of data should be treated the same by the module under test and should produce the same answer.

Equivalence class guidelines:

- For a range of input choose three values such that,
 - One value is above the range
 - One value is below the range
 - One value is within the range



Equivalence Class Partitioning

- **EXAMPLE SCENARIO**

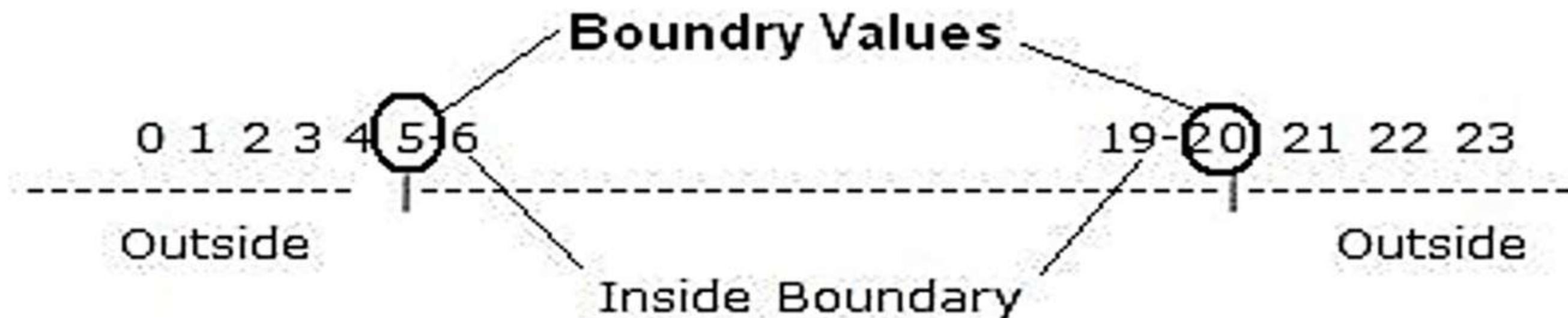
- The component “generate_grading” is passed an exam mark (out of 75) and a coursework (c/w) mark (out of 25), from which it generates a grade for the course in the range 'A' to 'D'. The grade is calculated from the overall mark, which is calculated as the sum of the exam and c/w marks, as follows:

- | | |
|---|-------|
| • greater than or equal to 70 | - 'A' |
| • greater than or equal to 50, but less than 70 | - 'B' |
| • greater than or equal to 30, but less than 50 | - 'C' |
| • less than 30 | - 'D' |
| • When a mark is outside its expected range, then a fault message ('FM') is generated. All inputs are passed as integers. | |



Boundary Value Analysis

- Black-box technique focuses on the boundaries of the input domain rather than its center
- Boundary value analysis guidelines:
 - If input condition specifies a range bounded by values a and b, test cases should include a and b, values just below a and just above b



Cause Effect Analysis



The cause effect analysis helps in identifying the causes and their effects (and identify how they are linked) associated with a particular problem or situation.

A cause is an input condition or an equivalence class of input conditions. An effect is an output condition or a system transformation.

It is suitable for applications in which combinations of input conditions are few.

Cause Effect Analysis

EXAMPLE:

- A module for calculating increment, that decides how much increment percent will be assigned to every employee. The decision based on the experience and the designation
 - For exp (valid input range is 1 to 50)
 - Between 1 – 3 then inc% 10
 - Between 4 – 10 then inc% 20
 - Greater than 10 then inc% 30
 - For designation
 - Developer then inc% 30
 - Tester then inc% 40

Error Guessing



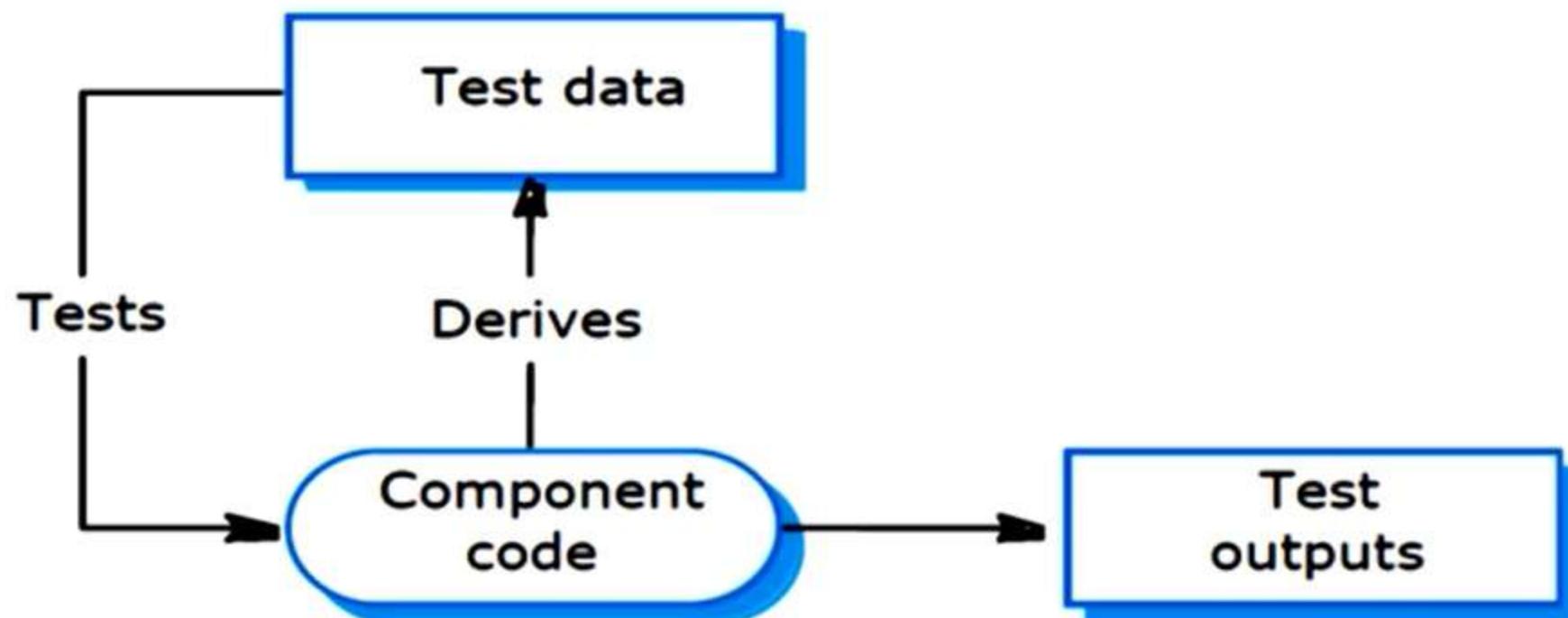
- It is an ad hoc approach guided by intuition and experience
- Example:
 - Consider a program is written into a file. The following test cases can be derived
 - Having an empty file
 - Not at all having a file
 - Having a file with read permission

Dynamic Testing – White Box Testing

It deals with the internal logic and structure of the code

Derivation of test cases according to the program structure

It is also called as glass box testing, structural testing, clear box testing



Basis Path Testing



- Basis Path Testing is white box testing method
- Design test cases to cover the following in the code written
 - ❖ every statement(Statement coverage)
 - ❖ every predicate (condition) in the code(branch coverage)
 - ❖ loops (loop coverage)

Basis Path Testing



Derive a logical complexity measure of procedural design

- Break the module into blocks delimited by statements that affect the control flow (eg condition, method call, loop)
- Mark out these as nodes in a control flow graph
- Draw connectors (arcs) with arrow heads to mark the flow of logic
- Identify the number of regions (Cyclomatic Number) which is equivalent to the McCabe's number

McCabe's Number (Cyclomatic complexity)

- It defines the number of independent paths
- Provides the number of tests that must be conducted to ensure that all the statements are executed at least once.

Basis Path Testing

Complexity of a flow graph 'G', $v(G)$, is computed in one of three ways

- $v(G) = \text{No. of regions of } G + 1$
- $v(G) = E - N + 2$ (E : No. of edges & N : No. of nodes)
- $v(G) = P + 1$ (P : No. of predicate nodes in G or No. of conditions in the code)

Define a basis set of execution paths

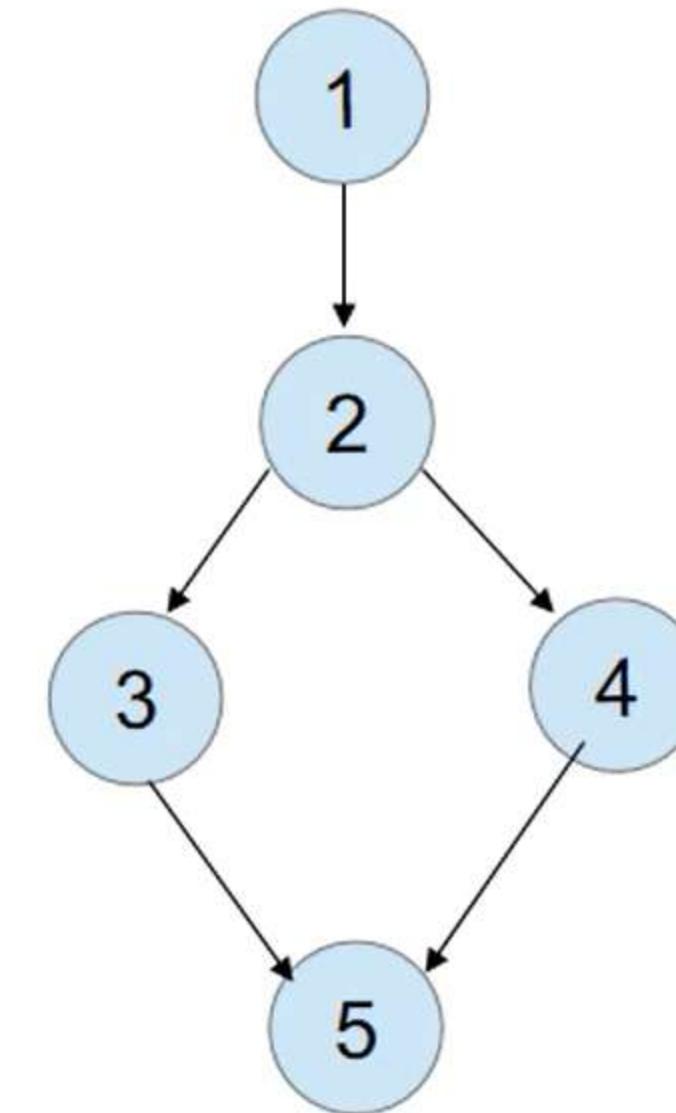
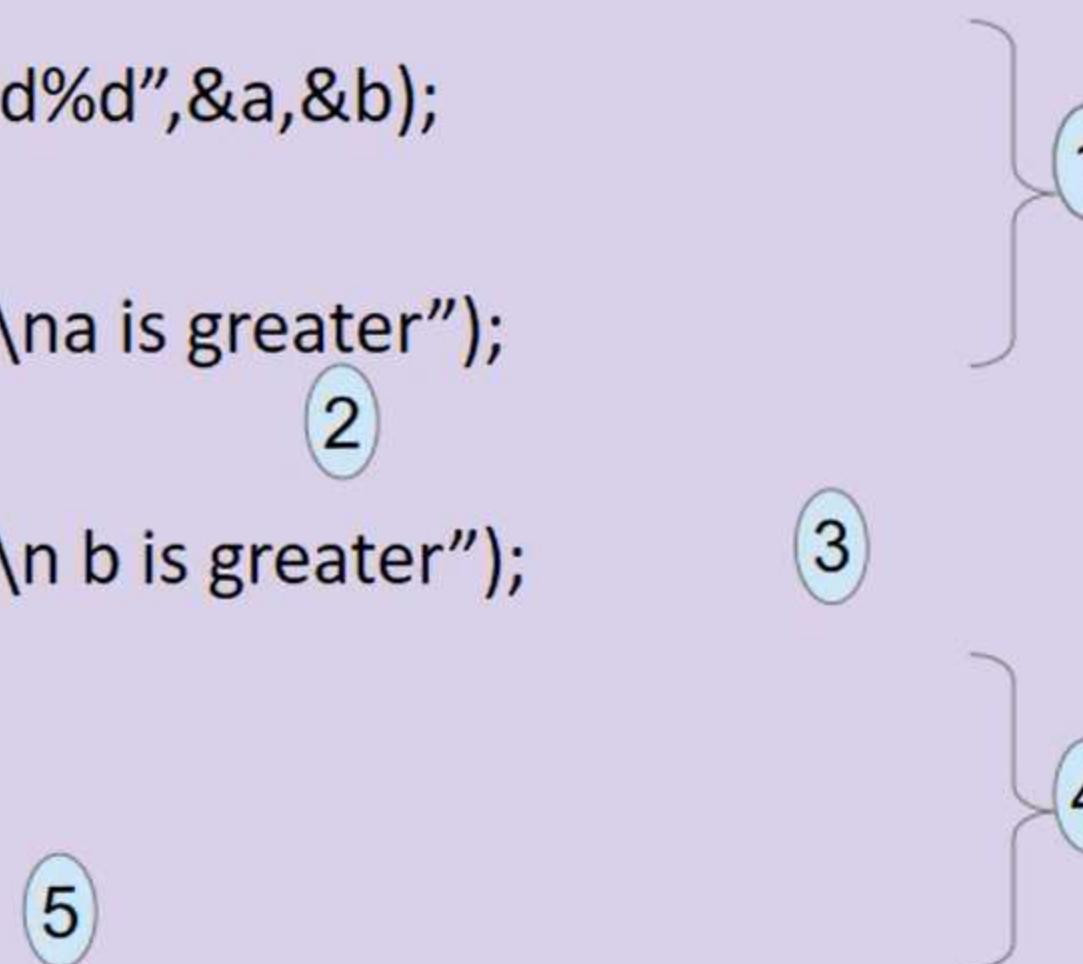
Determine independent paths

Eliminate infeasible paths

Derive test case to exercise (cover) the basis set

Basis Path testing

- int main() {
 - int a,b;
 - scanf("%d%d",&a,&b);
 - if(a > b)
 - printf("\n a is greater");
 - else
 - printf("\n b is greater");
 - }



Mc cab's No

$$P+1 = 2$$

$$E-N+2 = 5-5+2 = 2$$

$$R+1 = 2$$

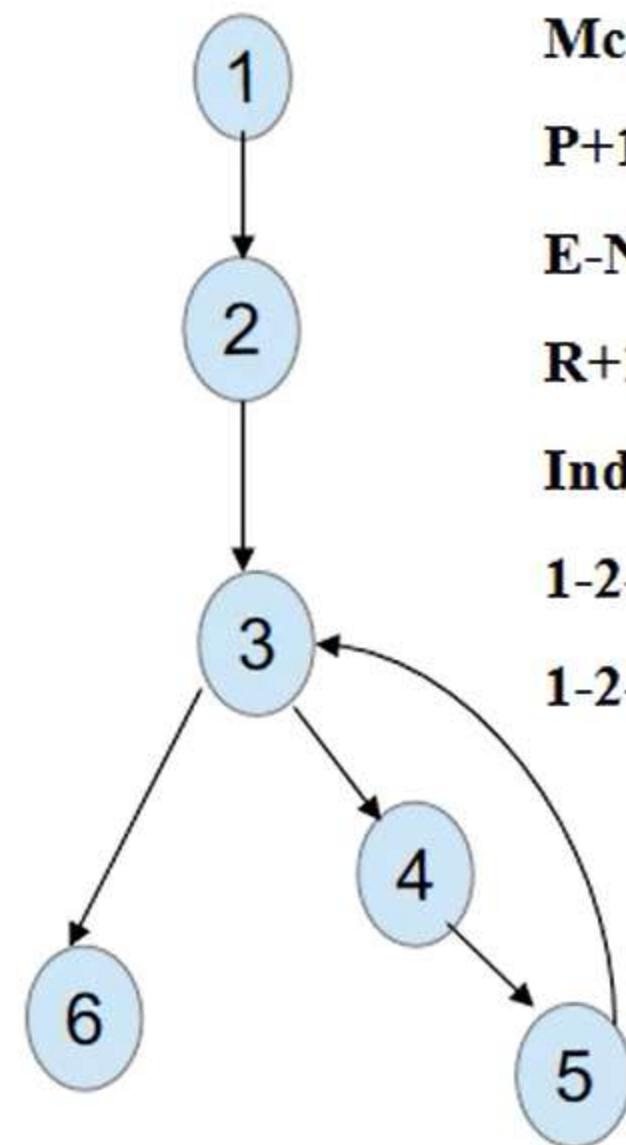
Individual Path	testcase
1-2-3-5	a=10 , b=5
1-2-4-5	a=5 , b=10

Basis Path Testing

```

• int main() {
    – int n,i;
    – scanf("%d",&n);
    – for(i=1; i<=n; i++) {
        – printf("\n%d",i);
        – }
    • }
  }
```

1 2 3 4 5 6



Mc cab's no

$$P+1 = 2$$

$$E-N+2 = 6-6+2 = 2$$

$$R+1 = 2$$

Individual Path

1-2-3-4-5-3-6 n=2

1-2-3-6 n=0

Black Box vs. White Box Testing

BLACK BOX TESTING

- Test cases are designed depending on the functionality
- Identifies hidden functionality
- Techniques
 - Equivalence class partitioning
 - Boundary value analysis
 - Cause effect analysis and graphing
 - Decision table

WHITE BOX TESTING

- Test cases are designed depending on the logic of the code
- Identifies unreachable code and checks for code coverage
- Techniques
 - Basis path testing

Static Testing Vs Dynamic Testing



STATIC TESTING

- manual examination (reviews) of the code or other project documentation
- Testing methodologies
 - Review
 - Inspection
 - Walkthrough
- Bugs discovered at this point are less expensive to fix

DYNAMIC TESTING

- the examination of the physical response from the system to variables that are not constant and change with time
- Testing methodologies
 - Black box testing
 - White box testing
- Depends on the type of bug identified.

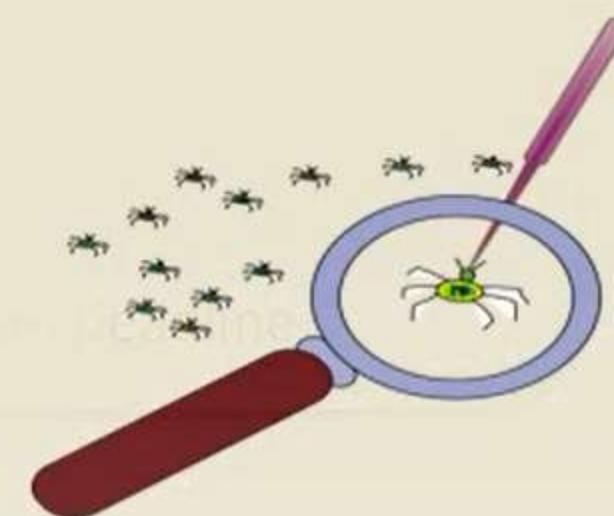
When to Stop Testing?

Some of the common factors to stop Testing

- Testing budget of the project
- Resources available and their skills
- Project deadline and test completion deadline

Debugging

- Debugging is the process of finding the source of already identified bugs and fixing it.
- Performed by Developers.
- Steps in debugging
 - Find the defect in the code
 - Identify the root cause of the problem
 - Identify the exact place in the code that is the cause of the problem
 - Fix the defect
 - Recheck to ensure that the defect fixed is working as expected



Summary

- What is Testing ?
- Importance of Testing
- Levels of Testing
- Types of Testing
- Debugging



SOFTWARE MAINTENANCE



In this module you will learn

- Software Evolution
- Software Change and its Strategies
- Software Maintenance
- Types of Software Maintenance



Software Evolution

It is impossible to produce a system of any size which does not need to be changed.

Parts of the software may have to be modified to correct the errors that are found in operation, or to improve its performance or other non-functional characteristics.

After delivery, software systems always evolve in response to demand for change.

Software Evolution Approaches

Software maintenance

Architectural transformation

Software re-engineering.



Software Change

New requirements emerge when the software is used

The business environment changes

Errors must be repaired

New equipment must be accommodated

The performance or reliability may have to be improved

Software change strategies

Software Maintenance

- Changes are made in response to changed requirements but the fundamental software structure is stable

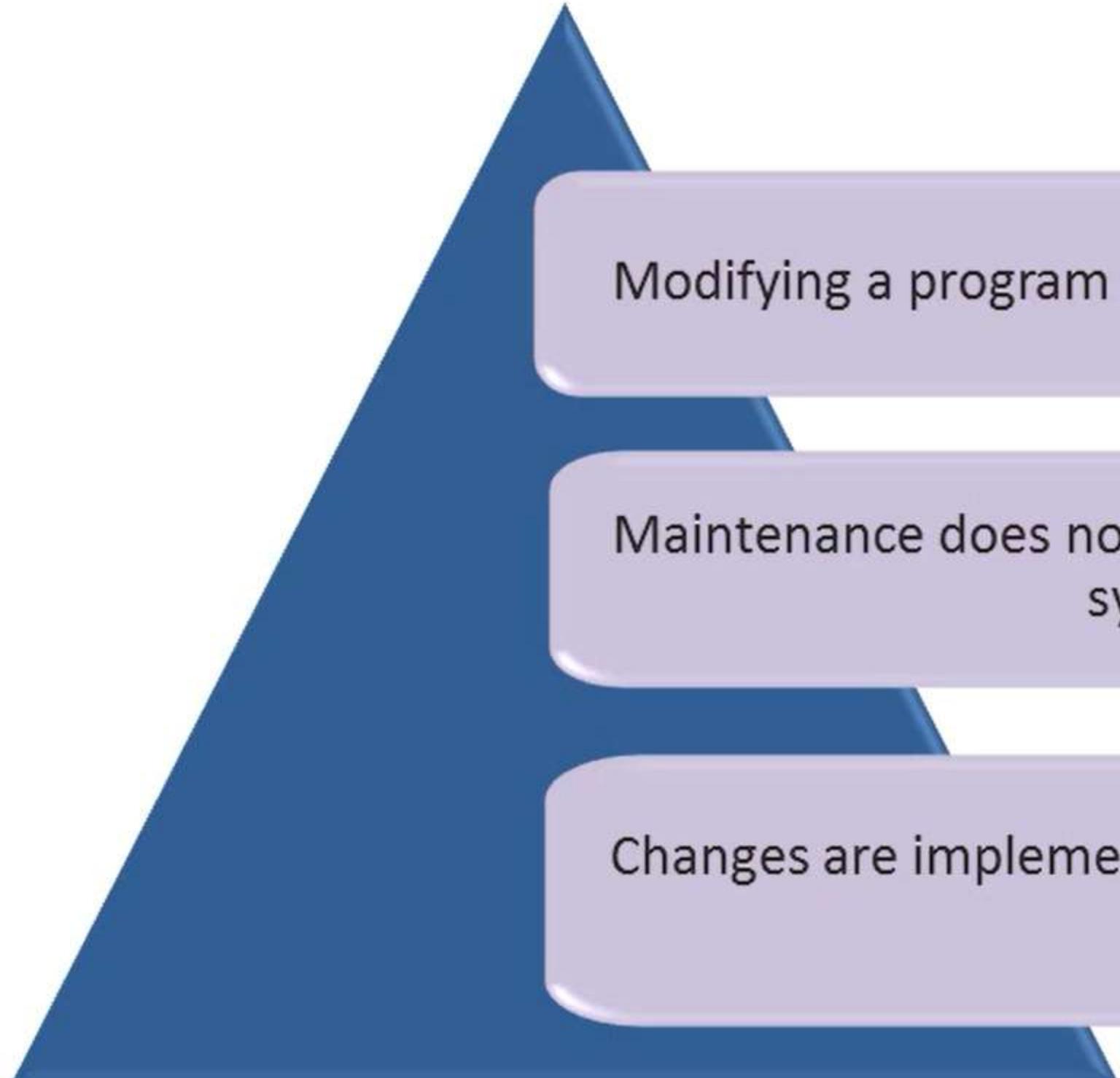
Architectural Transformation

- The architecture of the system is modified
- Generally from a centralized to a distributed architecture

Software Re-engineering

- New functionality is not added to the system but it is restructured and reorganized to facilitate future changes

Software Maintenance

A large blue triangle is positioned on the left side of the slide, pointing towards the text boxes.

Modifying a program after it is used

Maintenance does not normally involve major changes to the system's architecture

Changes are implemented by modifying the existing components or by adding new components to the system

Types of maintenance

Corrective

Adaptive

Perfective

Preventive

Types of maintenance

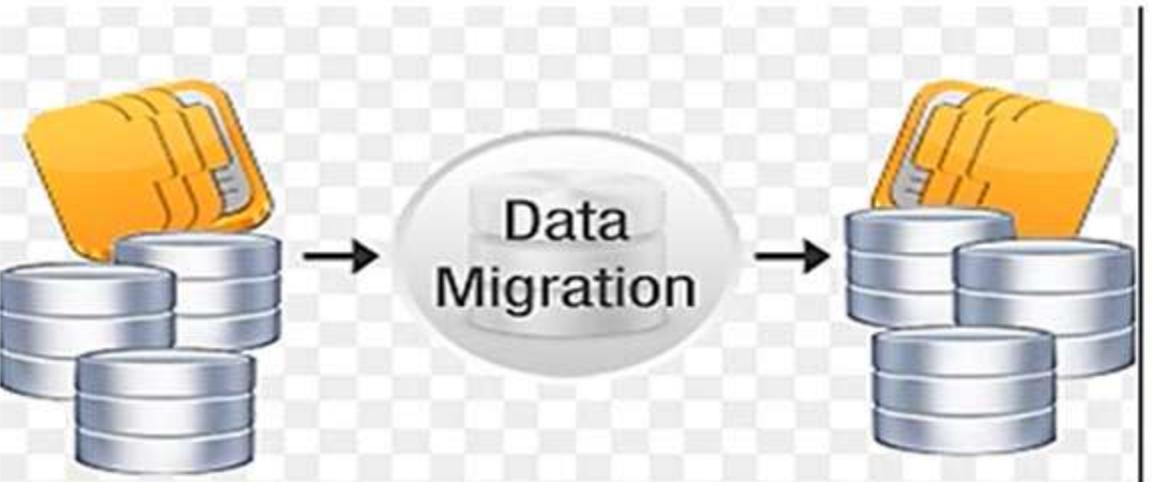
Corrective maintenance

- Maintenance that includes the repair of defects in an existing system
 - Defects can stem from
 - Requirements specification errors
 - Design errors
 - About 80% of all problems stem from requirements and design
 - Coding errors



Adaptive maintenance

- Maintenance to adapt software to changes in the working environment
- Invoked by:
 - Internal needs
 - External requirements e.g. changes in law



Types of maintenance

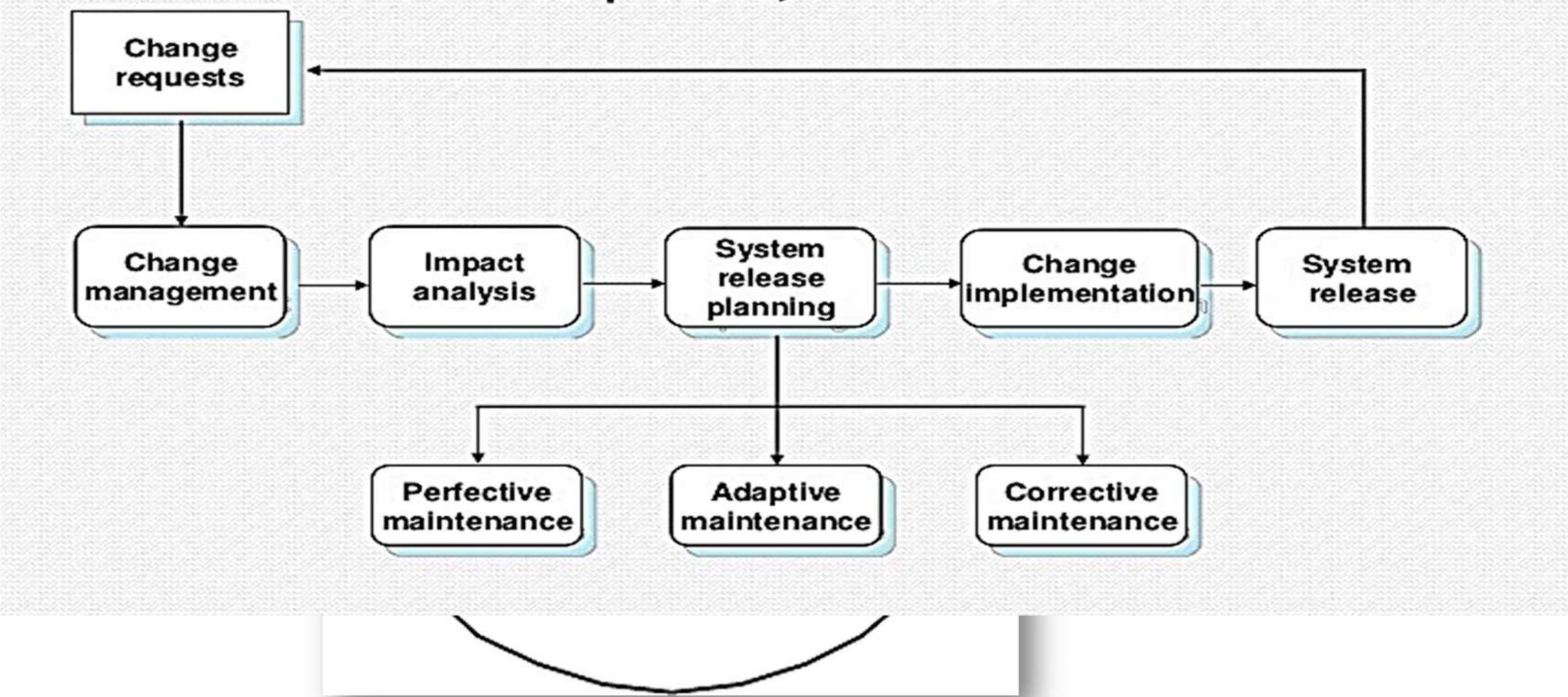
Perfective maintenance

- Includes all efforts to polish or refine the quality of the software or the documentation
- Important that the improvement reduces the system maintenance costs

Preventive maintenance

- Changes made to the system to avoid any software fault in the future

Software Maintenance



Maintenance costs

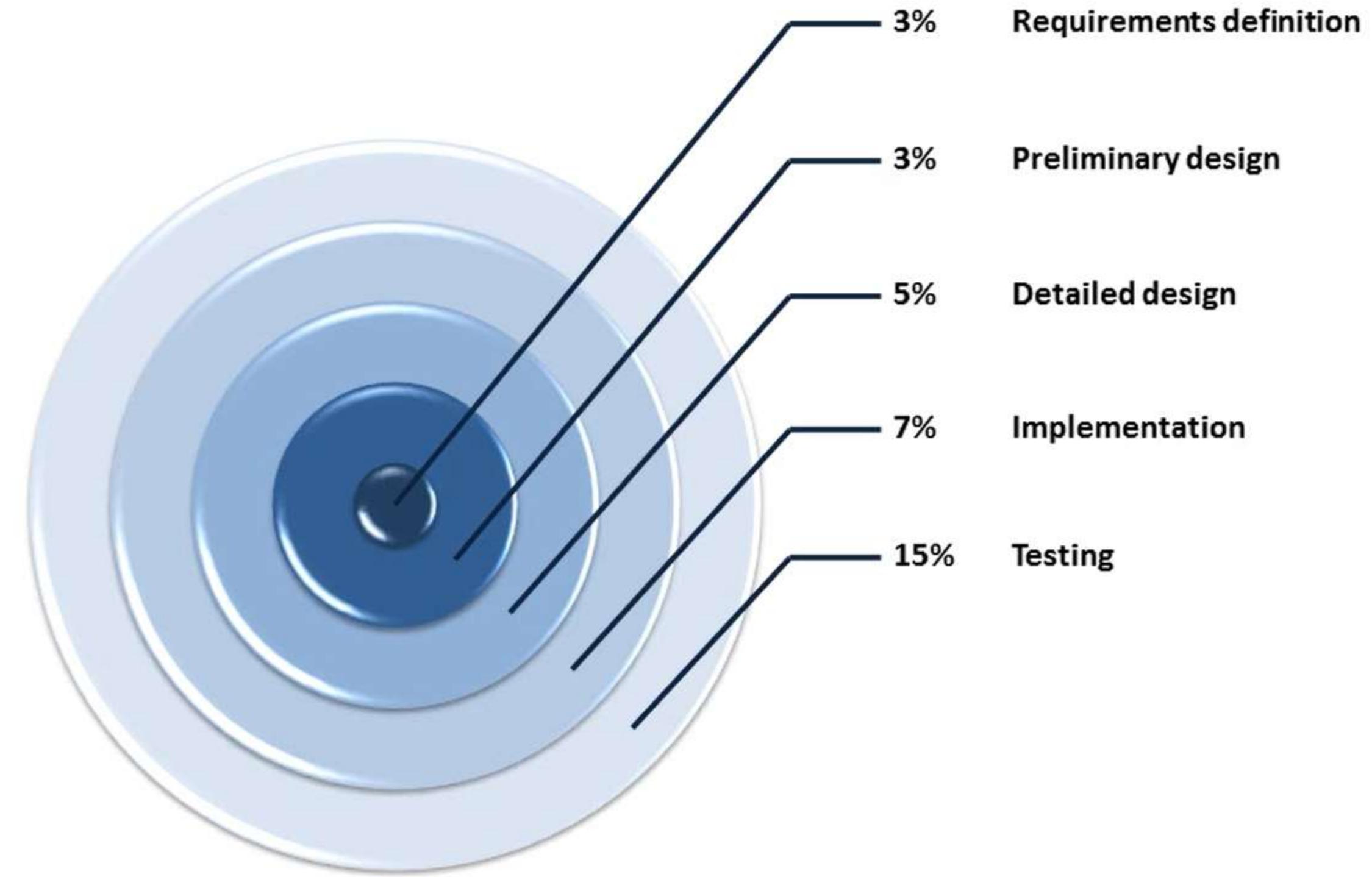
Usually greater than the development costs

Affected by both technical and non-technical factors

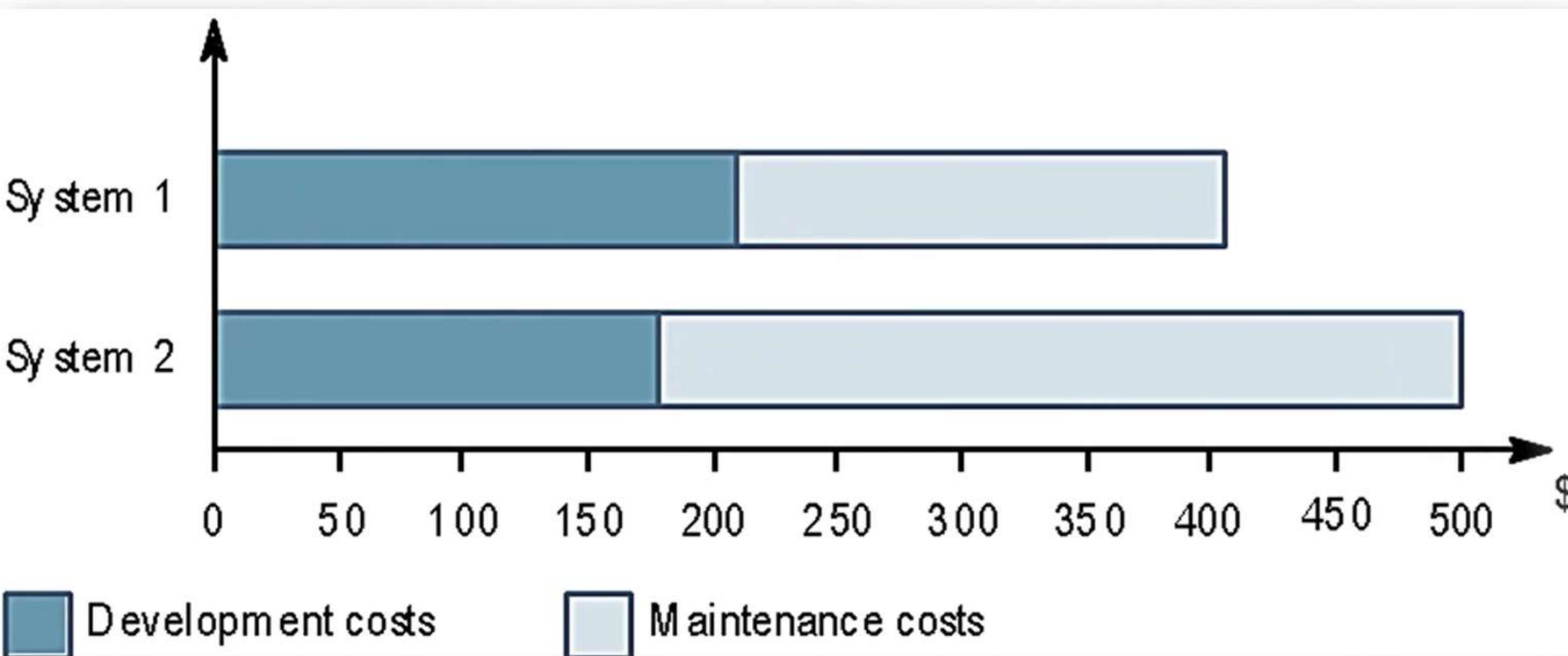
Increases as software is maintained

Ageing software can have high support costs

Relative Costs of Maintenance



Development / Maintenance cost



Maintenance Example



Anti-Virus Software

Operating System
Patching

Y2K

Summary

- Software Evolution
- Software Change and its Strategies
- Software Maintenance
- Types of Software Maintenance



SOFTWARE CONFIGURATION MANAGEMENT



In this module you will learn

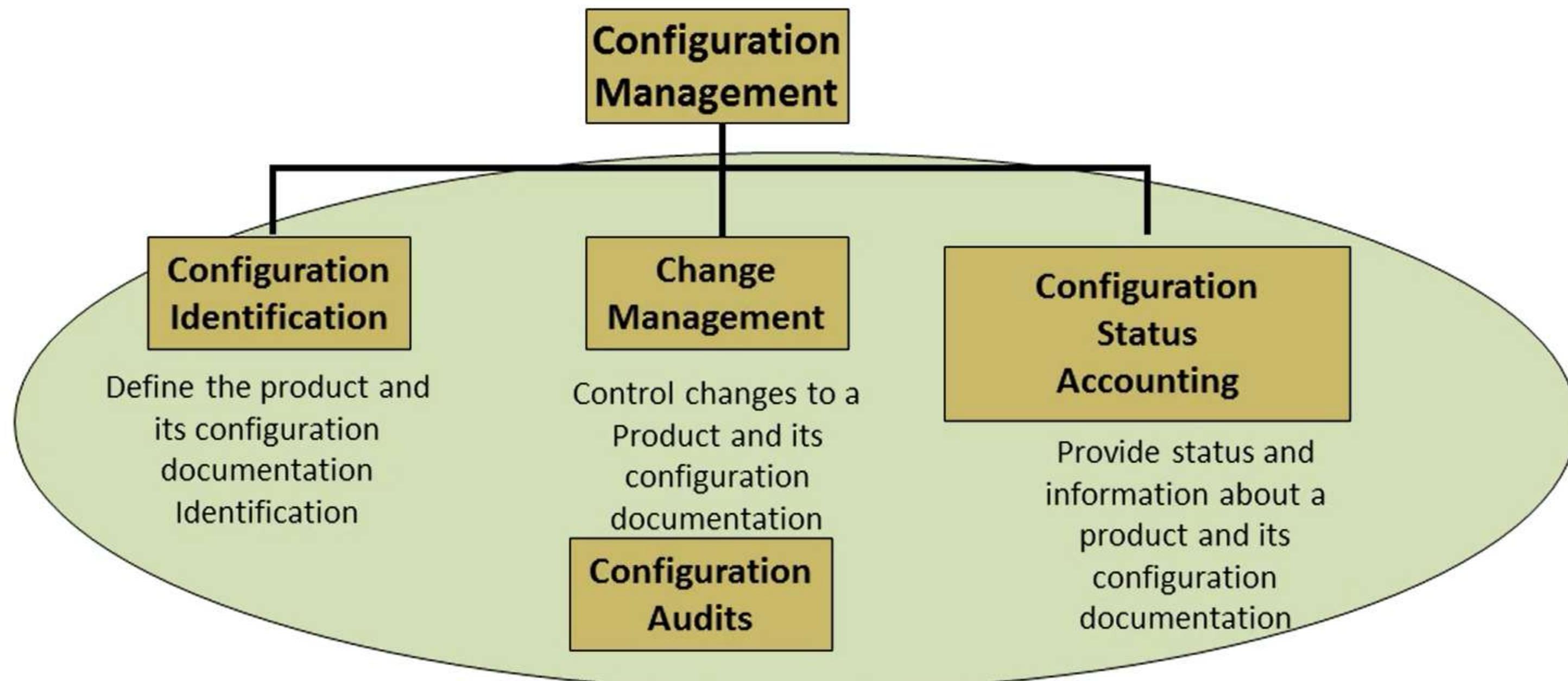
- Software configuration management
- Software configuration Object
- Software Configuration Repository
- Check in check out process
- Change management
- Version management



Software Configuration Management

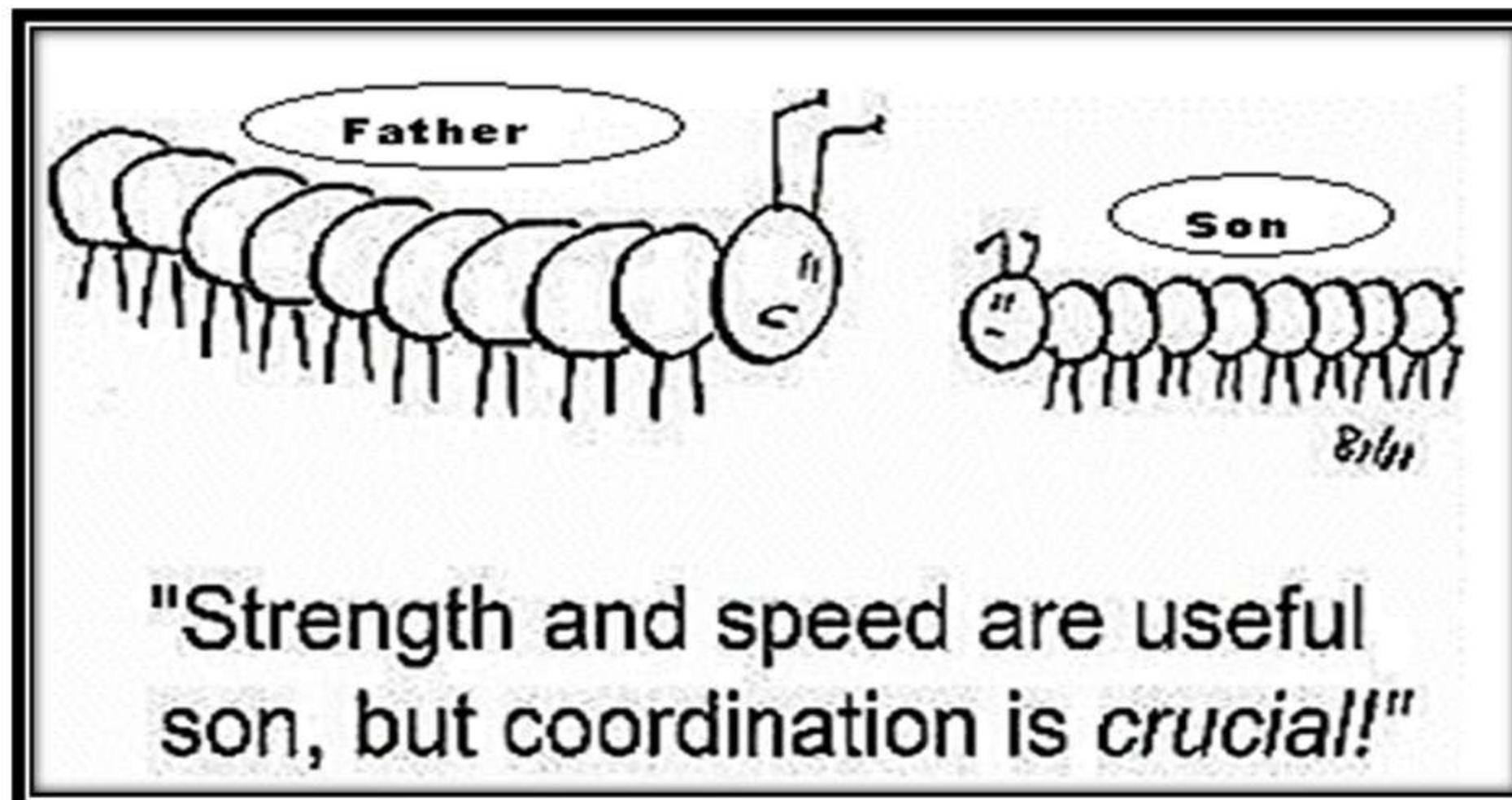
What is configuration

A configuration is the functional and physical characteristics of a hardware or software as set forth in technical documentation or achieved in a product.

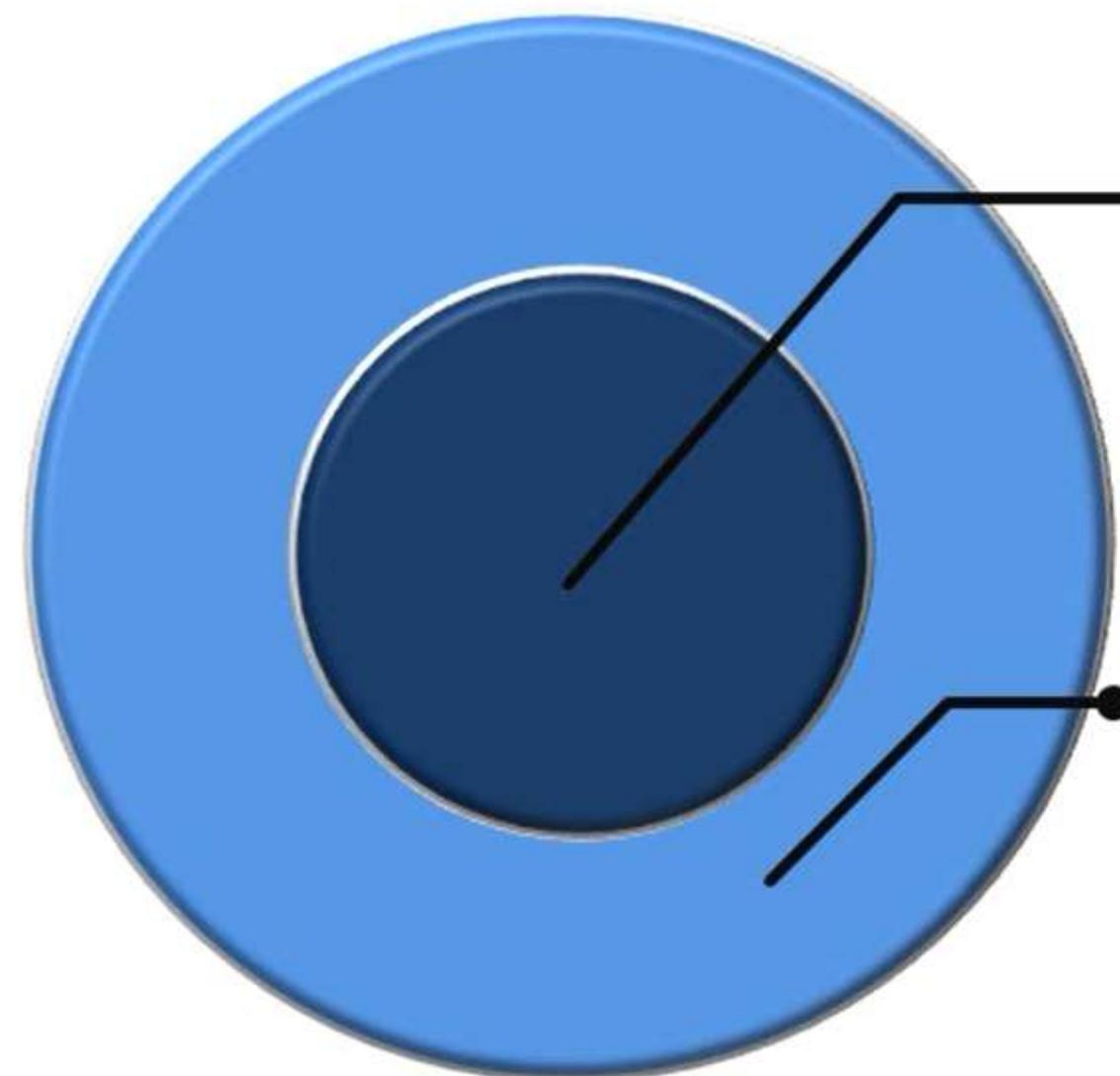


Software Configuration Management

The art of coordinating S/W development activities, minimizing confusion by identifying, modifying and controlling modifications to software



Software Configuration Management



New versions of software systems are created as they change:

- For different machines/OS Offering different functionality
- Tailored for particular user requirements

Configuration management is concerned with managing evolving software systems:

Software Configuration Management



SCM defines

- the types of documents to be managed and a document naming scheme
- who takes responsibility for the CM procedures and creation of “baselines”
- policies for change control and version management
- the CM records which must be maintained

Describes the tools which should be used to assist the CM process and any limitations to their use

Effectiveness of Software Configuration Management



Software entities that SCM is expected to manage include :

- Plans
- Specifications (SRS, Design)
- User Documentation
- Test data
- Support Software Tools, Source Code, Executable, and Libraries

SCM is said to be effective:

- when every work product can be accounted for
- when every work product or change made to it can be tracked and controlled

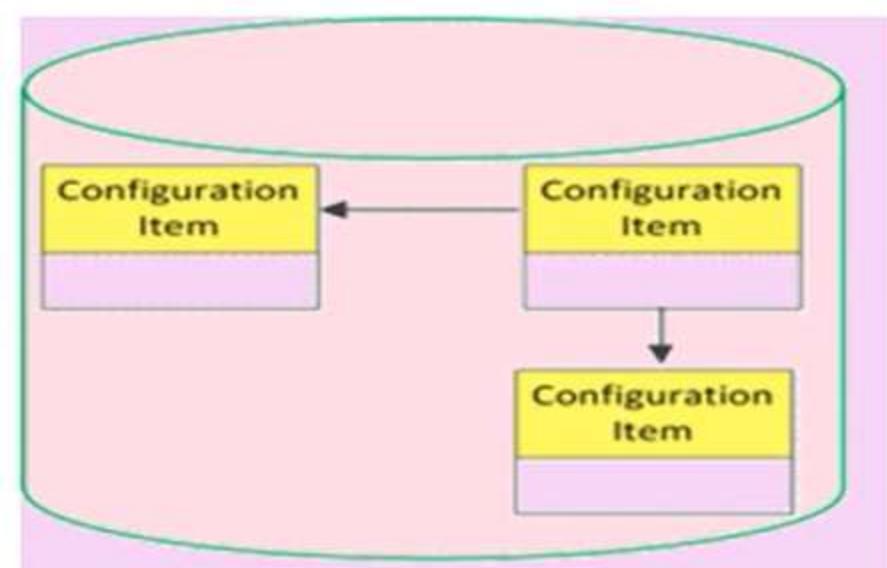
Configuration Item

A Configuration Item is an aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process.

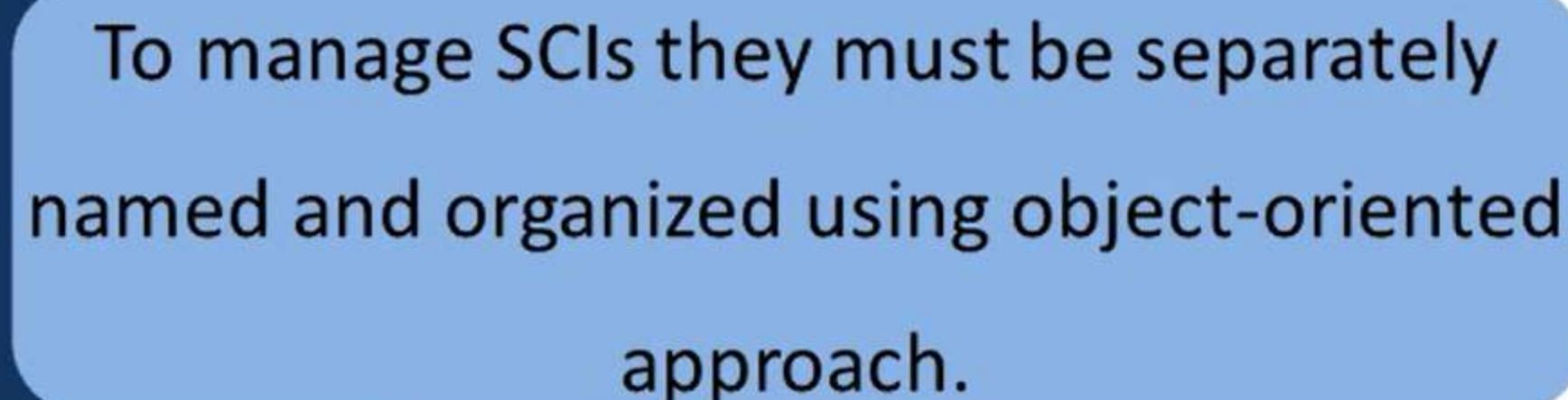
Large projects typically produce thousands of documents which must be uniquely identified.

Some of these documents must be maintained for the lifetime of the software.

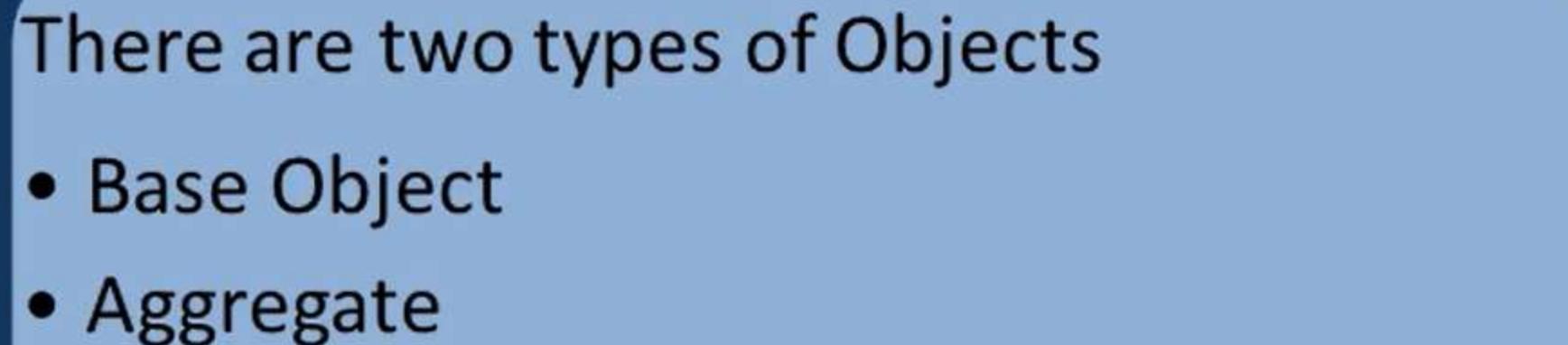
Document naming scheme should be defined so that related documents have related names.



Types of Configuration Objects



To manage SCIs they must be separately named and organized using object-oriented approach.



There are two types of Objects

- Base Object
- Aggregate

Types of Configuration Objects

Base Object

A base object is the "unit of text" that has been created by a software engineer during analysis, design, code, or test.

Example

- section of a requirement specification
- a source listing for a component
- a suite of test cases that are used to exercise the code.

Relationship between Configuration Types of Configuration Objects



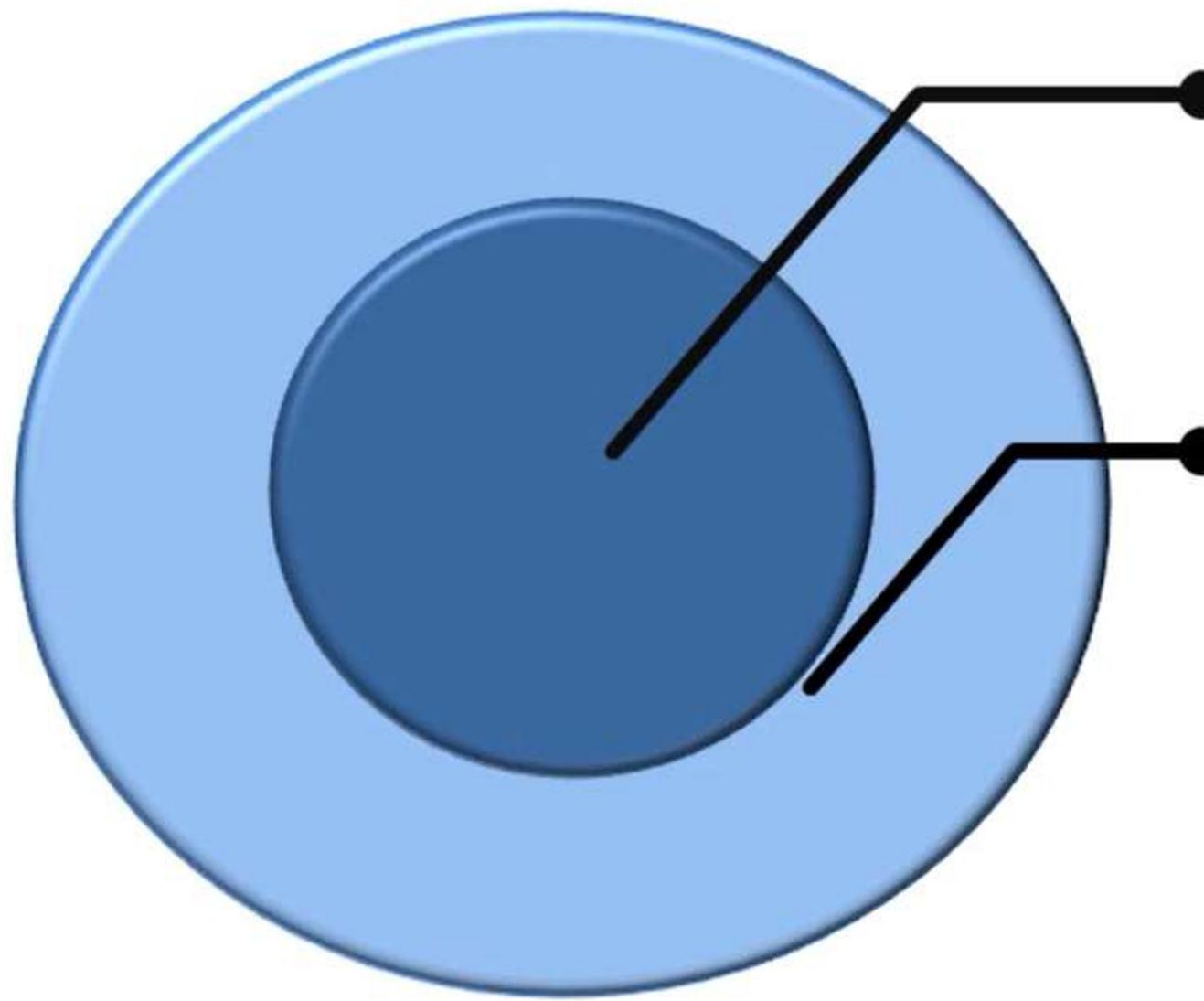
Aggregate Object

- An Aggregate Object is a collection of basic objects and other aggregate objects

Example

- Design Specification is an aggregate object which comprises of Data Design, Architectural Design, Module Design, Interface Design

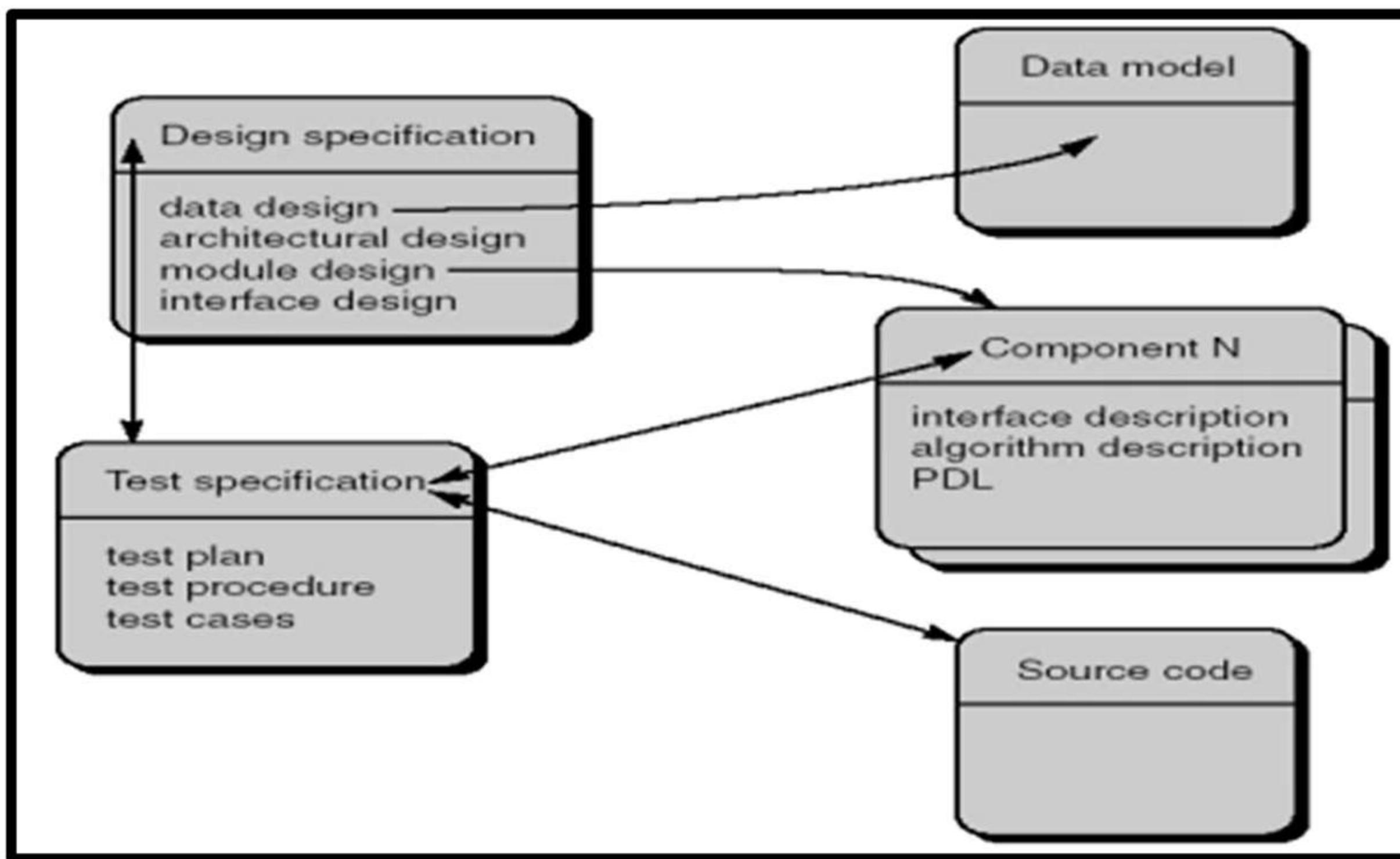
Relationship between Configuration Objects



A curved arrow indicates a
compositional relation.

A double-headed straight arrow
indicates an interrelationship.

Relationship between Configuration Objects



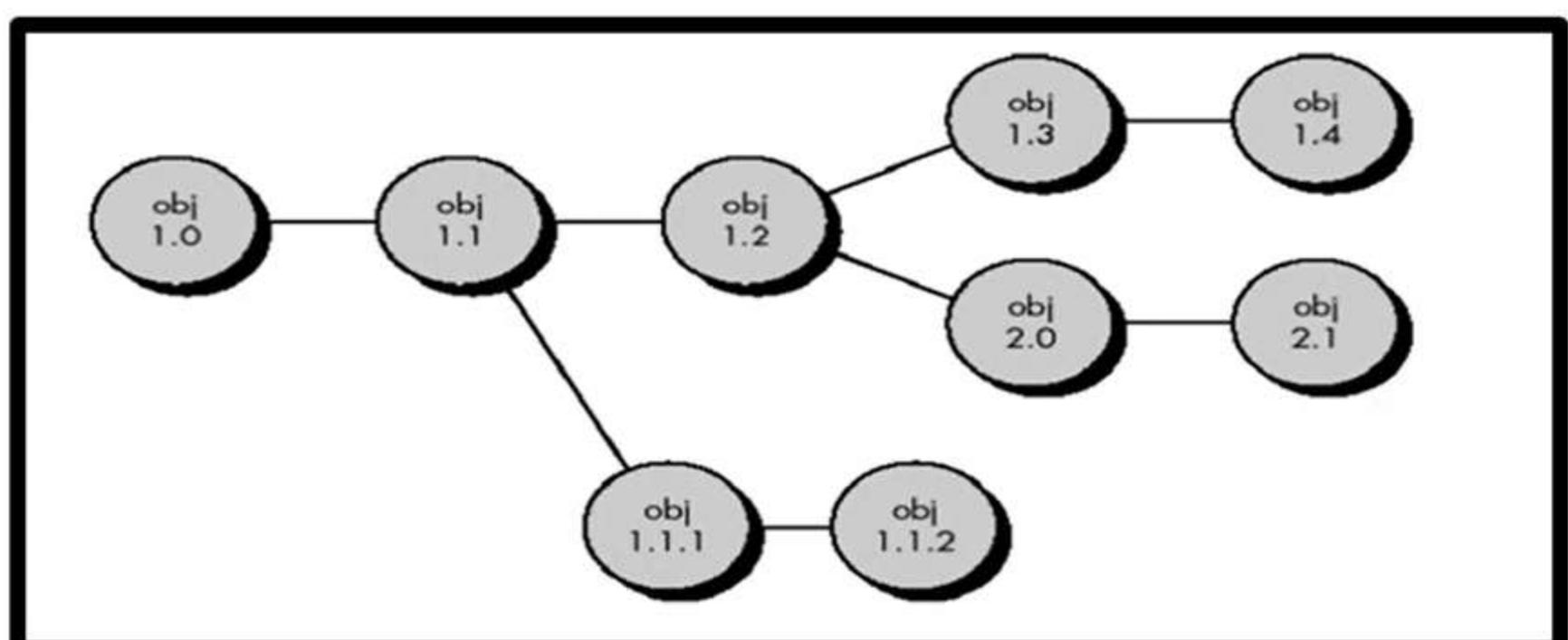
Baseline and Evolution Graph

Baseline

"Specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal Change Control Procedures"

The evolution graph

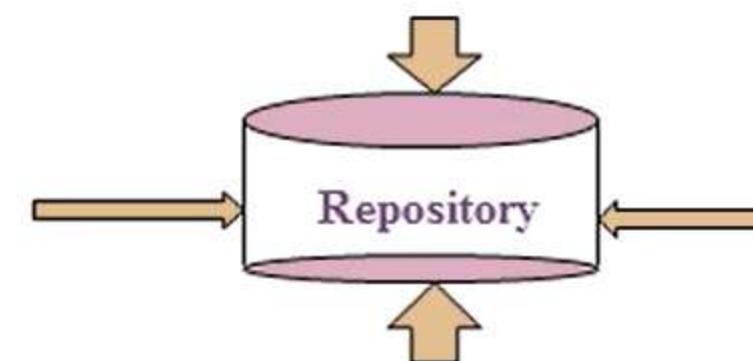
Describes the change history of an object.



Configuration Repository

All CM information should be maintained in a configuration database

This should allow queries about configurations to be answered



- Who has a particular system version?
- What platform is required for a particular version?
- What versions are affected by a change to component X?
- How many reported faults in version T?

The CM database should preferably be linked to the software being managed

Check -in and Check-out

The Configuration items available in the Configuration Management system will be in read-only mode by default

Check in

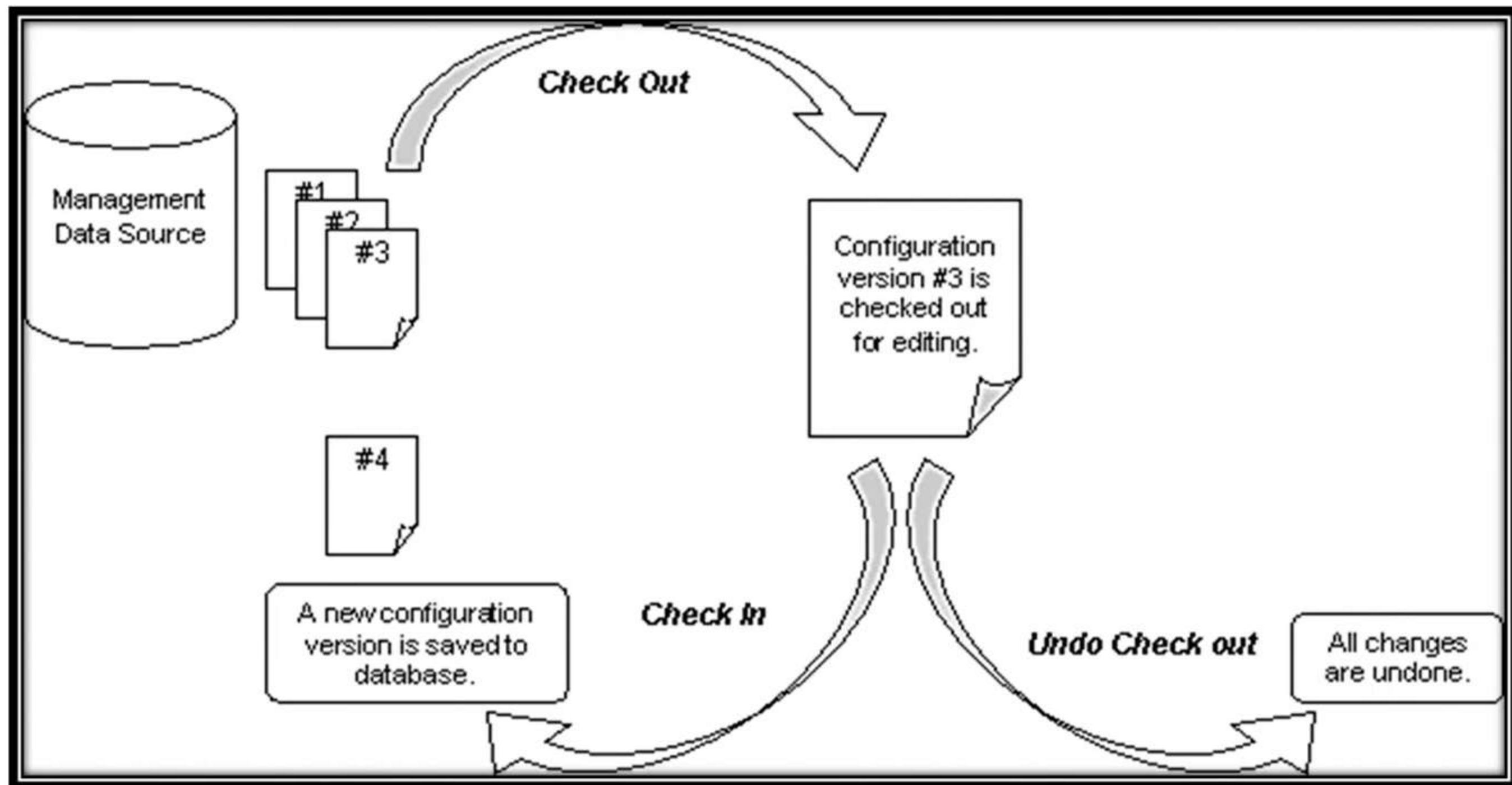
An operation used to make a developer's object version available to other users.

Check out

A process that creates a new version of an object from an existing version stored in the database.

Developers check out objects so they can work on them.

Check -in and Check-out



Change management

Change management (or change control) is the process during which the changes of a system are implemented in a controlled manner by following a pre-defined framework / model with some reasonable modifications.

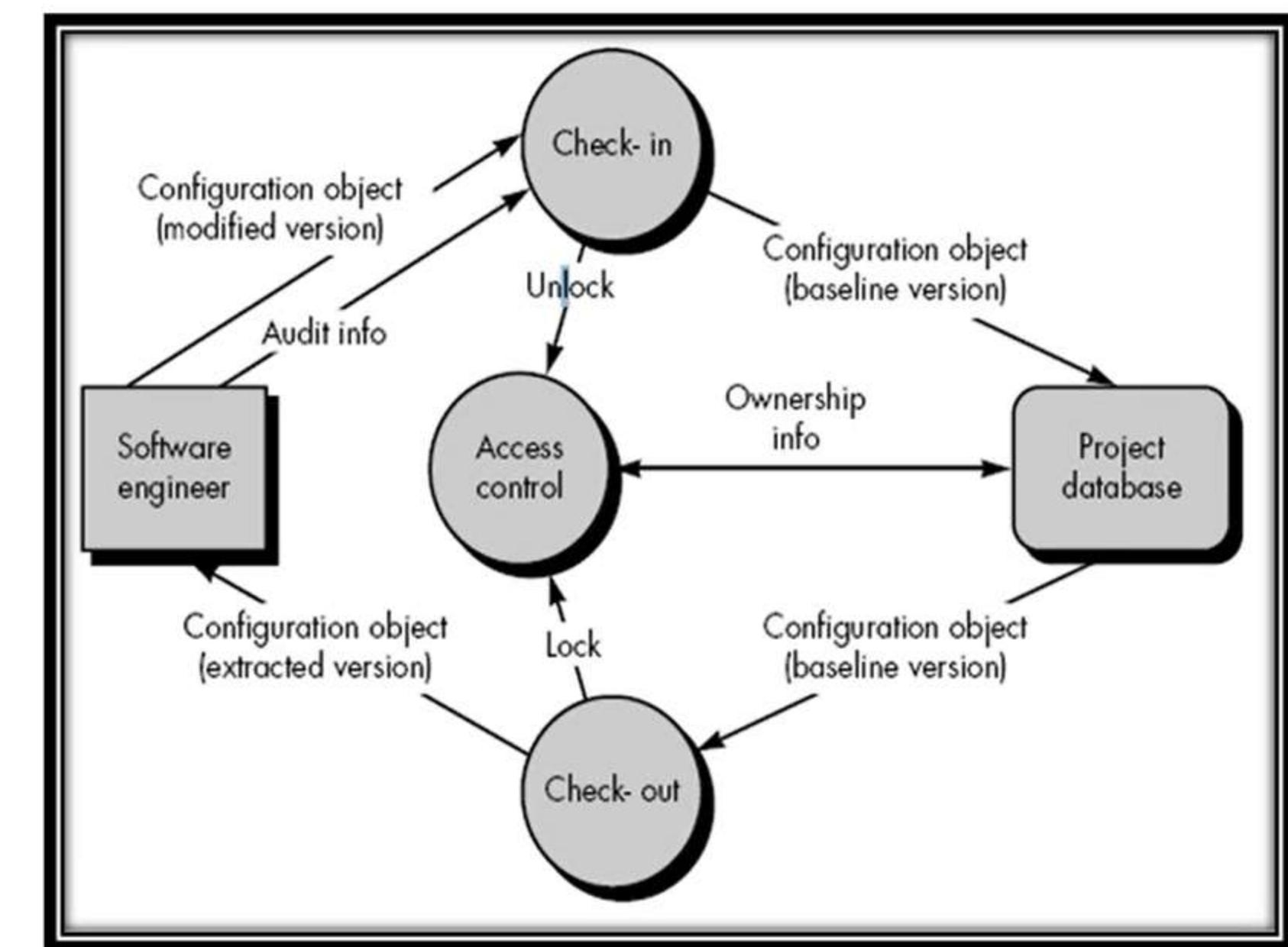
Activities in Change Management:

- Filtering changes
- Managing changes and the change process
- Reviewing and closing of Requests for Change (RFCs)
- Management reporting and providing management information

Synchronization Control

Synchronization Control

- helps to ensure that parallel changes, performed by two different people, don't overwrite each other.
- locks the object in the project database so that no updates can be made to it until the currently checked out version has been replaced.



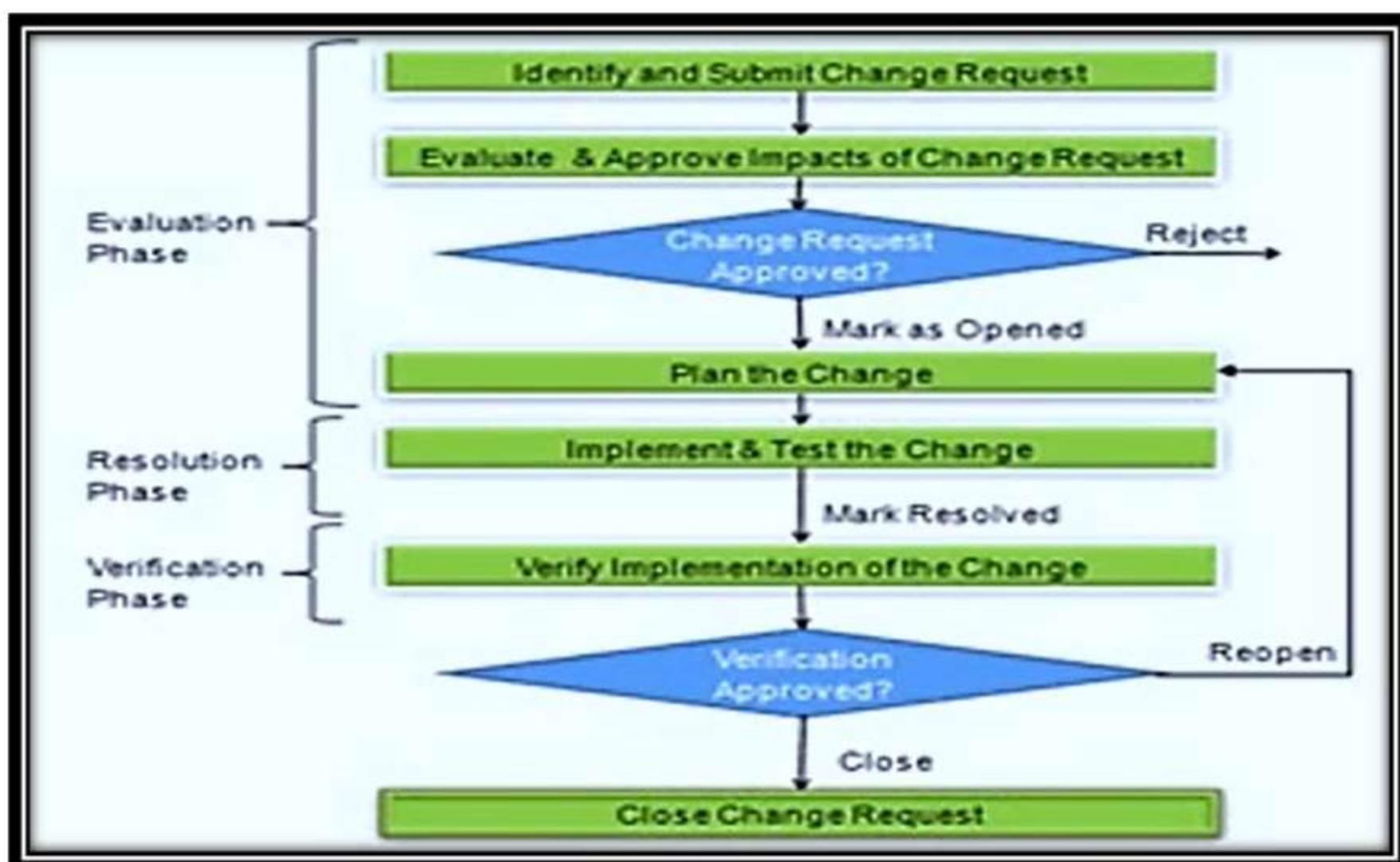
Change Control Board(CCB)

Change Control Board (CCB) or Software Change Control Board (SCCB)

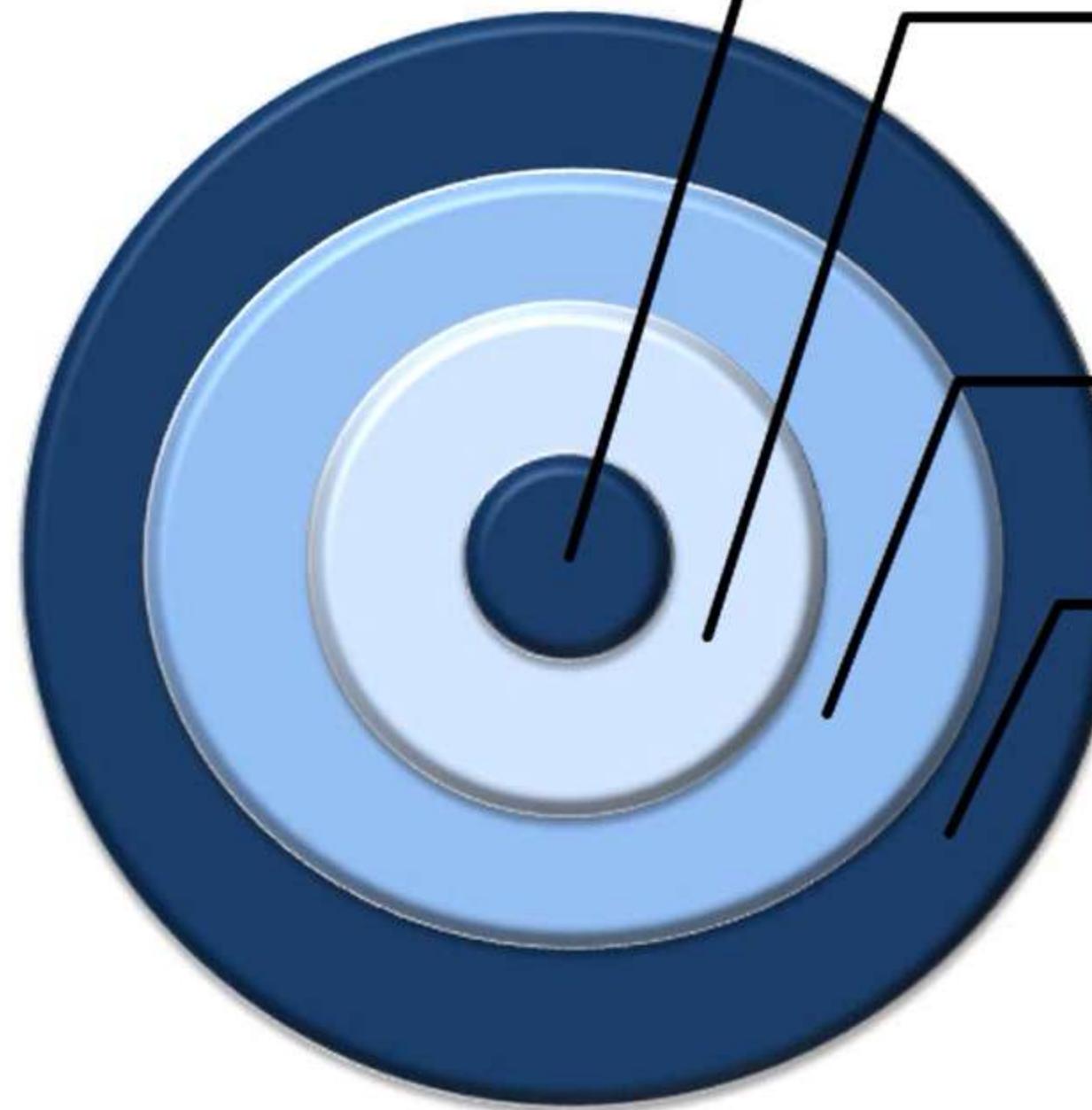
- is a committee that makes decisions regarding whether or not proposed changes to a software project should be implemented.
- is constituted of project stakeholders or their representatives.

The authority of the change control board may vary from project to project, but decisions reached by the change control board are often accepted as final and binding.

Change Control Process

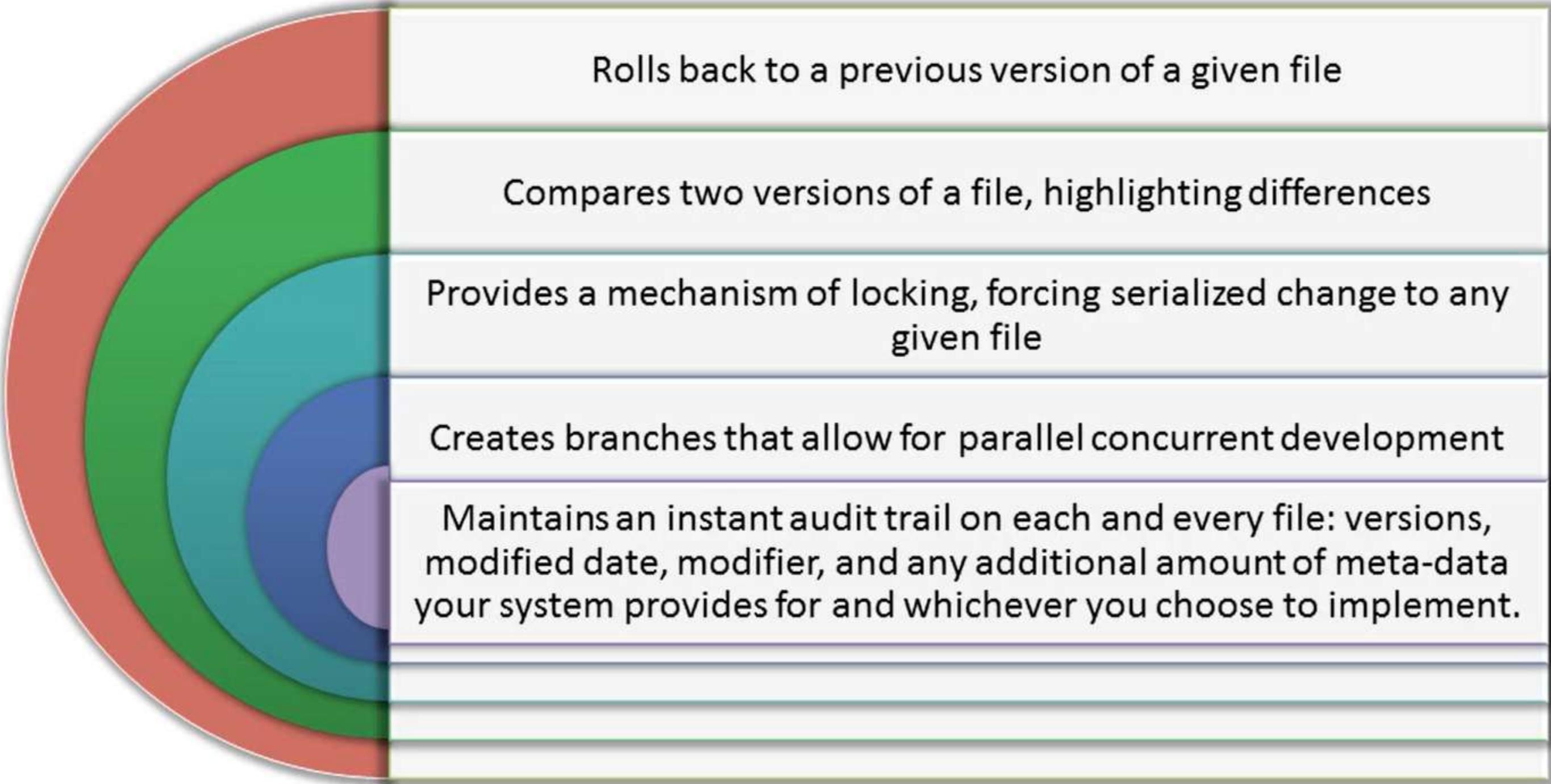


Version Management



- Version control is a mechanism used to manage multiple versions of computer files and programs
- It allows users to
 - lock files so they can only be edited by one person at a time
 - track changes to files
- Version Control allows you a place to store your work as it progresses, and allows you instant recall of any work from any point in time.
- It also allows others to recall communal (or just your) work at any time, in a read only fashion, or also to make changes themselves.

Benefits of Version Management



Summary

- Software configuration management
- Software Configuration Object
- Software Configuration Repository
- Check in check out process
- Change management
- Version management



AGILE METHODOLOGY



Objectives

In this module you will learn,

- Agile methodology
- Scrum
- Dynamic System Development Method
- Crystal
- Feature Driven Development
- Lean Software Development
- Extreme Programming



Agile Vs Waterfall



Agile Methodology

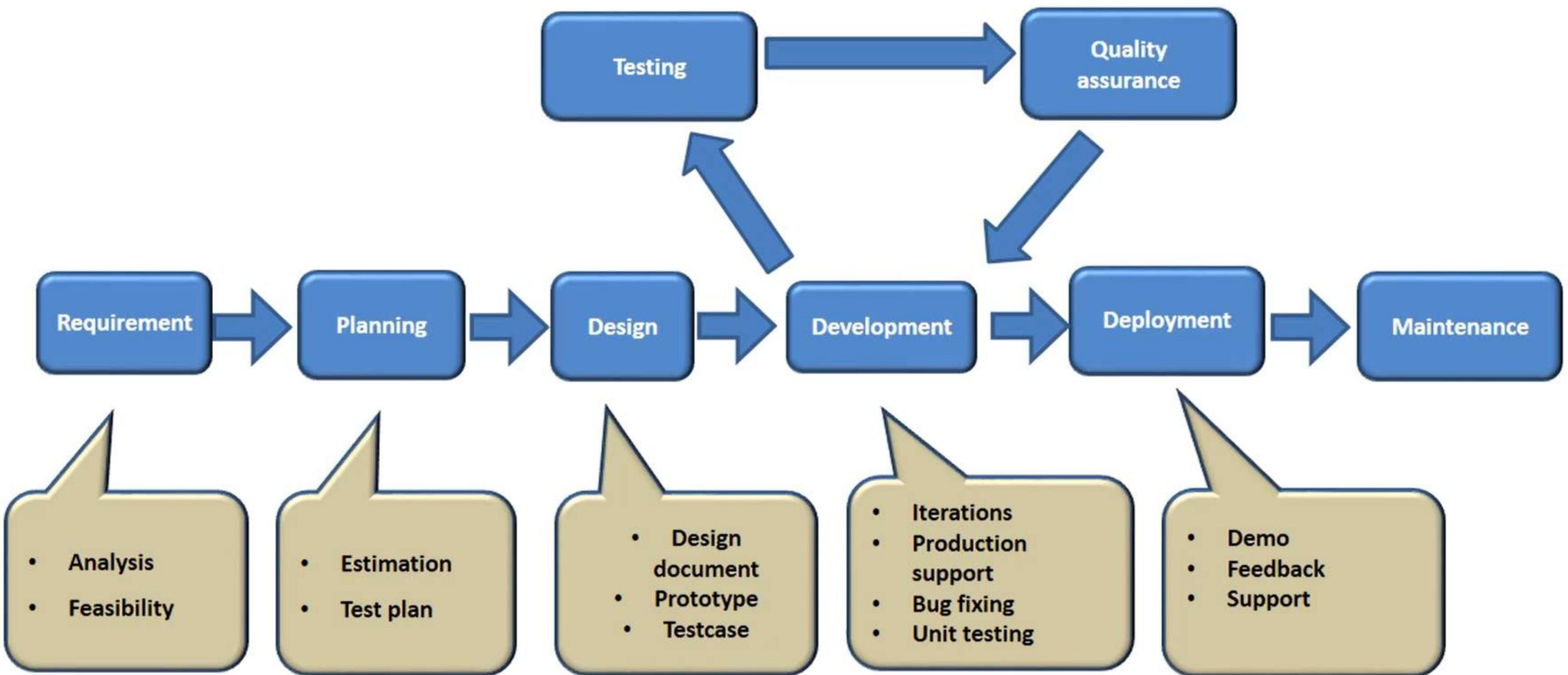
A conceptual framework where software is developed in iterations.

It satisfies the customer through early and continuous delivery of valuable software.

Useful when there is no clarity on the client's requirement or when the client frequently changes his requirement.

The development activities can be carried out using the iterative actions.

Agile Methodology



Agile Methodology Characteristics

Iterative and evolutionary development.

Many builds are delivered in the iteration process.

Accepts change of requirement at any stage.

Requires close communication among business group, development and testing people.

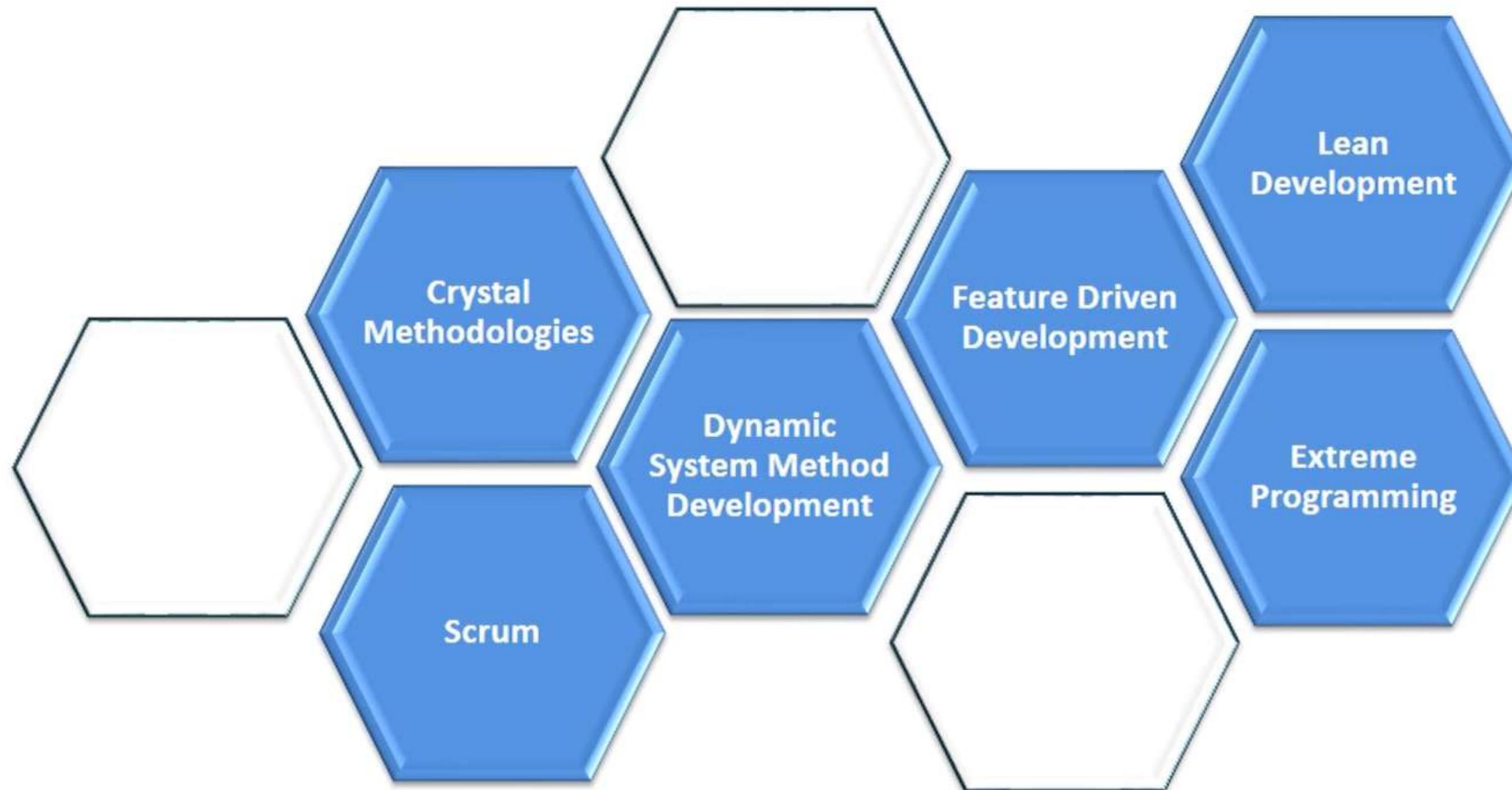
More focused on success than sticking with a plan.

Reduced risk and time to develop.

Less documentation work compared to other methodologies.

Requires continuous testing.

Methodology That Promotes Agility



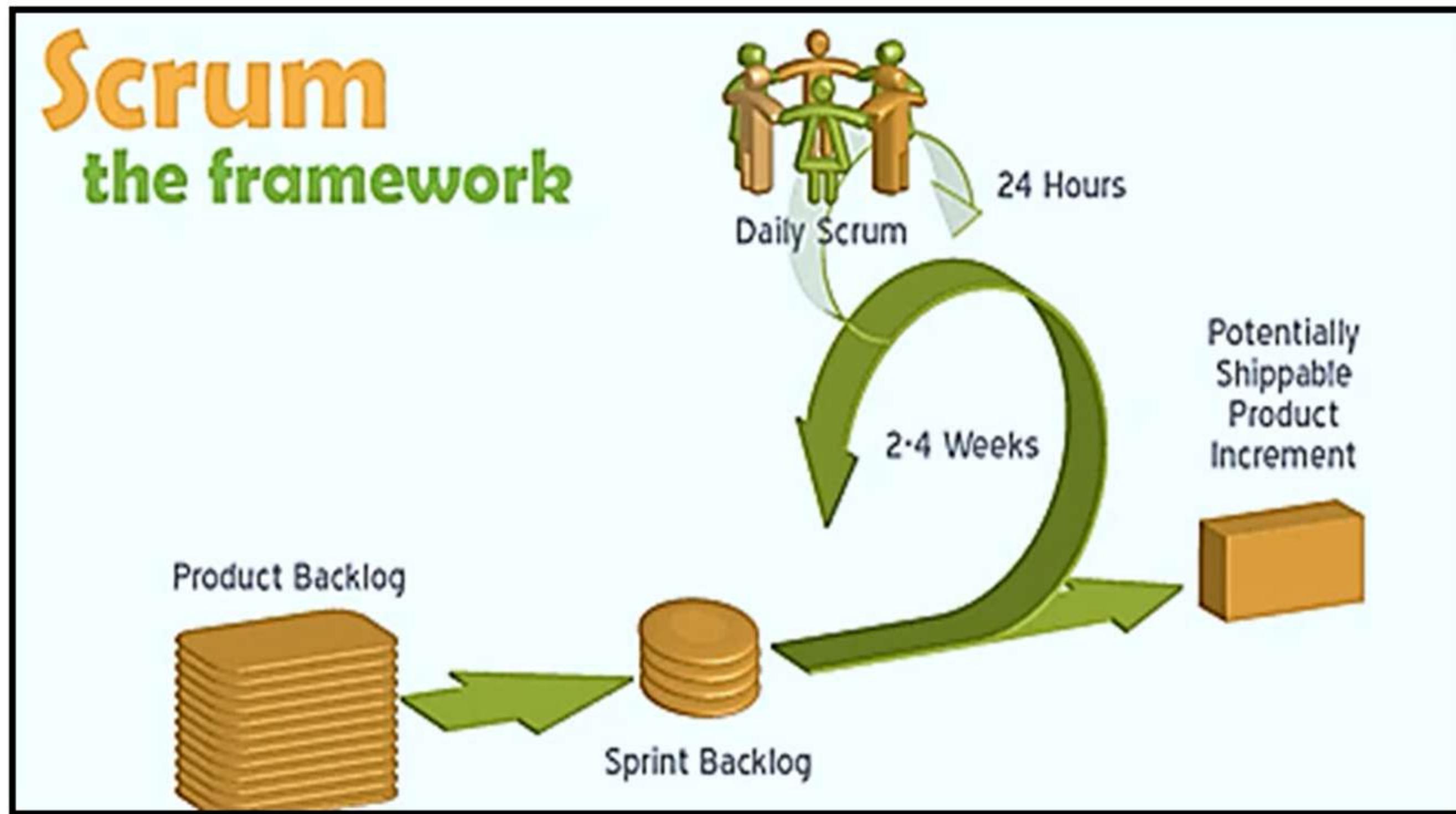
Scrum

Scrum divides the development into short cycles called sprint cycles. It advocates daily team meetings for coordination and integration.

The Roles of Scrum are as follows:

- **Product Owner**
- **Scrum Master**
- **Team Member**

Scrum



Limitations of Scrum

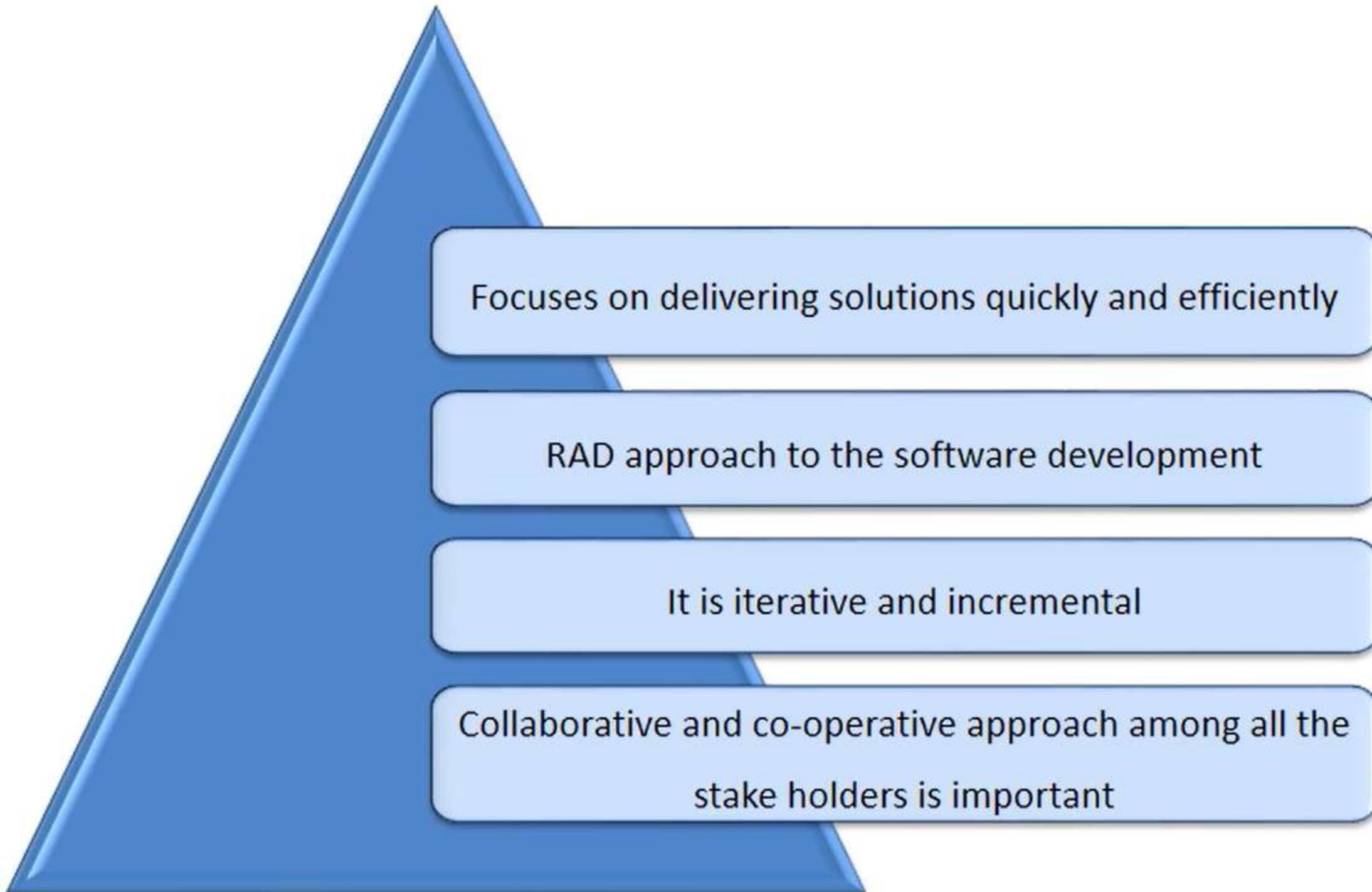
Increase in Cost

Increase in Time

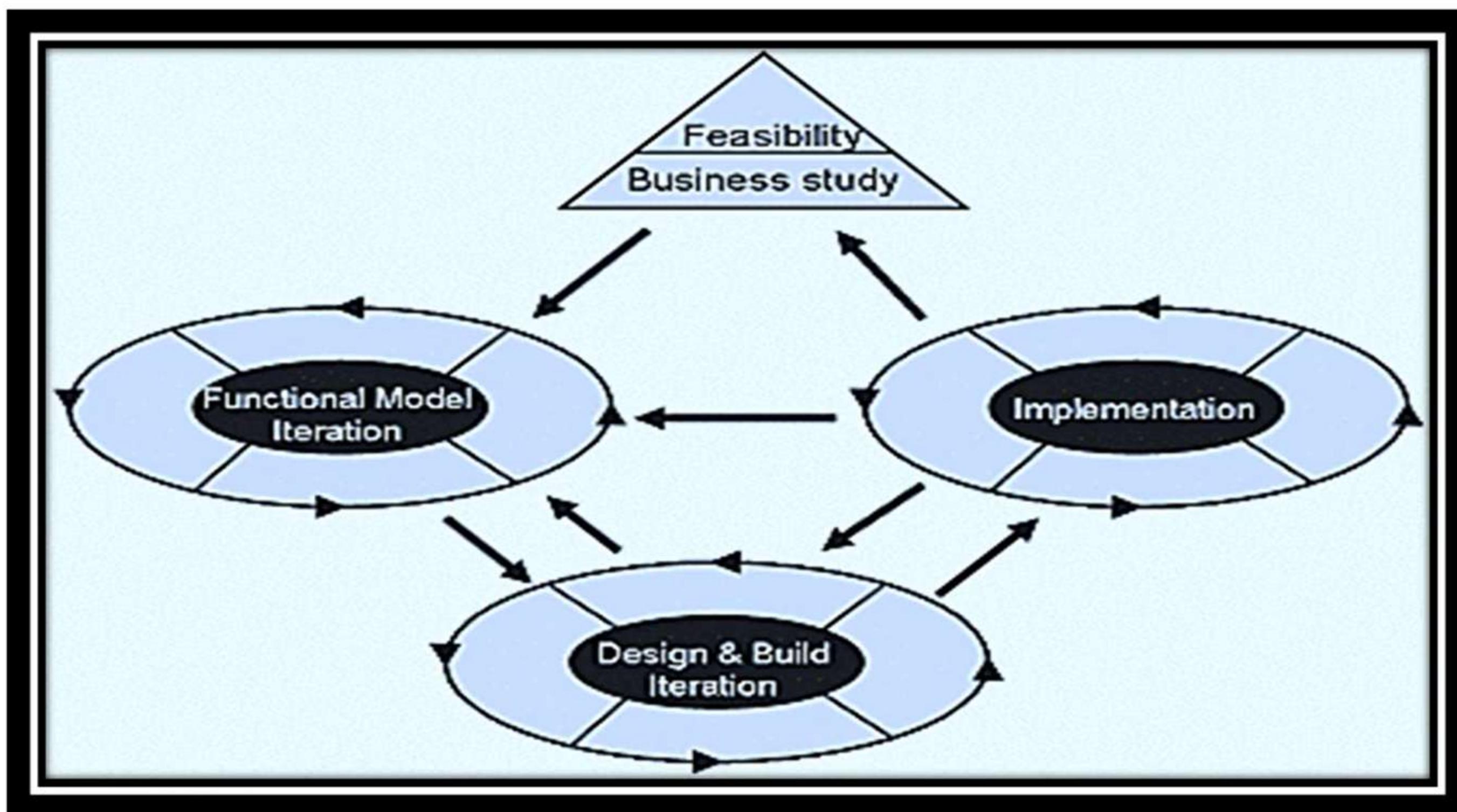
More useful for small and fast moving projects

Needs experienced team members

Dynamic System Development Method(DSDM)



Dynamic System Development Method(DSDM)



Eight Principles of DSDM



Roles in DSDM

Executive Sponsor

Visionary

Ambassador User

Advisor User

Project Manager

Technical Coordinator

Team Leader

Developer

Tester

Scribe Facilitator

Specialist Roles

Core Concepts of DSDM

Active user involvement is imperative.

All changes during development are reversible.

The focus is on frequent delivery of products.

Communication and collaboration among all the stakeholders is very important for the success of the project.

Business suitability is an essential criteria for acceptance of deliverables.

Requirements are the baseline at a high level.

Iterative and incremental development is necessary to converge on an accurate business solution.

Testing is integrated throughout the life cycle.

Advantages DSDM

Users are highly involved in the development of the system

Basic functionality of the software is delivered quickly

On time delivery of the software

Drawbacks of DSDM

Training is required for both the developers and the users

Difficulty in understanding the model

Crystal Methodology

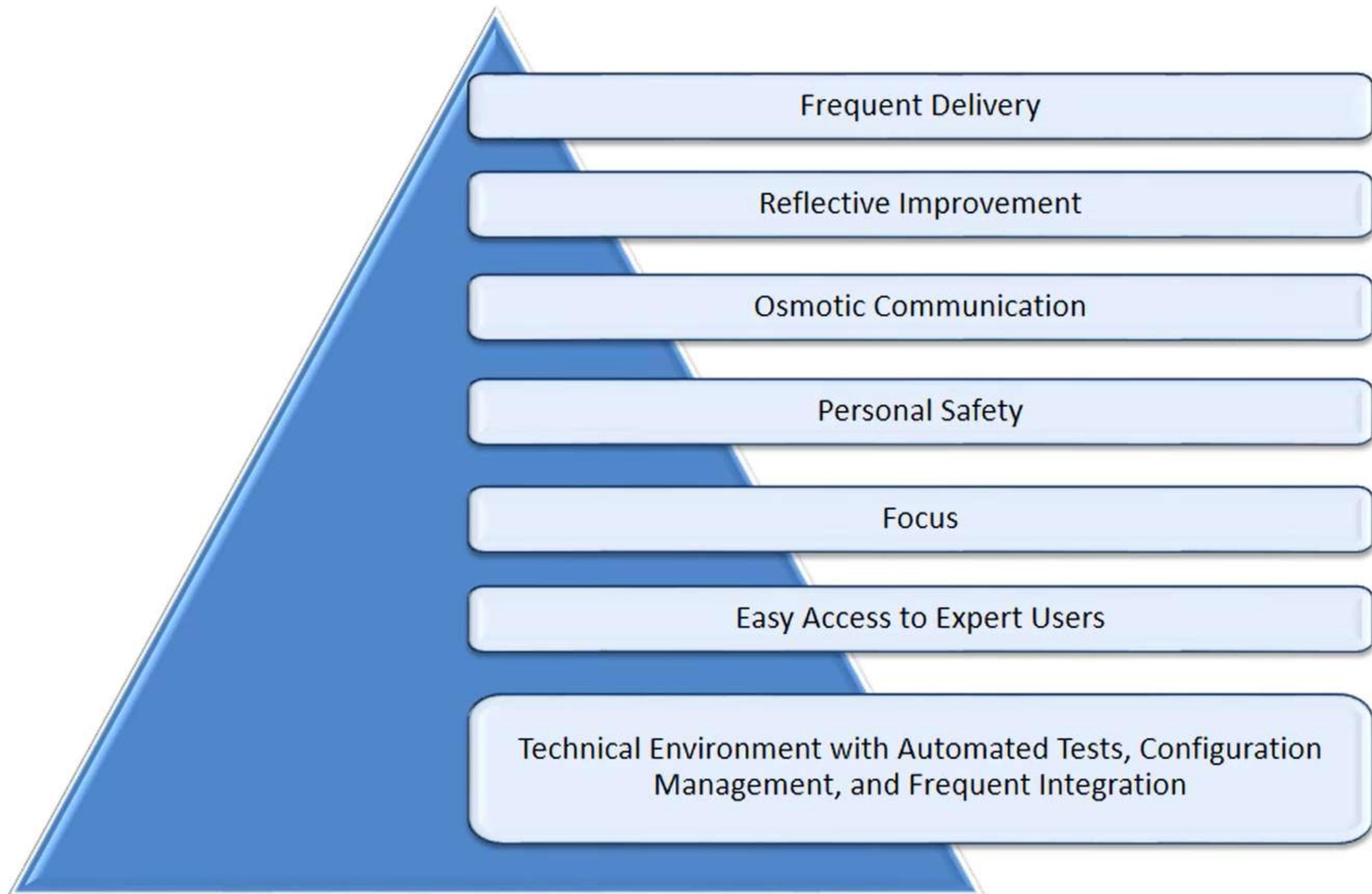
The word "Crystal" refers to the degree of hardness and the different colours of the methodology

The degree of hardness pertains to the use of rigor and ceremony

The colour of the methods are as follows:

- Clear, Yellow, Orange, Orange Web, Red, Magenta, and Blue.
- The colour is concerned with the “heaviness” of the project
- The lighter the colour the fewer the number of people on the project, whereas the darker colours indicate the need for more resources.

Seven Properties of Crystal Methodology



Feature Driven Development (FDD)



A short iteration framework for software development

Focuses on building an object model, build feature list, plan by feature, design by feature, and build by feature

Emphasizes Quality at each step

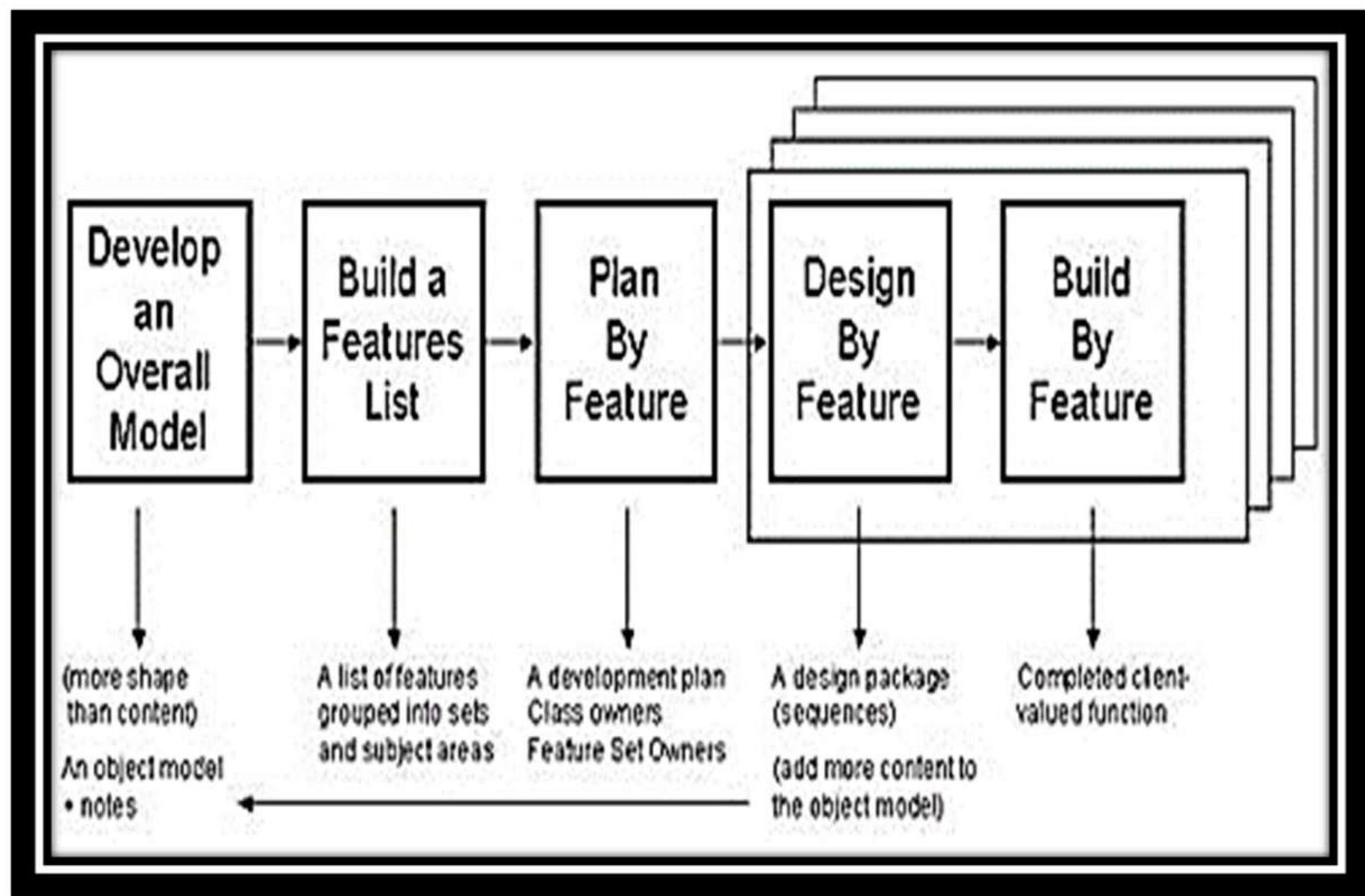
Delivers frequent, tangible, working results

Offers accurate and meaningful Project Progress Tracking

Combines many of the best practices of other agile models

Puts less focus on initial design and quickly gets to the point where the team can deliver new functionality to the project feature by feature

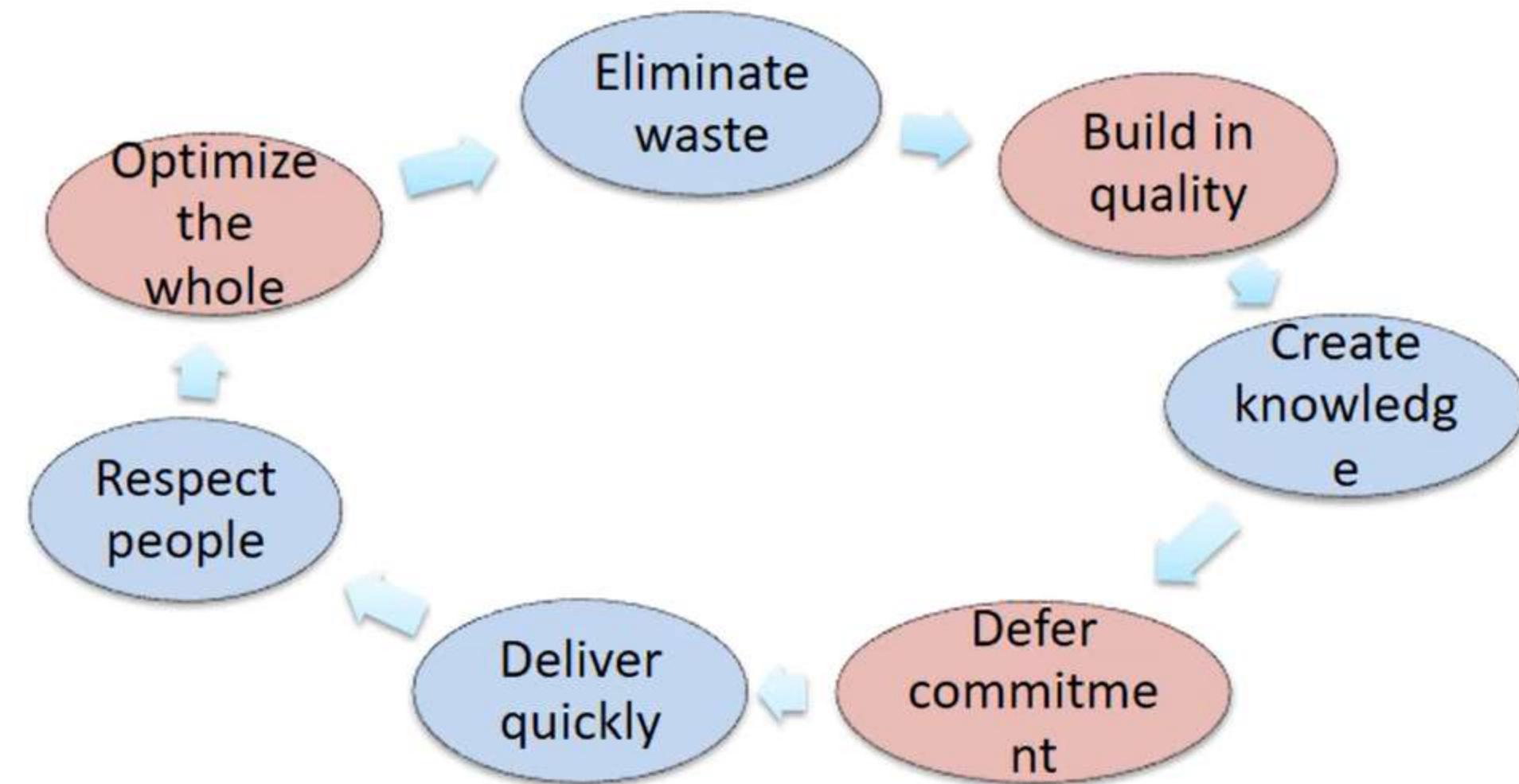
Feature Driven Development (FDD)



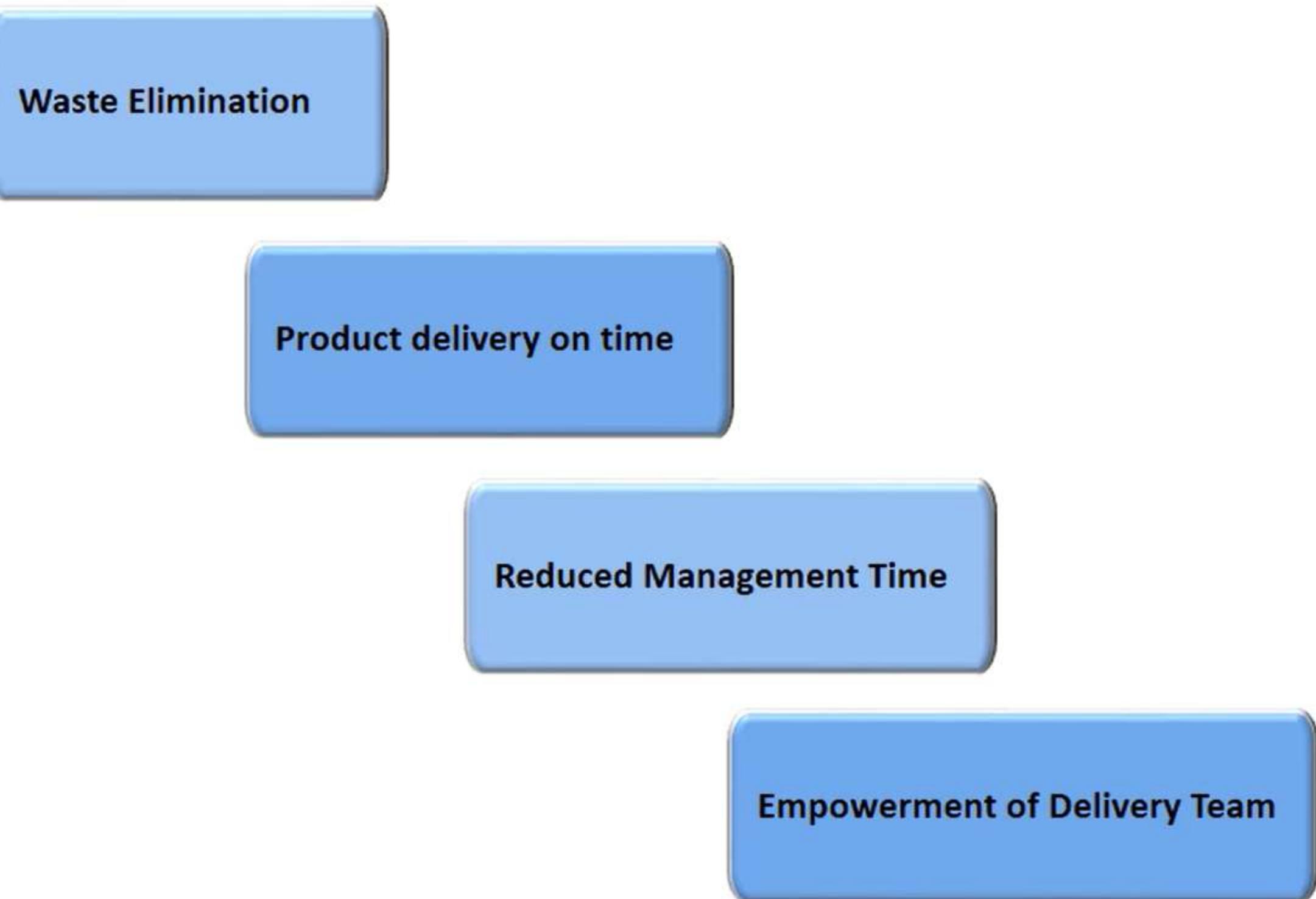
Lean Software Development

Lean software development is the application of lean manufacturing principles when developing software.

Principles of Lean software Development:



Advantages of Lean Software Development



Limitations of Lean Software Development

Project is highly dependent on

- Team Cohesiveness
- Exceptional Technical skill of Team Members

Decisions are made when required. So difficult to take tough decisions

SRS will evolve. But if changes are more, developers lose sight of the original objectives of the project

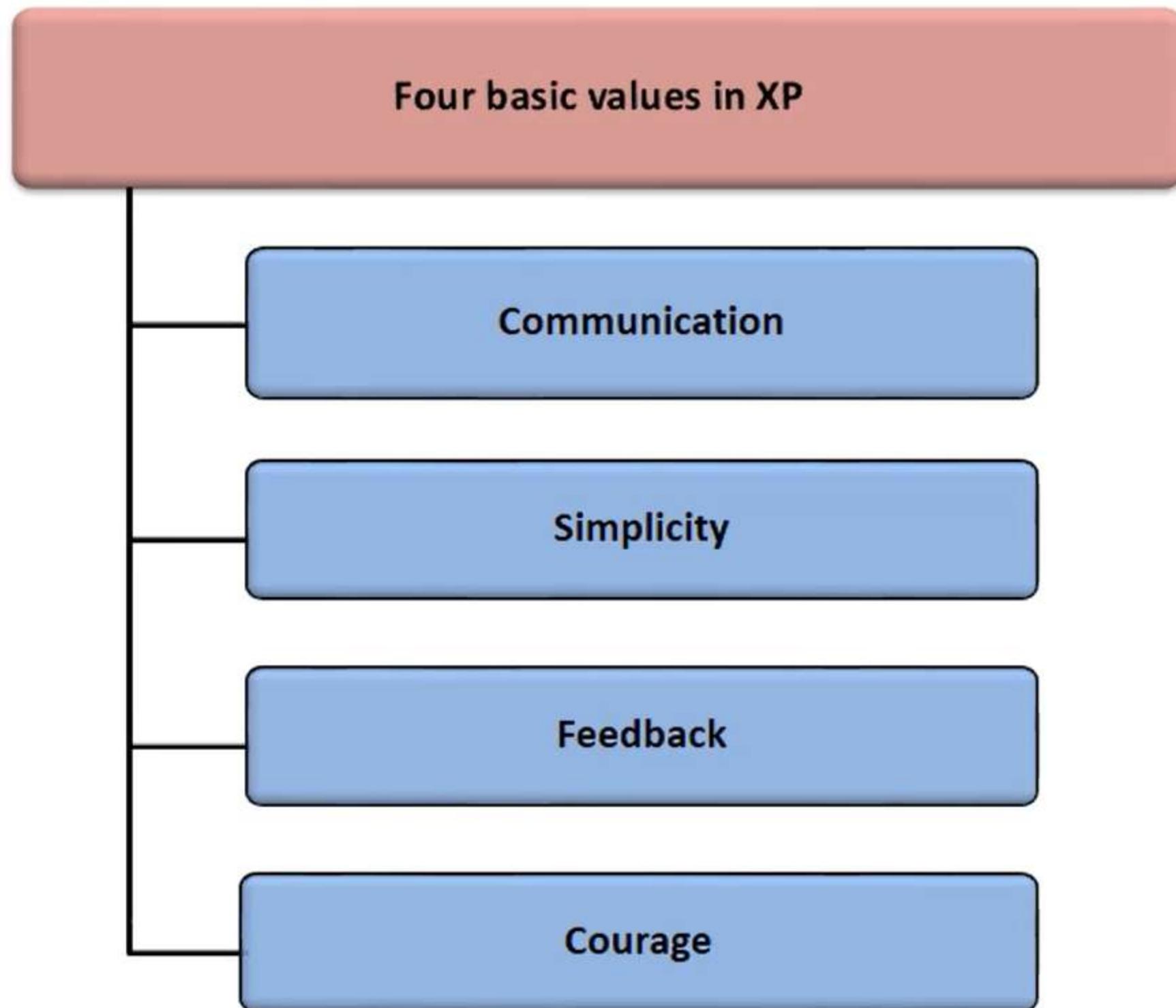
Extreme Programming (XP)



XP is a lightweight (agile) process

- Instead of excessive documentation, XP gives more importance to customer feedback
- Embraces change : iterates often design and redesign, codes and tests frequently, keeps the customer involved
- Delivers software to the customer in short (2 week) iterations
- Eliminates defects early, thus reducing costs

Extreme Programming (XP)



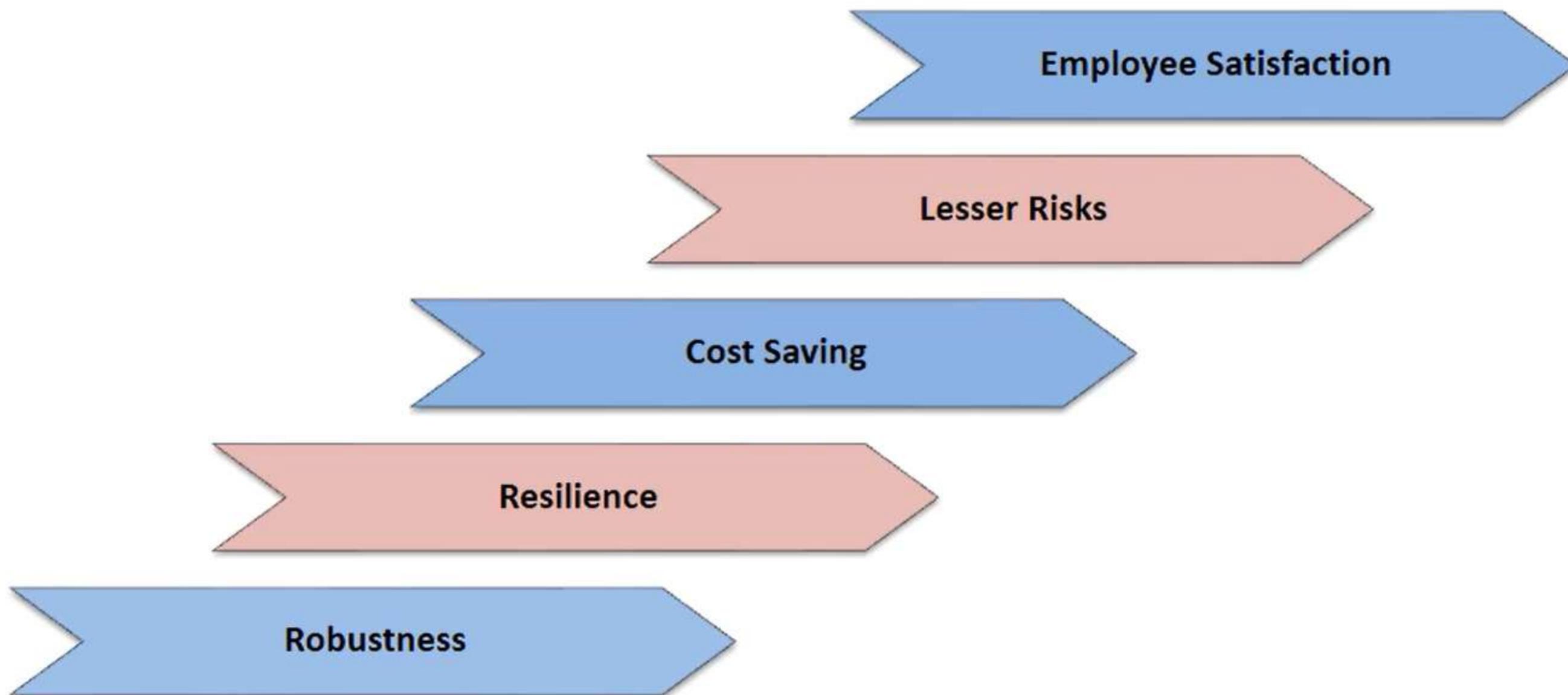
Principles of Extreme Programming



Principles of XP

- Rapid feedback
- Assume Simplicity
- Incremental Changes
- Embrace Change
- Quality Work

Advantages of XP



Limitations of XP

XP is geared towards a single project, developed and maintained by a single team.

XP will not work in an environment where:

- a customer or the manager insists on a complete specification or design before they begin programming
- programmers are separated geographically

XP has not been proven to work with systems that have scalability issues

Summary

- Agile methodology
- Scrum
- Dynamic System Development Method
- Crystal
- Feature Driven Development
- Lean Software Development
- Extreme Programming



OBJECT ORIENTED PROGRAMMING



Overview



I need to know the address of a loan customer. Ok let me go and take the customer file.

Miss Jenifer, how can you take a customer file without my permission?



Overview

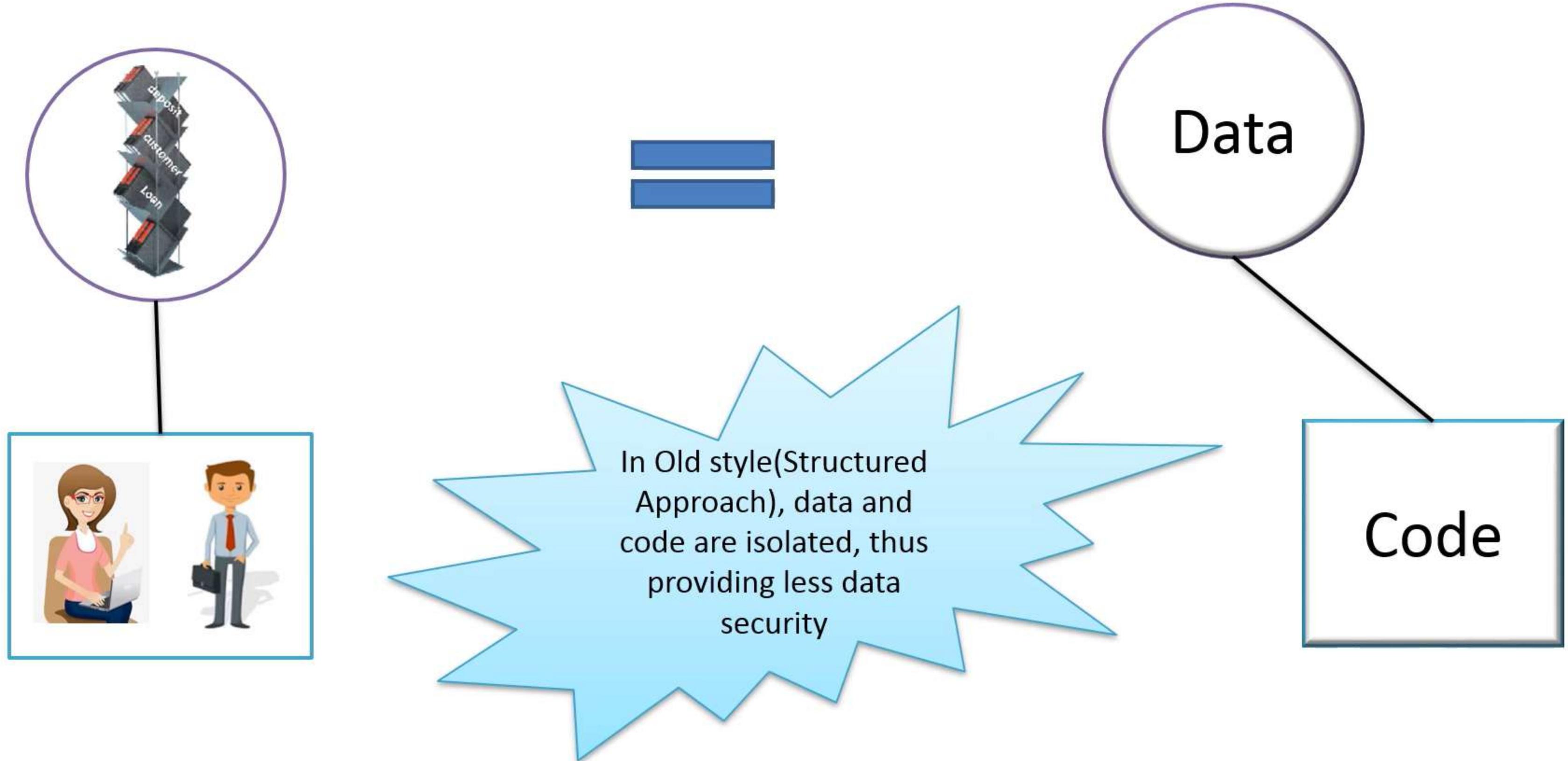


Yes Jenifer. here's the
key. You can take it

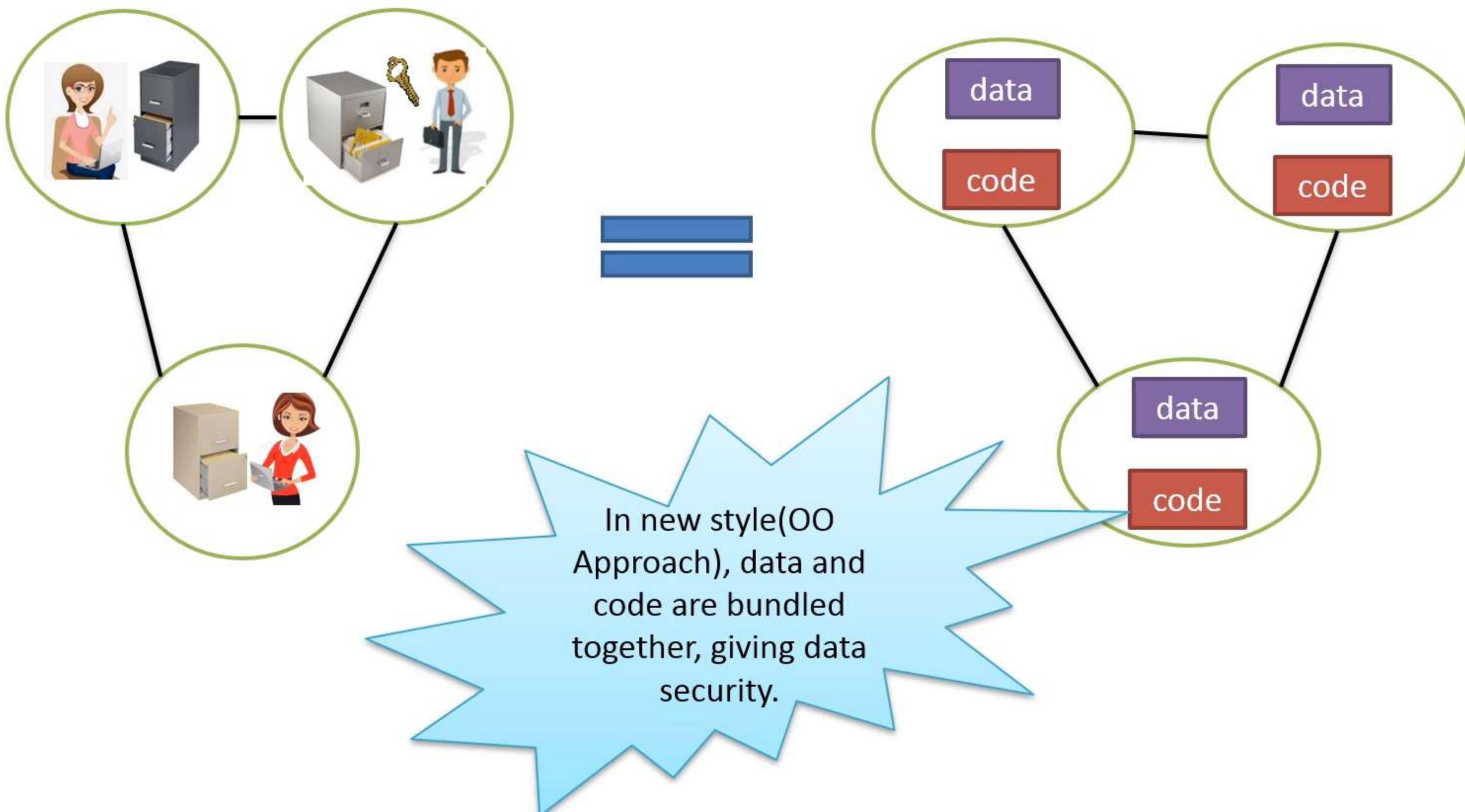


I need Mr.Tom's contact number.
John, could you give me the
customer cupboard key? I need to get
some details.

Overview



Overview



Objectives

- Class
- Object
- Encapsulation
- Abstraction
- Polymorphism
- Hierarchy
- Modularity
- Typing
- Persistent



Object Oriented Concepts

An Object Oriented program consists of many well encapsulated objects, and interacts with each other by sending messages

The object-oriented languages focus on components that the user perceives, with objects as the basic unit.

Benefits

Ease in software design as you could think in the problem space rather than the machine's bits and bytes.

Ease in software maintenance: object-oriented software are easier to understand, therefore easier to test, debug and maintain.

Reusable software: you don't need to keep re-inventing the wheels and re-write the same functions for different situations.

Object Oriented Concepts

The basic unit of OO Approach is

- Class
- Object
- messages

TeamLead : Richard



Object

Web Programmer : Hawking



Message
passing

Object

Class

Classes are the fundamental building blocks of a Java program

A class is a definition of objects of the same kind.

Class can be defined as a blueprint, template or a prototype that defines and describes the **attributes** and **behaviors** common to all objects of the same kind.

STUDENT

Rollno
Name
Class

setRollNo()
assignClass()
study()

Attribute of a class

Attribute is a named property of a class describing a range of values that instances of the class may hold for that property.

An attribute has a type and only the holding object is able to change the values of its own attributes.

The set of attribute values defines the state of the object



27, 5th street,
T. Nagar,
Chennai

My address

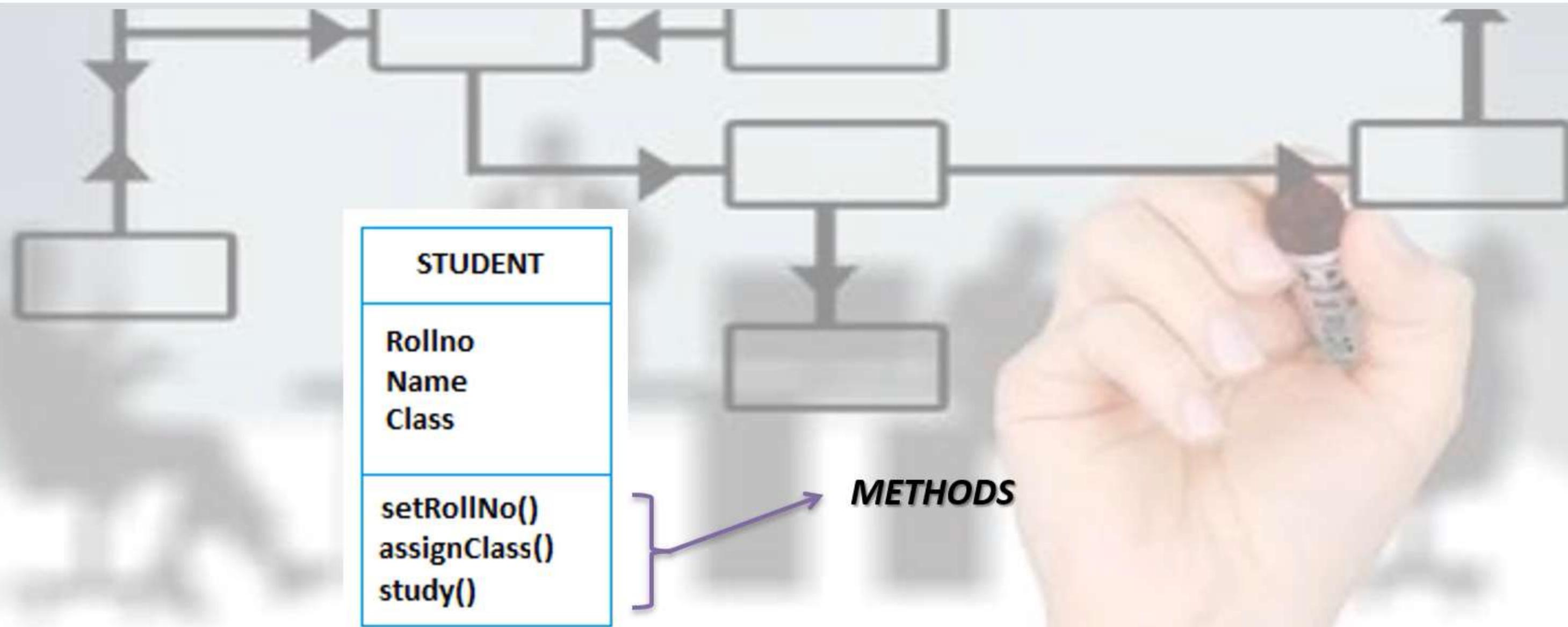
ATTRIBUTES

STUDENT
Rollno
Name
Class
setRollNo()
assignClass()
study()



Methods of Class

Methods are members of a class that provide a service for an object or perform some business logic.



Object

An object is an instance of a class.

Objects are the real time entity which are created through their template, their class

Object is dynamic



STUDENT
Rollno
Name
Class
<code>setRollNo()</code>
<code>assignClass()</code>
<code>study()</code>

Properties of Object

- **State :** An object's state is defined by the attributes of the object and by the values they have.
- **Behavior :** Behavior is how an object acts and reacts in terms of its state changes and message passing
- **Identity :** Identity is that property of an object which distinguishes it from all others



Principles of OOP

Abstraction

Encapsulation

Polymorphism

Hierarchy

Modularity

Typing

Persistence

Principles of OOP

In Real Life...



Abstraction



Encapsulation



Inheritance

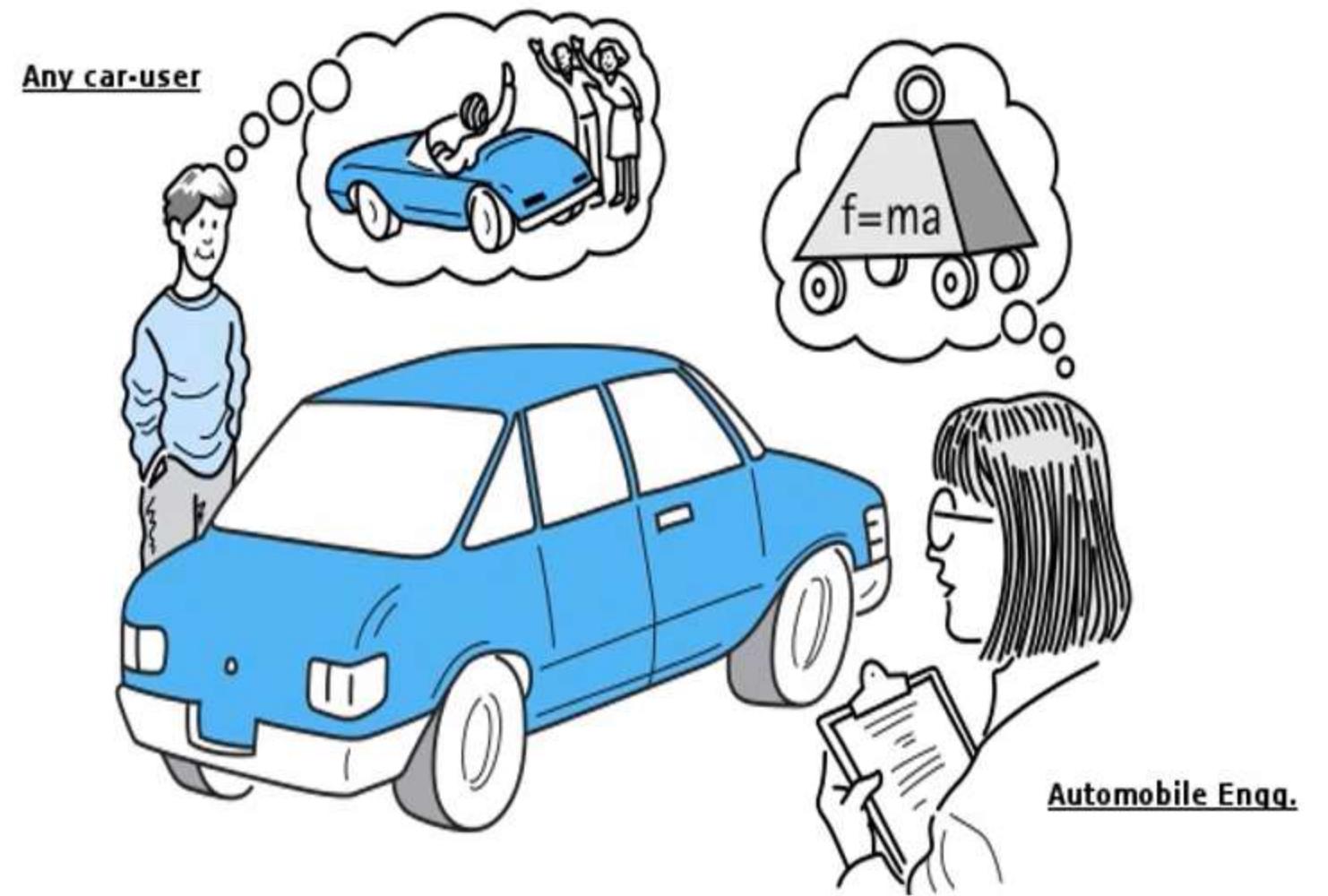


Polymorphism

Abstraction

Abstraction includes the essential details relative to the perspective of the viewer

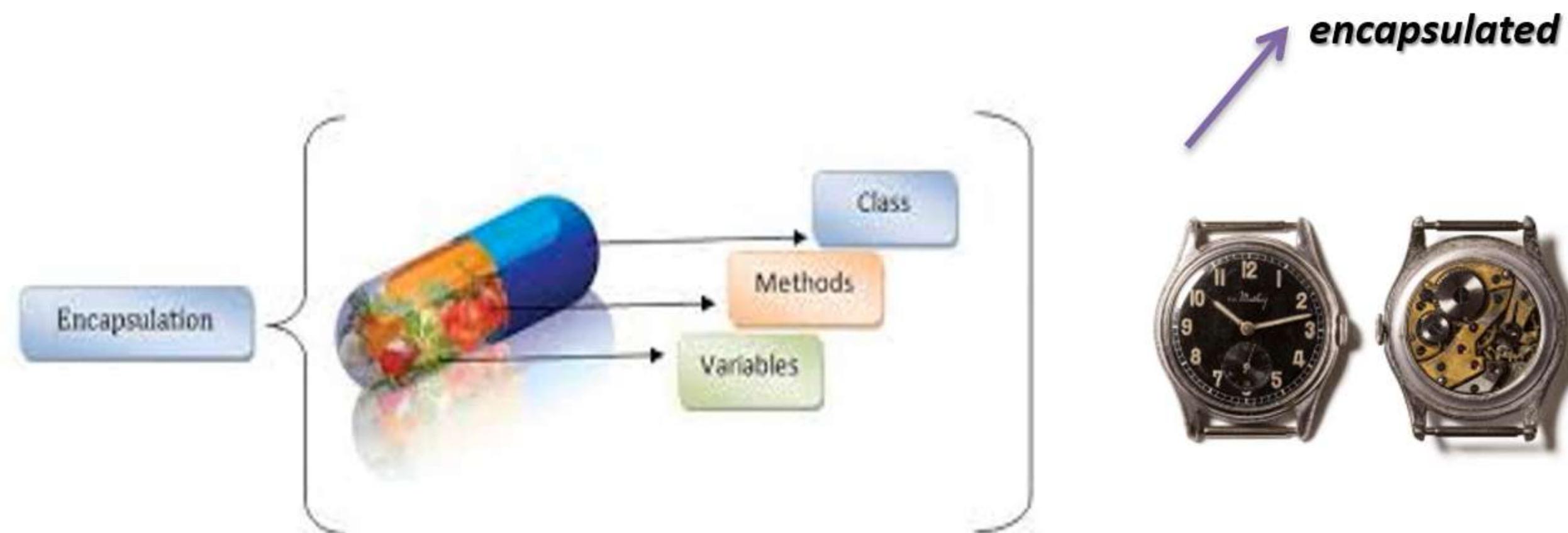
Abstraction allows us to manage complexity by concentrating on the essential aspects making an entity different from others.



Encapsulation

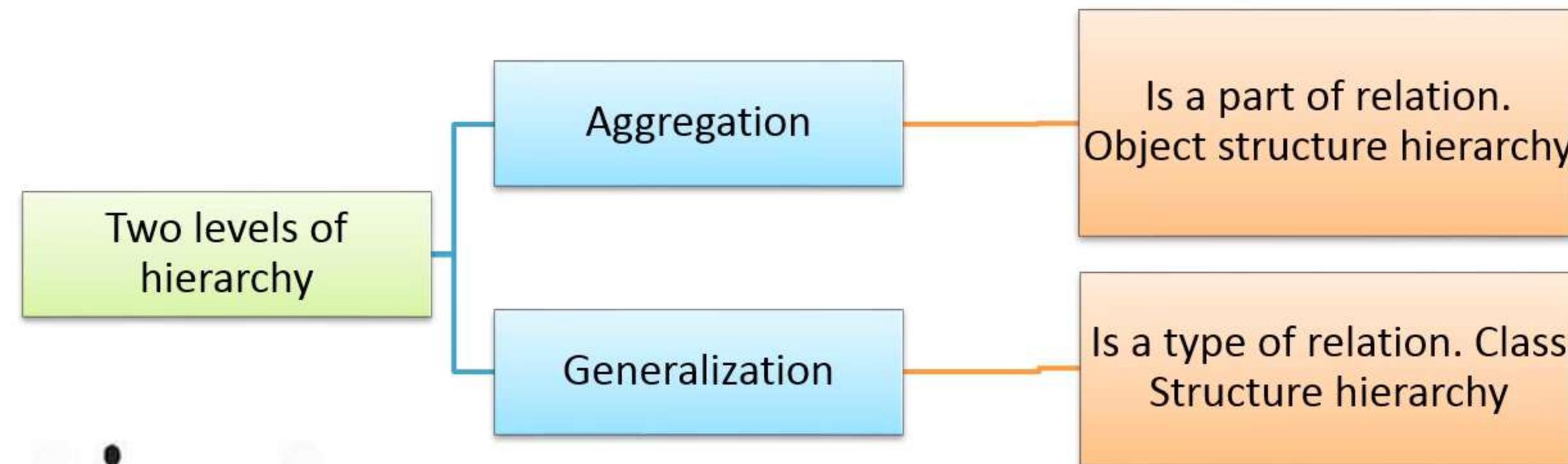
Encapsulation is the process of binding data and functions together into a single functional unit.

Provides data security using information hiding



Hierarchy

Hierarchy is ranking or ordering of abstractions.



Generalization

Objects of the real world often form conceptual hierarchies.

Ordering of abstractions into a tree-like structure.

The parent class defines the common properties of one or more child classes.

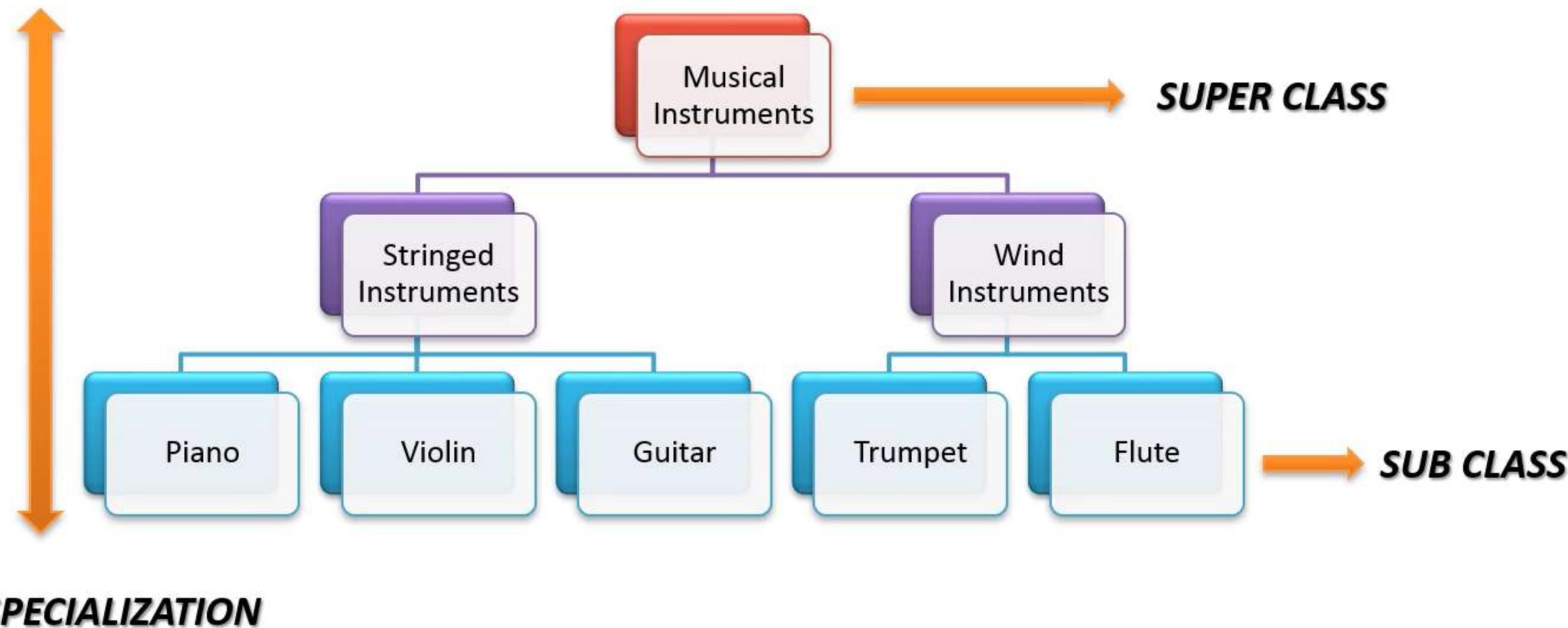
The child classes derive the properties of the parent and also have specific properties of their own

A parent class itself can have another parent and hence forming a hierarchy.



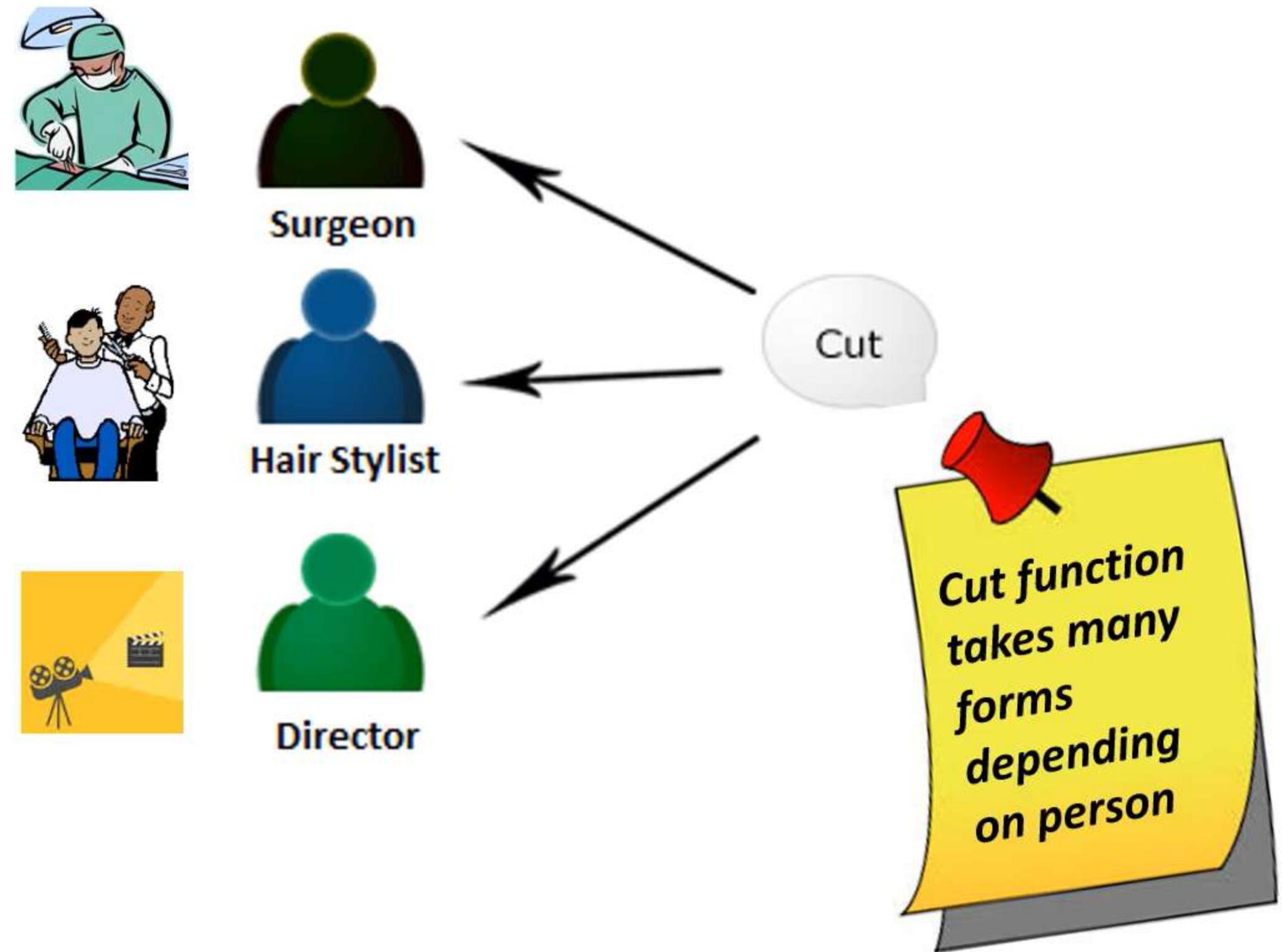
Generalization

GENERALISATION



Polymorphism

Taking many forms



Polymorphism

Having multiple forms

Refers to a **programming** language's ability to process objects differently depending on the context

Polymorphism
could be

Overloading - Static polymorphism in simple words means two methods having the same method name, but taking different input parameters.

Overriding - derived class is implementing a method of its super class.

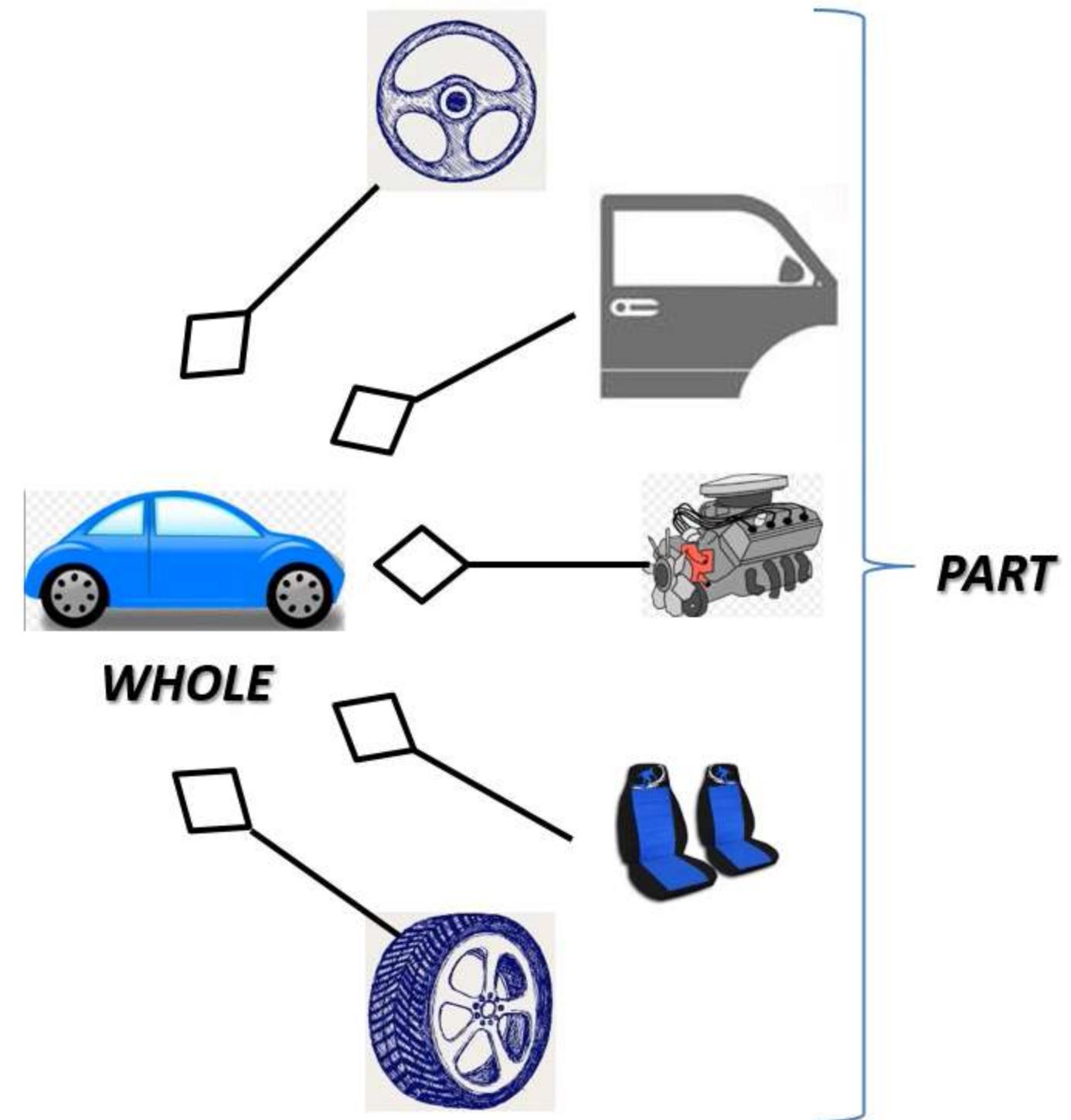
Aggregation

Aggregation represents a “whole-part or a-part-of” the relationship.

When an object has another object within it, it is a ‘has-a’ relationship, called aggregation

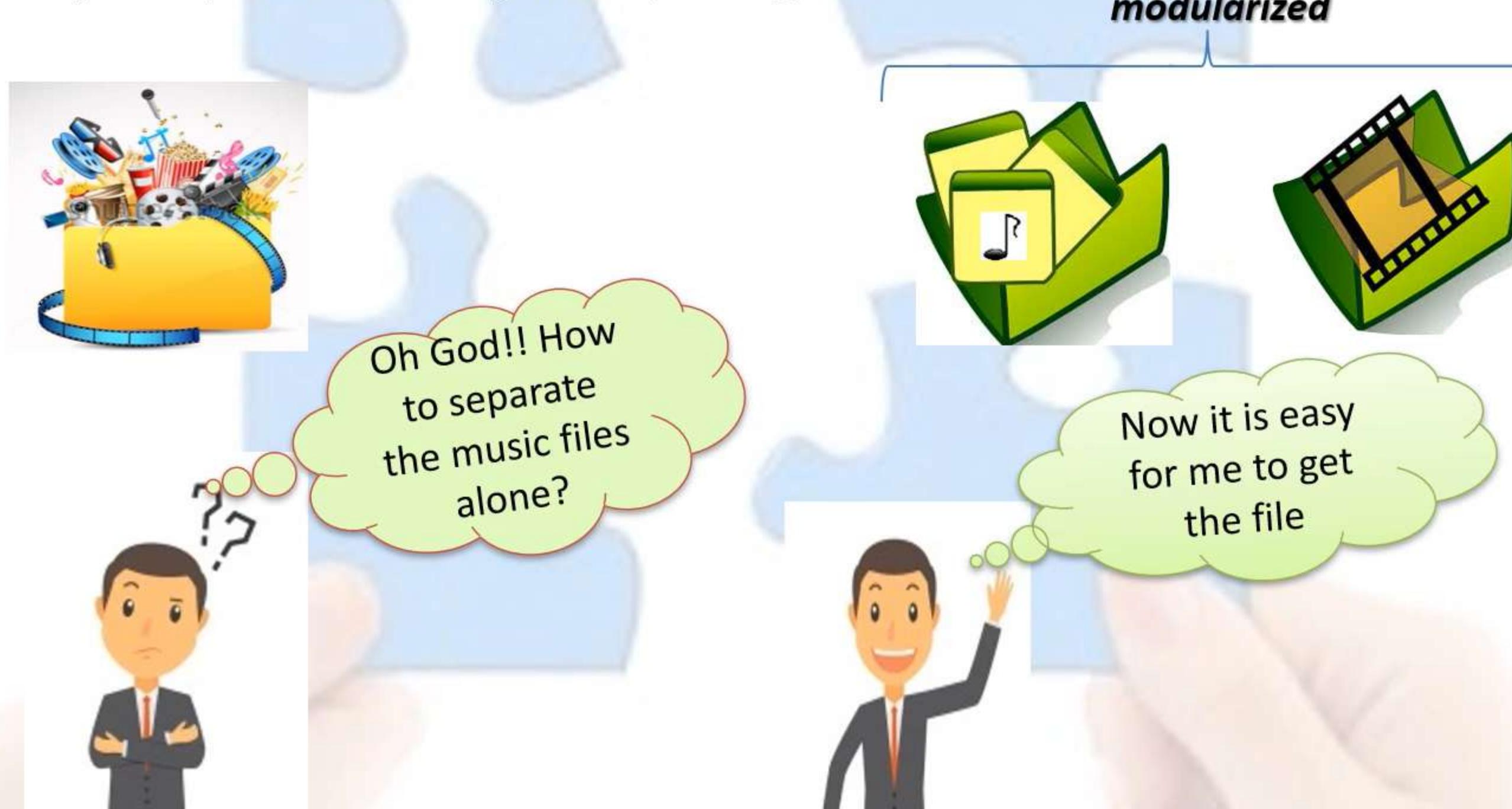
When the whole Object is destroyed the part object can still be used

Stronger form of aggregation is composition
Eg: House and room



Modularity

- Modularity is a method to divide a program into smaller units
- Modularity is implemented in java as packages



Typing

Identifies a programming language type conversion and characteristics.

It can be either strongly or weakly typed

Strong typing requires explicit conversions for conversions between different data types in expressions.

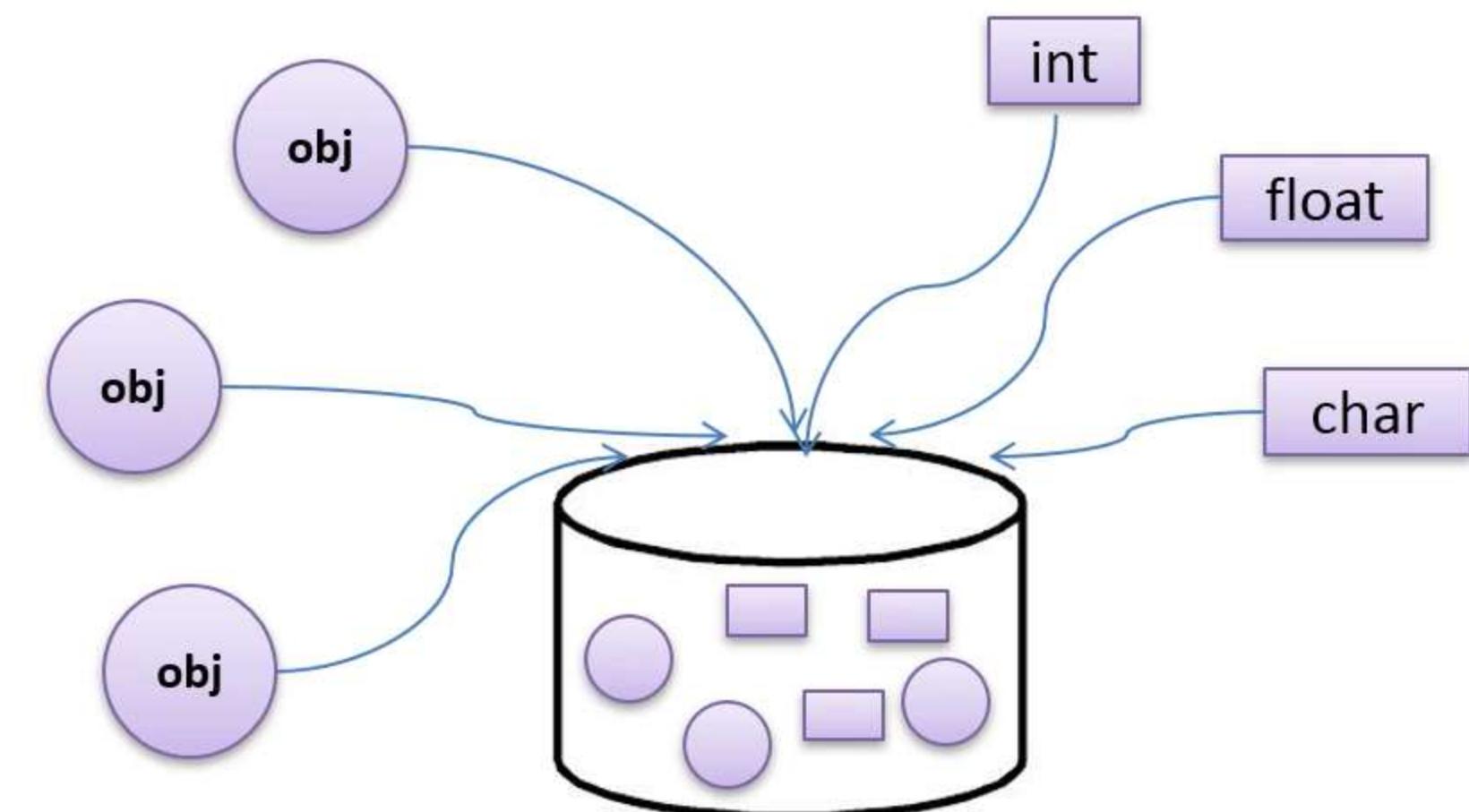
Typing is inherently Language Implementation Dependent



Persistence

Persistence is the ability of an object or data to outlive the lifetime of the program that creates, accesses or modifies it.

A persistent object is one that continues to exist after a program that creates or uses it, terminates.



Few more concepts of OOP

Static

Interface

Abstract
class

Static



Exactly! What do you infer from this scenario?
Certain things are available for each individual apartment, whereas certain things are common for all apartments.

Lift, Generator, Gym,
Swimming Pool



Static - Overview

Likewise, in an object oriented way , certain attributes will be applicable for each instance of that class and some attributes will be common to all instances in a class. These common attributes are termed static.



CompanyName="Teknoturf"

Abstract class

Methods that do not have any implementations are called as abstract methods.

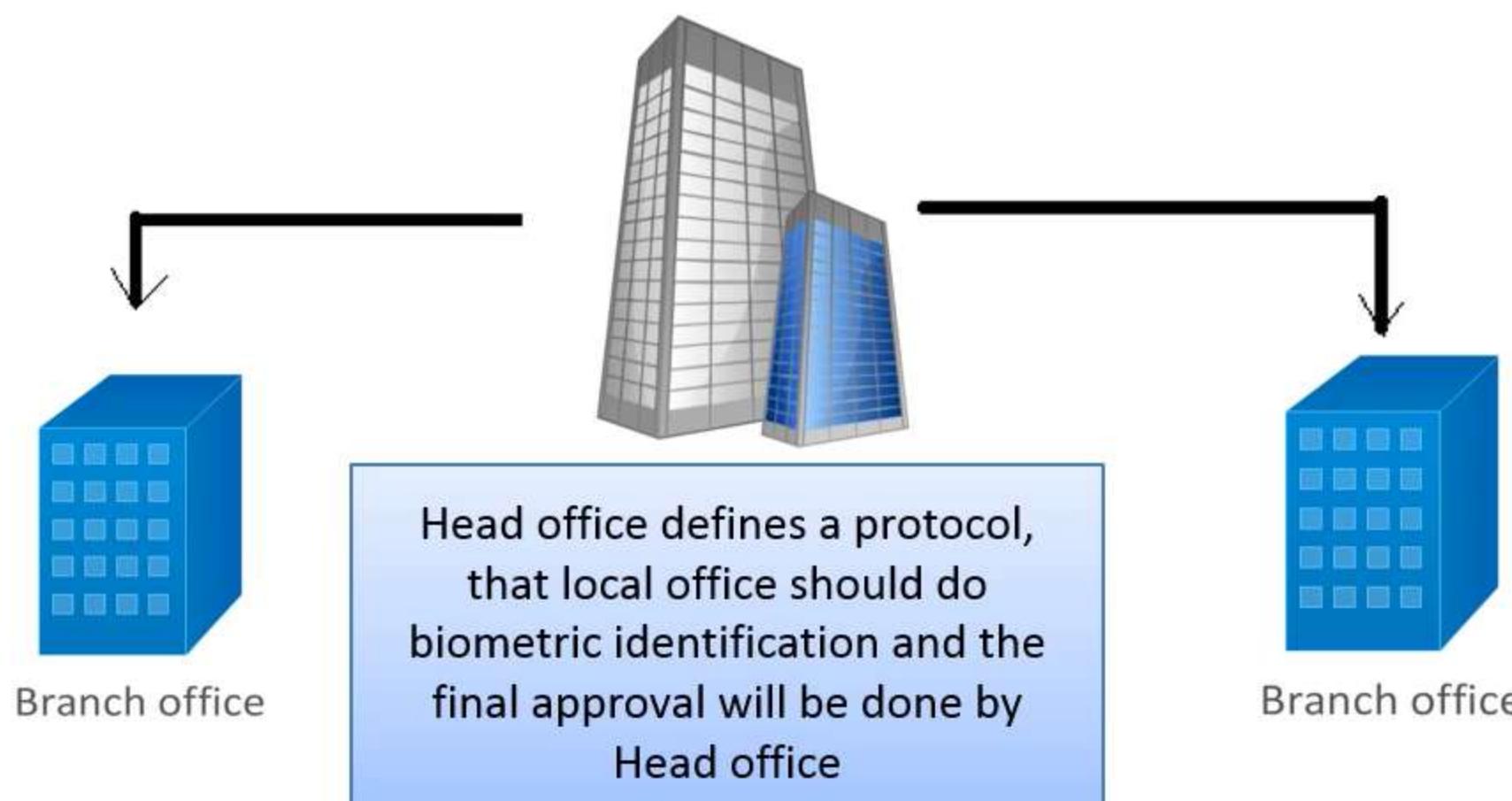
Abstract class may contain zero or more abstract methods

Abstract class cannot be instantiated

Abstraction- Real world scenario

Let us consider the Aadhaar card issuing process. The head office for the final approval is present in Bangalore and we have two local offices one at Chennai and the other at Hyderabad. Aadhaar card issuing involves 2 steps. Biometric identification and the other is final approval and issuing the acknowledgement. Chennai does the biometric identification through the finger print and Hyderabad does it through retinal scan. So the Bangalore head office has decided to delegate the biometric identification to both the local offices and take care of the final approval and issuing the acknowledgement

Chennai office adheres to the head office and does finger print identification; and sends the documents to the head office for the final approval.



Hyderabad office adheres to the head office and does retinal scan identification; and sends the documents to the head office for the final approval.

Interface

Interface is a contract for the classes to implement the behavior in their own way.

All methods inside the interface are public and abstract



Interfaces

RBI fixes the base interest for Home Loan.

All banks calculate the interest for home loan based on some specific factors of theirs.

Here RBI acts as an interface with the method calculateHomeLoanInterest as abstract

All banks implement this interface and give implementation for this abstract method



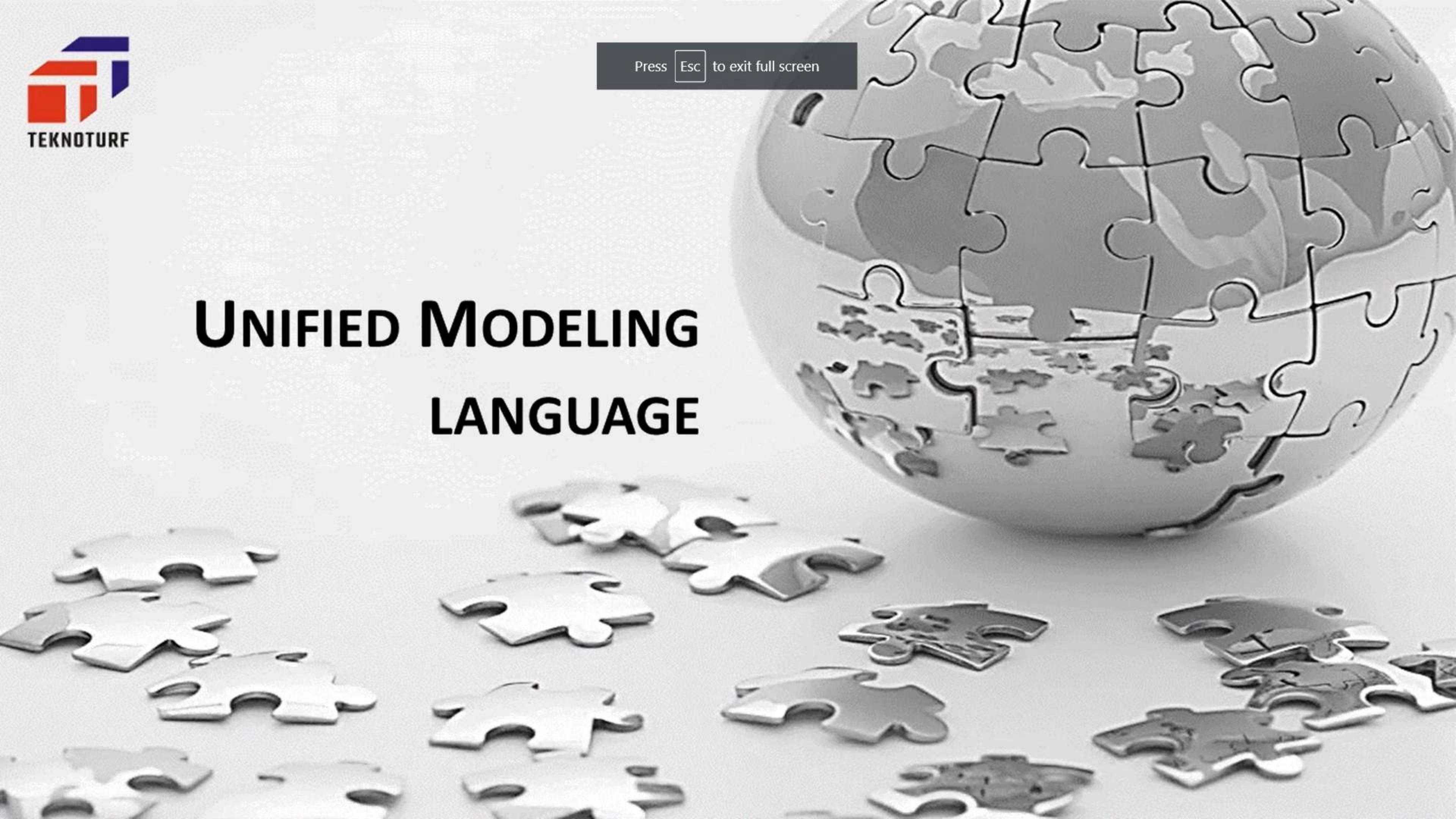
Summary

- Class
- Object
- Encapsulation
- Abstraction
- Polymorphism
- Hierarchy
- Modularity
- Typing
- Persistent



Press Esc to exit full screen

UNIFIED MODELING LANGUAGE



Objectives

- UML Architecture
- Use case Diagram
- Class Diagram
- Sequence Diagram
- Activity Diagram
- State chart Diagram



Modeling

A way of thinking about the problems using models organized around the real world ideas.

A modeling method comprises of a language and a procedure for using the language to construct models.

It is the way to visualize your design and check it against requirements before your crew starts to code.

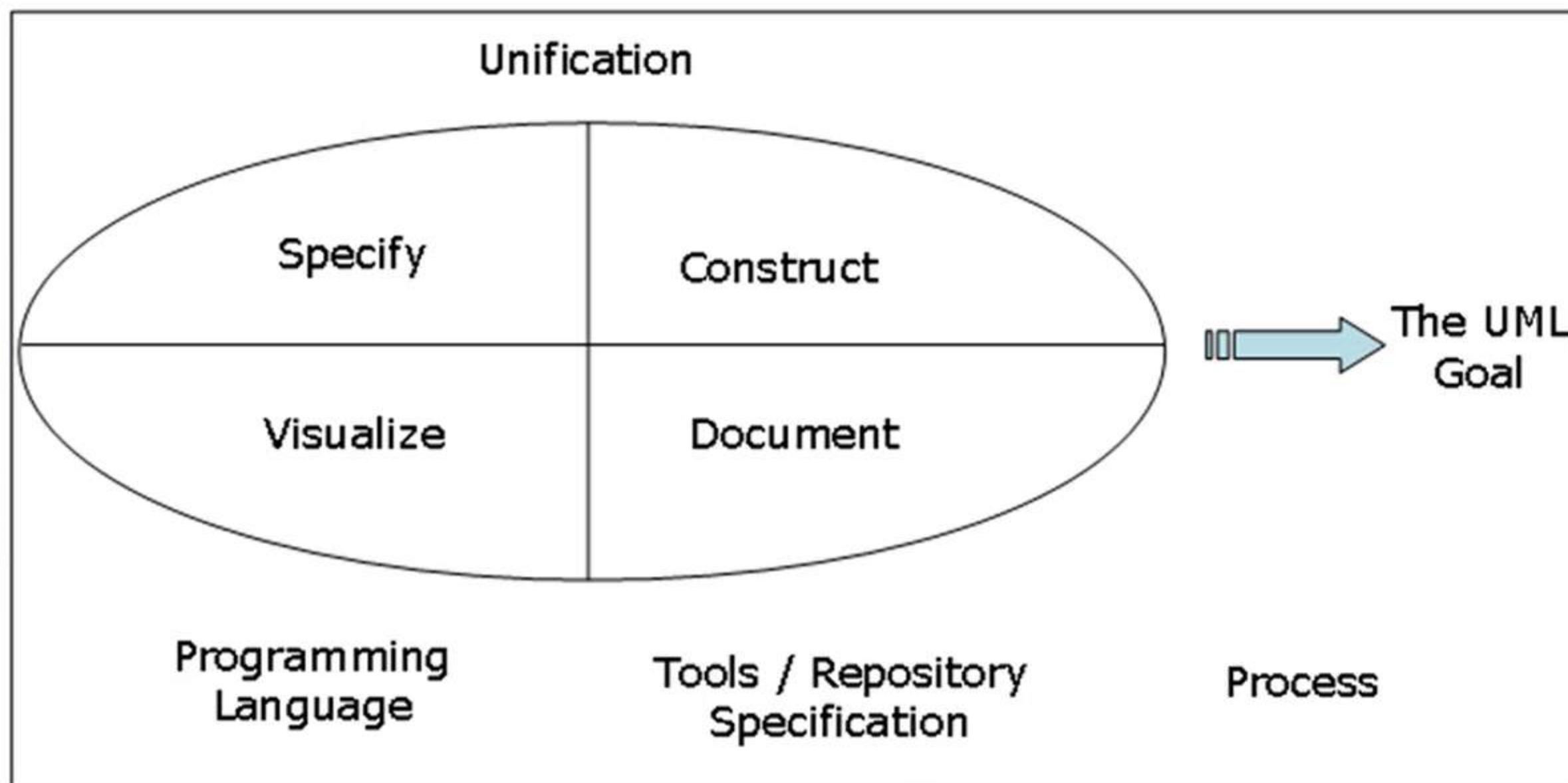
What is UML?

UML – Unified Modeling Language is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, business modeling and other non-software systems

UML is a pictorial language used in making of software blue prints.

Applies to modeling and systems

Primary goal of UML



UML Architecture

User View

- Models which define a solution to a problem as understood by the client or stakeholders.
- Use Case Diagram

Structural View

- Models which provide the static, structural dimensions and properties of the modelled system.
- Class diagram, Object diagram

Behavioural View

- Models describe the behavioural and dynamic features and methods of the modelled system.
- Interaction diagram(sequence diagram, Collaboration diagram), Activity diagram, state chart diagram

UML Architecture

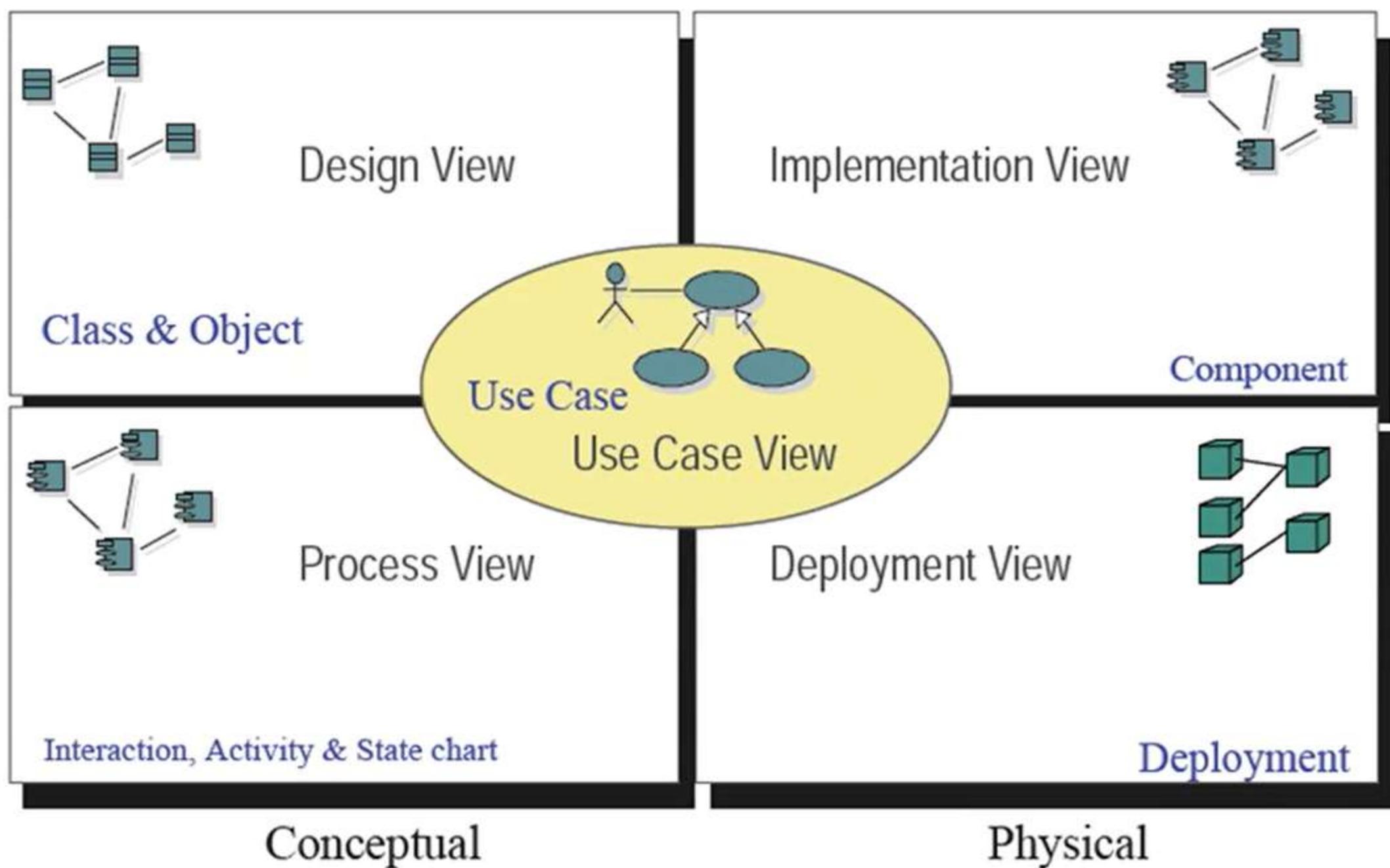
Implementation View

- View combines the structural and behavioral dimensions of the solution realization or implementation.
- Component diagram, package diagram

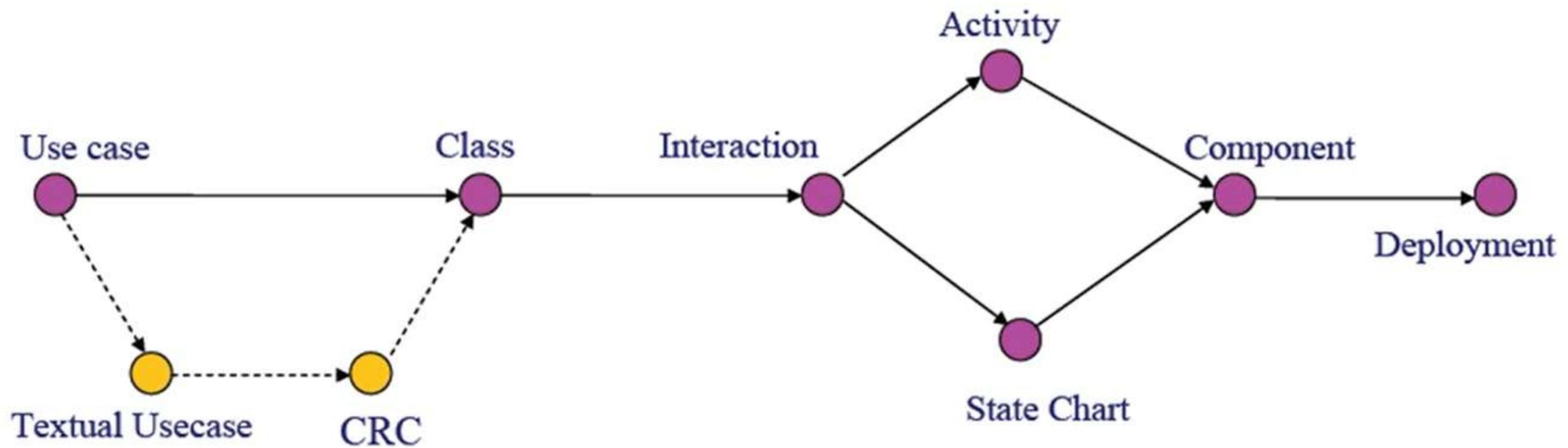
Environment view

- Models describe both structural and behavioral dimensions of the domain or environment in which the solution is implemented.
- Deployment diagram

UML



UML



- UML Diagrams
- Not Part Of UML

UML Diagrams

UML Diagram is classified as

- Use case Diagram
- Class Diagram
- Interaction Diagram
- Activity Diagram
- StateChart Diagram

Use Case Diagram

Depicts different users and the functions of the system.

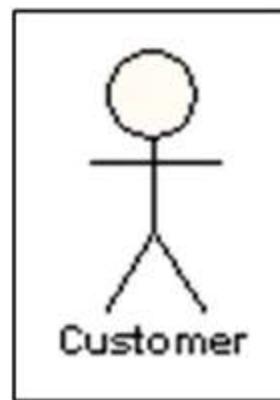
Expose the requirements of the system.

Actors interact with a use case and the functions are called as use cases.

It describes the interaction between the actor and the use case.

Use Case Diagram Components

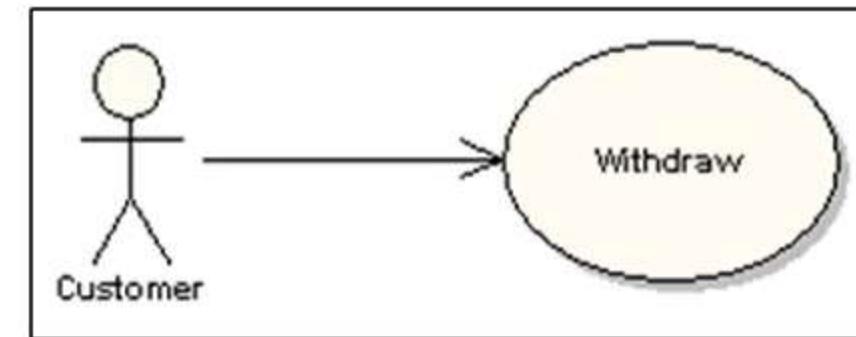
Actor : which represent users of a system, including human users and other systems.



Use case : which represent functionality or services provided by a system to users.

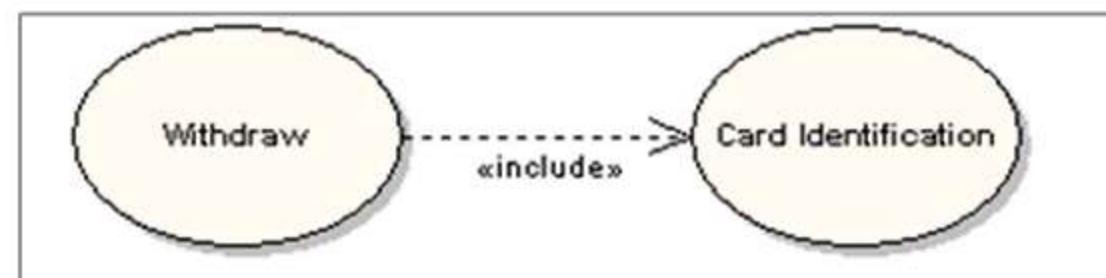


Association : Associations between actors and use cases are indicated by solid lines. An association exists whenever an actor is involved with an interaction described by a use case.

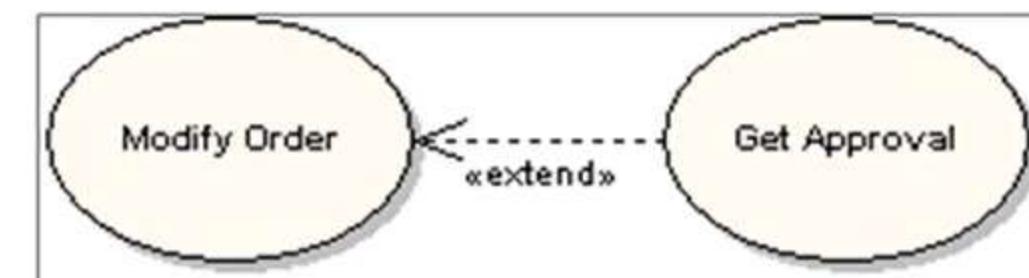


Use Case Diagram

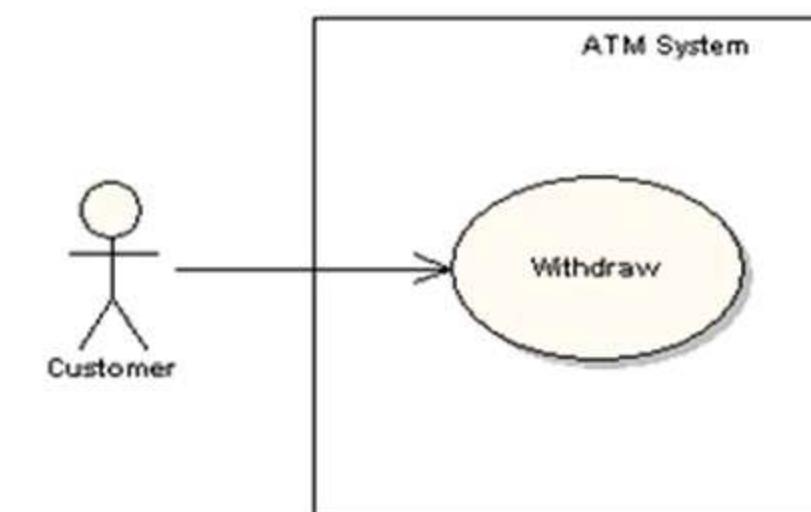
Include: Use cases may contain the functionality of another use case as part of their normal processing. In general it is assumed that any included use case will be called every time the basic path is run.



Extend : One use case may be used to extend the behavior of another; this is typically used in exceptional circumstances.



System Boundary : It is usual to display use cases as being inside the system and actors as being outside the system.



Use Case Diagram Specification

- Use case name - States the use case name
- Use case actor - Actor interacting with the use case
- Preconditions - A state of the system that must be present before a use case is performed
- Basic flow of events - Describes the ideal, primary behavior of the system
- Alternative flow of events - Describes exceptions or deviations from the basic flow
- Post conditions - A list of possible states for the system immediately after a use case is finished
- Extension points - A point in the use-case flow of events at which another use case is referenced

Class Diagram

Class Diagrams describe the static structure of a system, or how it is structured rather than how it behaves.

These diagrams contain the following elements.

- Classes - represent entities with common characteristics or features.
- Features include attributes, operations and associations.
- Associations - represent relationships that relate two or more classes

Class Diagram Notation

Class Name

attribute:Type = initialValue

operation(arg list):return type

Order

- orderId : int
- productList : List
- orderDate : String
- price : long
- delivered : boolean = false

+ placeOrder(list,price) : void
+ shipProduct(orderid) : boolean

Class Diagram

Library



1..*

Books

Aggregation

Relationship

- Relationships connect two or more classes
- Relationships among classes can be
 - Association
 - Inheritance or Generalization
 - Aggregation
 - Composition
 - Dependency
 - Realization

Bank Account

Composition

Printer Setup

Inheritance

Realization

Class Diagram

- The association is a simple relationship.
- It connects two classes, denoting the structural relationship between them.
- An association is shown as a line connecting the related classes.



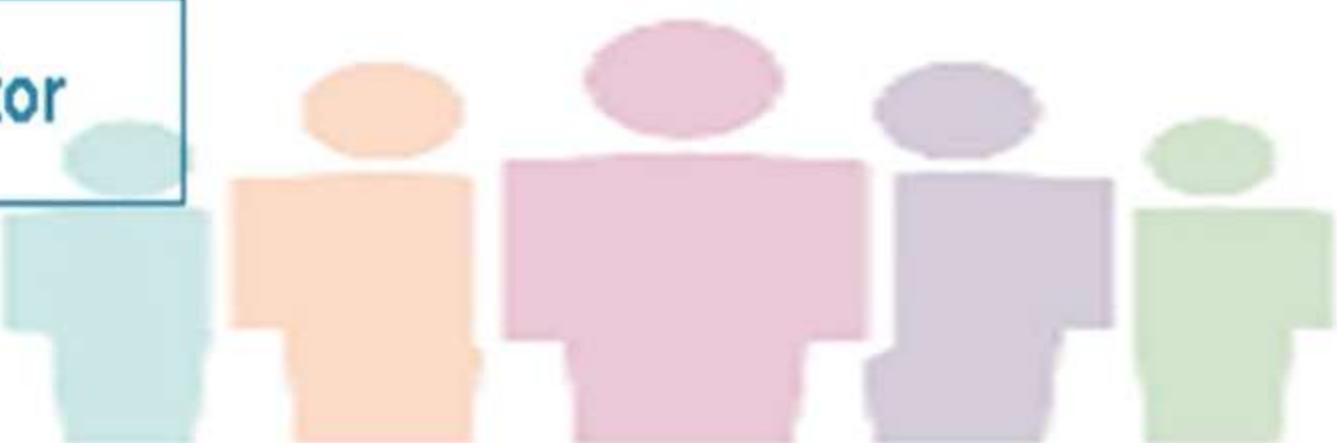
- **Multiplicity**
 - This association relationship indicates that (at least) one of the two related classes make reference to the other.
 - Indicates whether or not an association is mandatory.
 - Provides a lower and upper bound on the number of instances.



Class Diagram

Multiplicity Indicators

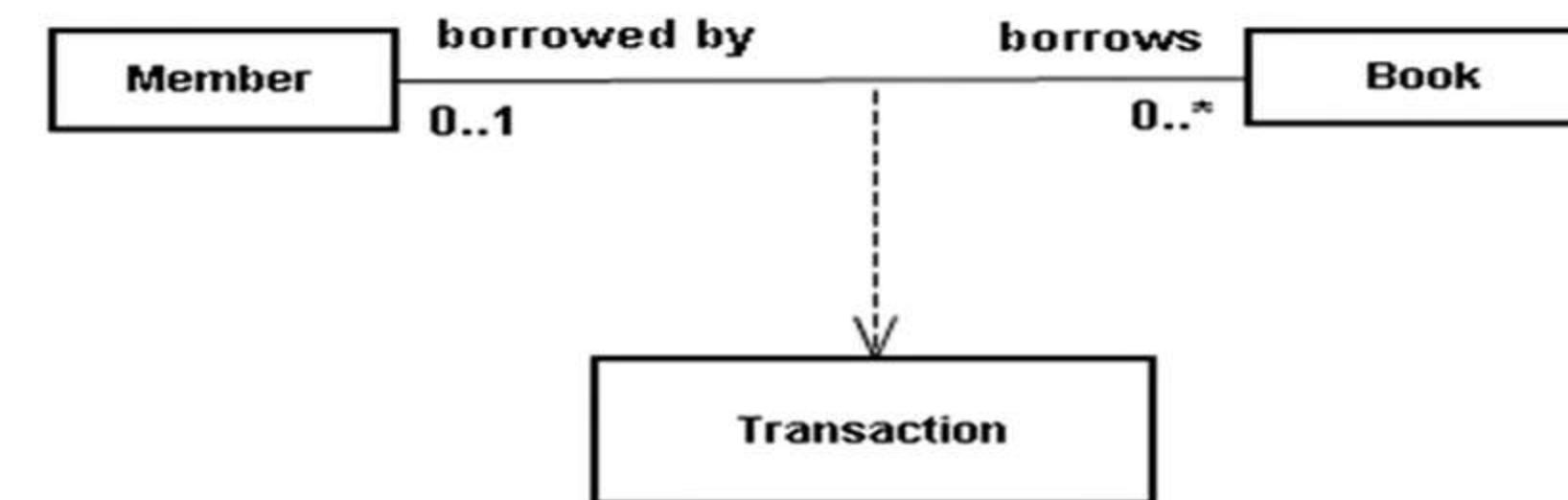
Exactly one	1
Zero or more (unlimited)	* (0..*)
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8



Class Diagram

Association class

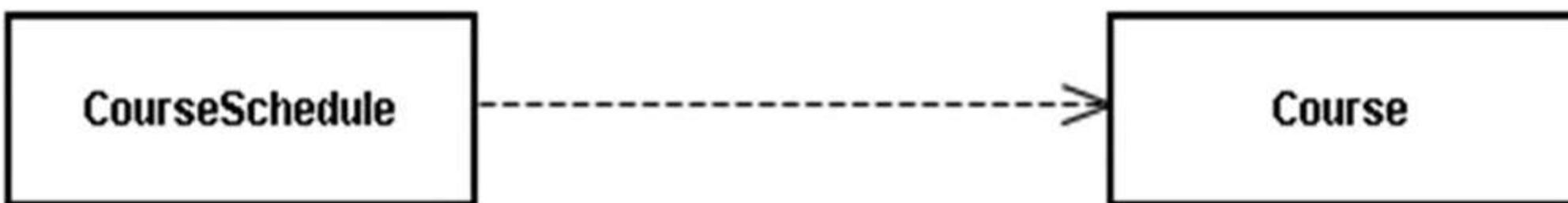
- We can also show an association class in an association relationship.
- An association class is one that makes it possible for the association to exist between two other classes.



Class Diagram

Dependency

- Is the “using” or “uses” relationship where one element is using another element to get its task done.
- A change in one element affects another element that uses it. i.e. if one changes, then it affects the other element. The reverse need not be true.
- The dependency from *CourseSchedule* to *Course* exists because *Course* is used in both the **add** and **remove** operations of *CourseSchedule*.



Class Diagram

Aggregation

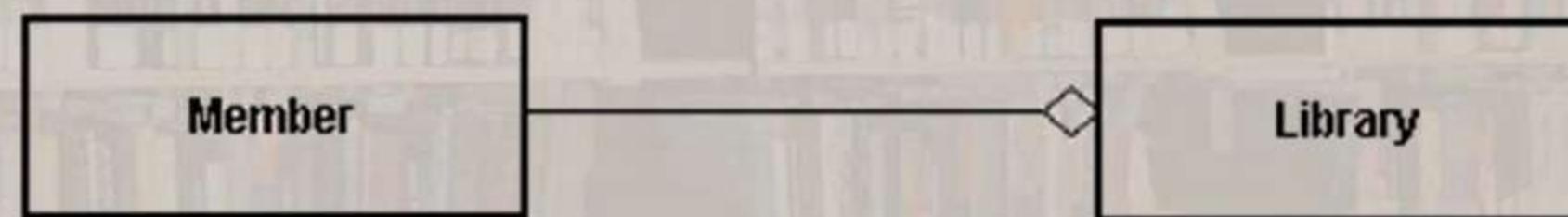
- Aggregation is a variant of the "has a" association relationship, it is more specific than association
- It is an association that represents a part-whole or part-of relationship
- Examples:
 - A room has a door.
 - Building has rooms.
 - Vehicle has an engine.
 - Company has departments
 - Book has chapters.

Two forms of Aggregation:

- Simple Aggregation
- Composition

Class Diagram

- Aggregation – “HAS-A” relationship. The part remains even after the whole is destroyed



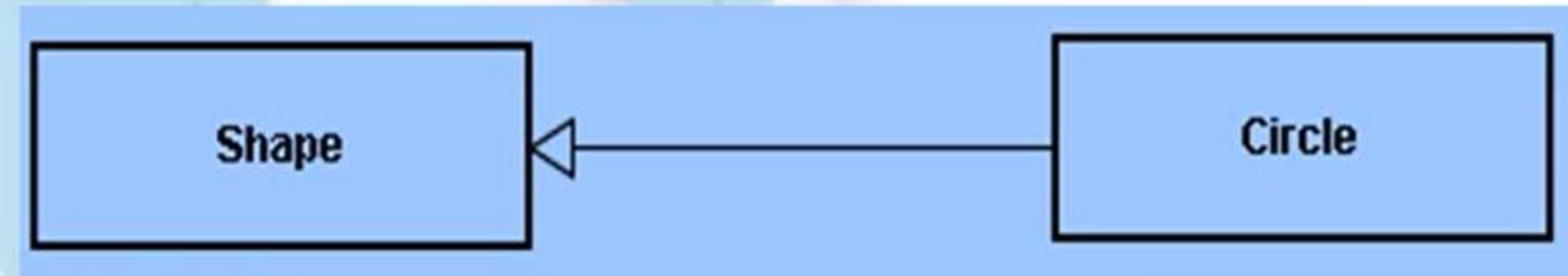
- Composition – Stronger form of aggregation. The Whole is solely responsible for the part



Class Diagram

Generalization

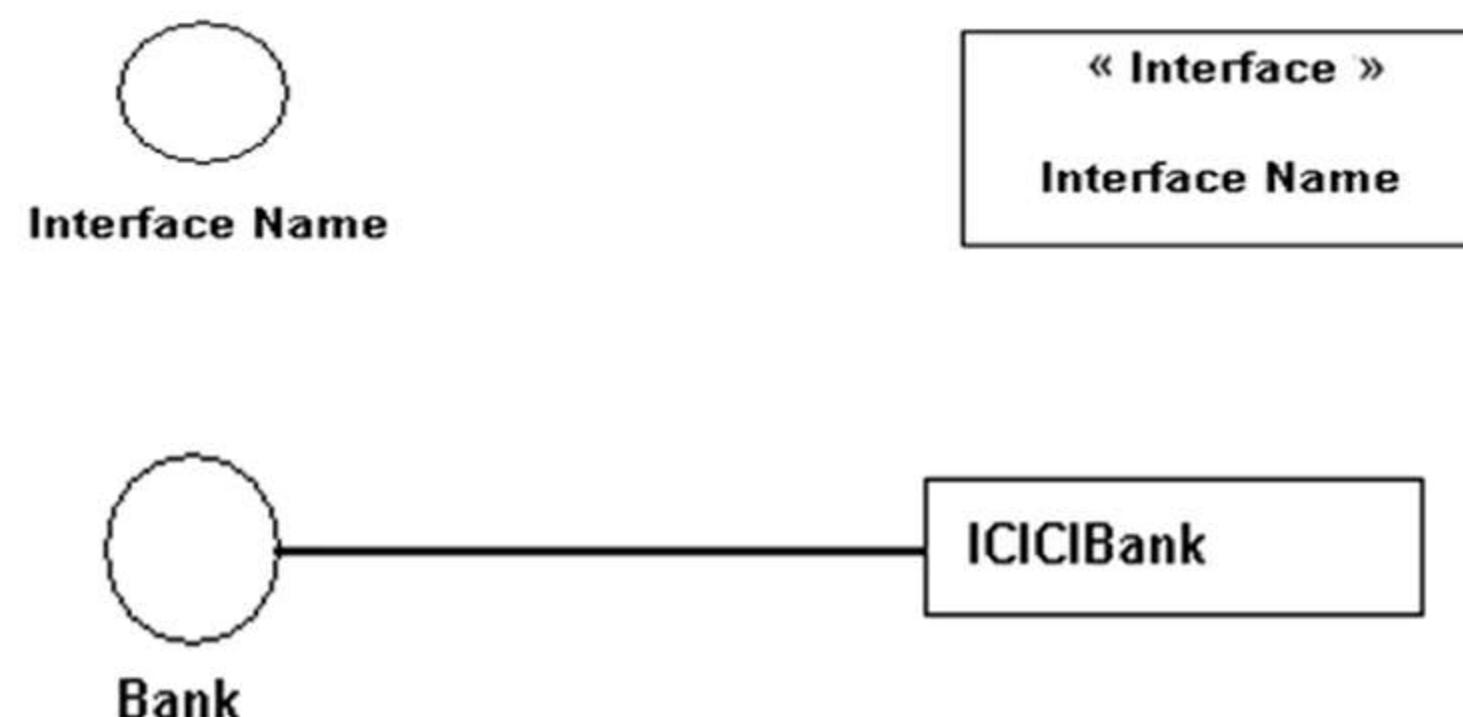
- Generalization consists of the super class and sub-class relationship.
- The super class represents a general concept while the sub class represents a more specific concept.
- Also called as “is a type of”.
- Examples:
 - Rose is a type of flower.
 - Windows is a kind of operating system.



Class Diagram

Realization

- The relationship between the class and the interface.
- An interface is a contract which can be implemented by classes which may /may not be related
- All methods inside the interface are public and abstract



Interaction Diagram

Visualize the interactive behavior of the system

Purpose of interaction diagram

- To capture dynamic behavior of a system.
- To describe the message flow in the system.
- To describe structural organization of the objects.
- To describe interaction among objects.

Interactive behaviour is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram.

Sequence Diagram

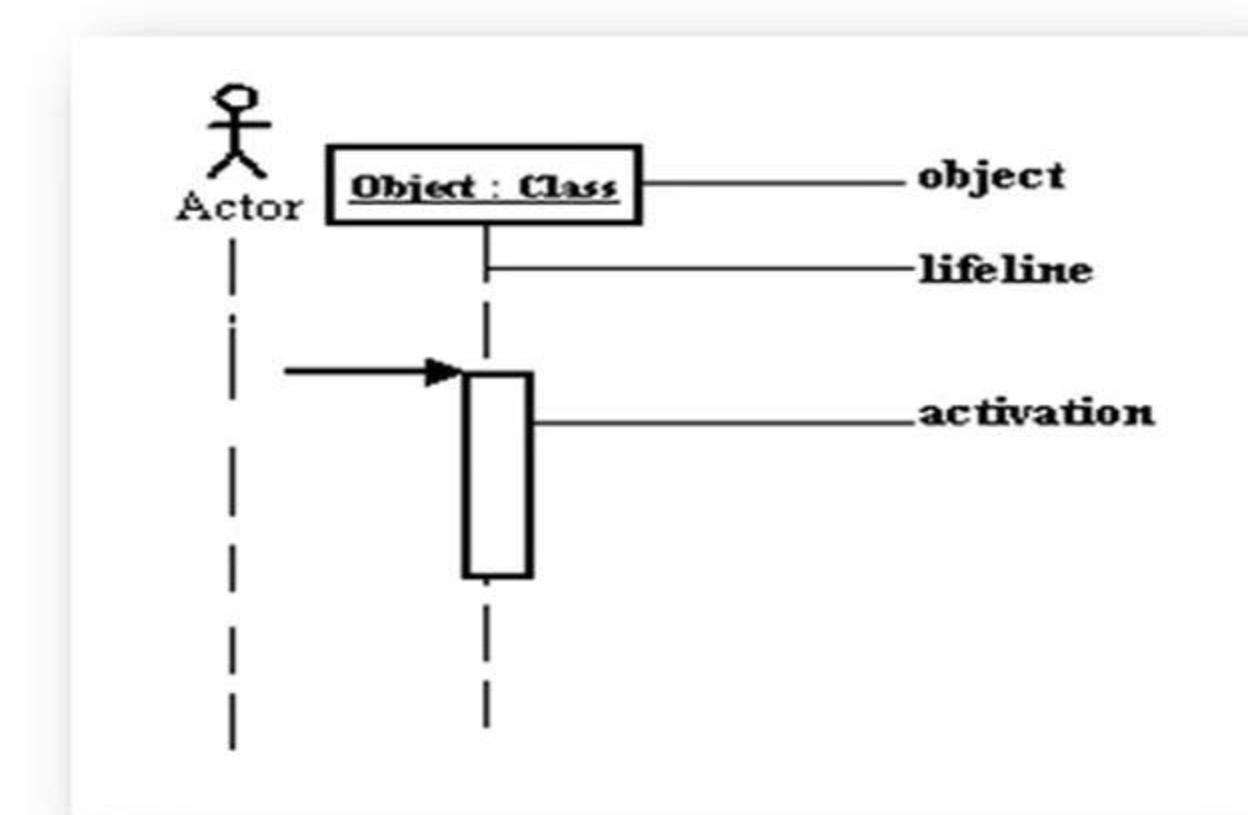
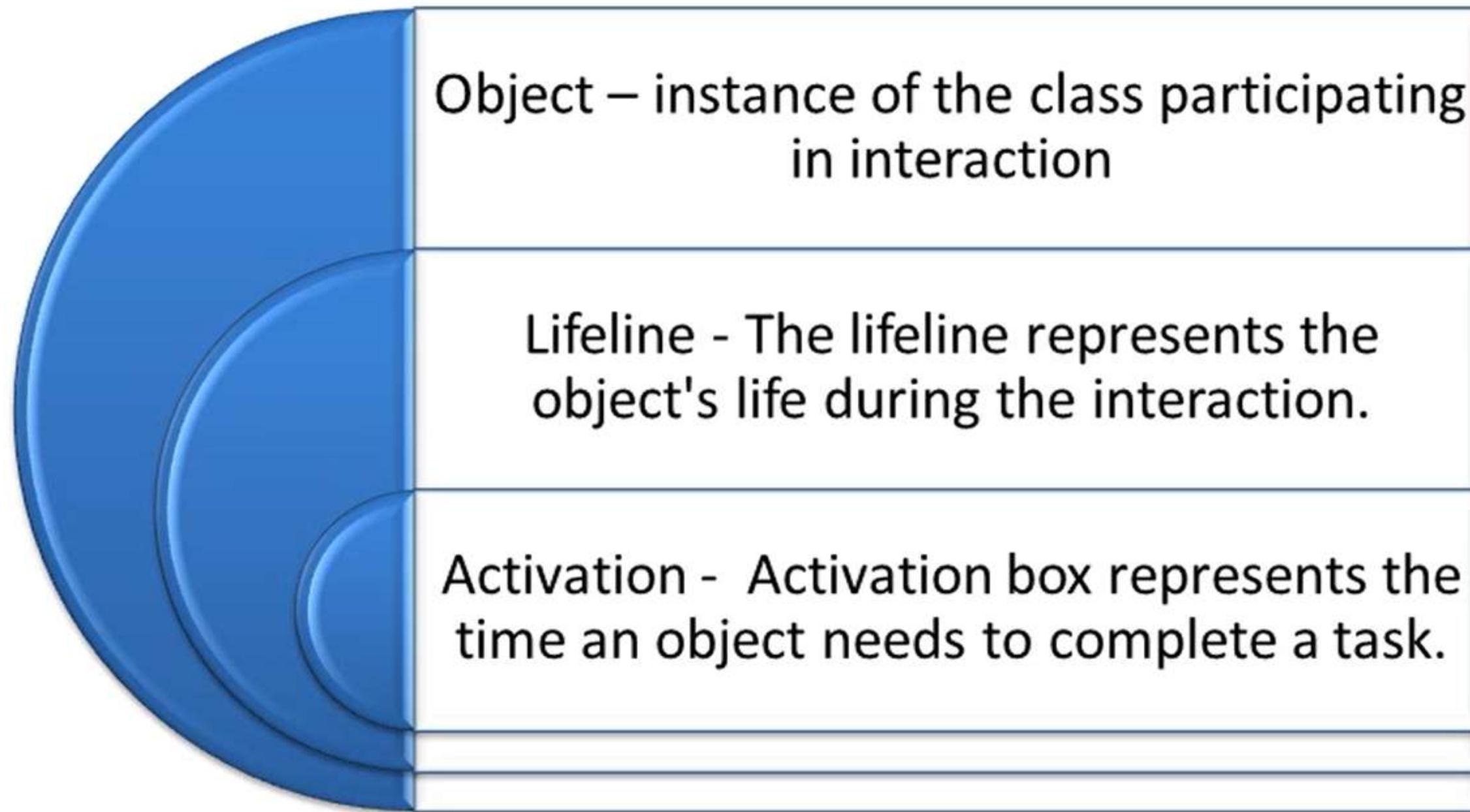
A sequence diagram shows object interactions arranged in time sequence

It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality

Graphically a sequence diagram is a table that shows objects arranged along X axis and messages, ordered in increasing time along the Y axis.

It can model simple sequential flow, branching, iteration and concurrency.

Sequence Diagram - Notation

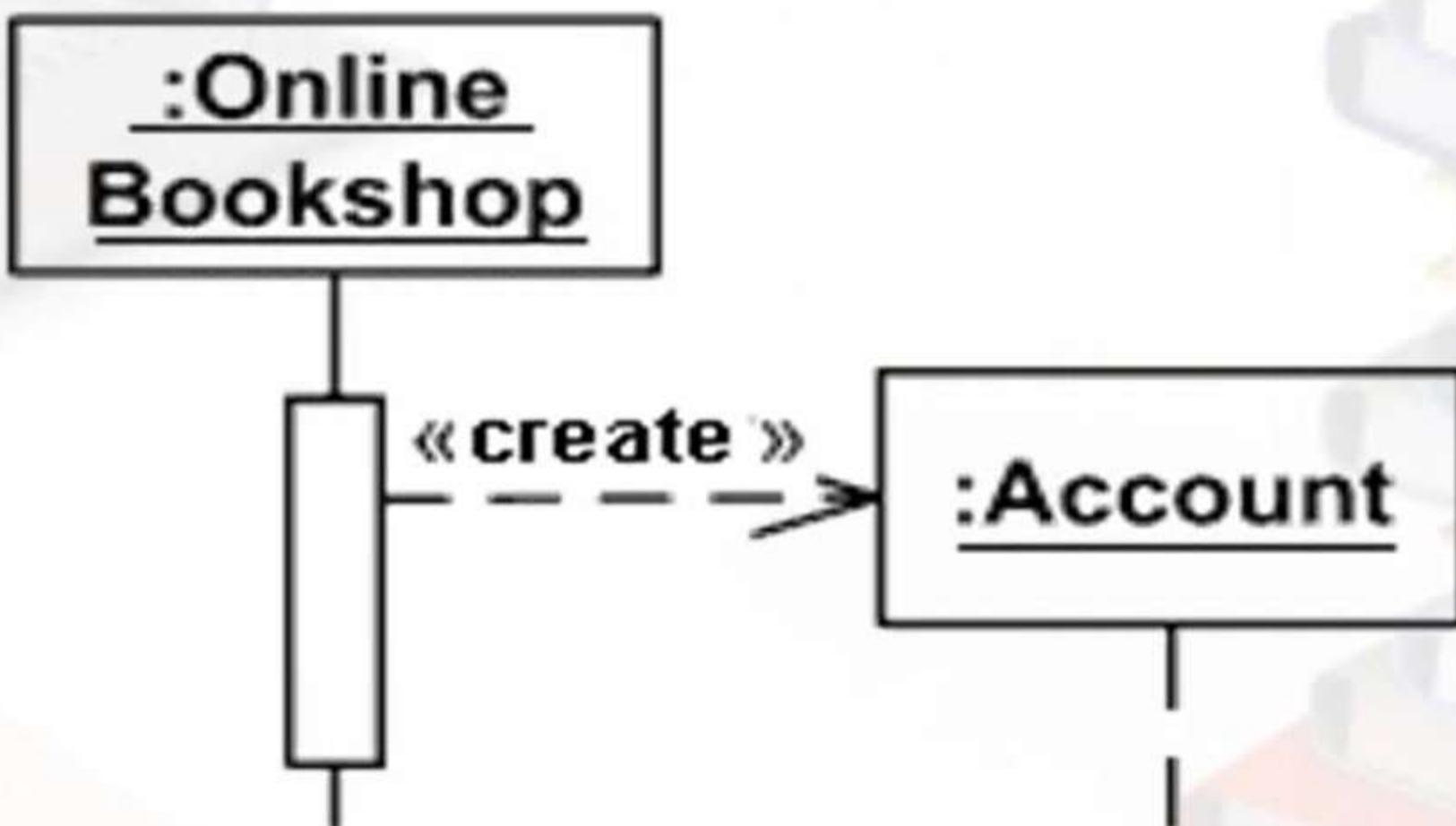


Sequence Diagram - Notation

Messages

Create

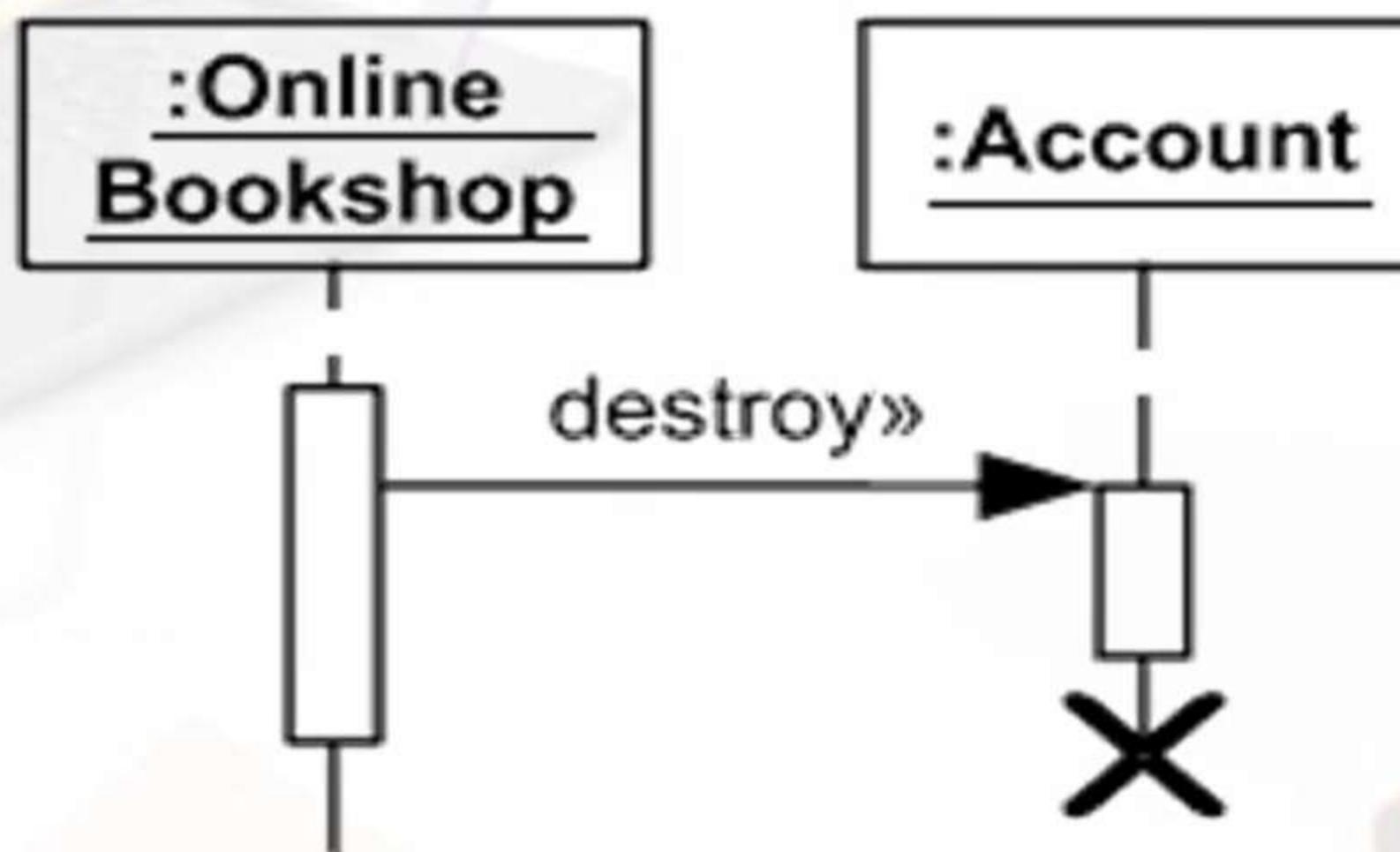
- Create message is a kind of message that represents the instantiation of (target) lifeline.



Sequence Diagram - Notation

Destroy

- Destroy message is a kind of message that represents the request of destroying the life cycle of target lifeline.



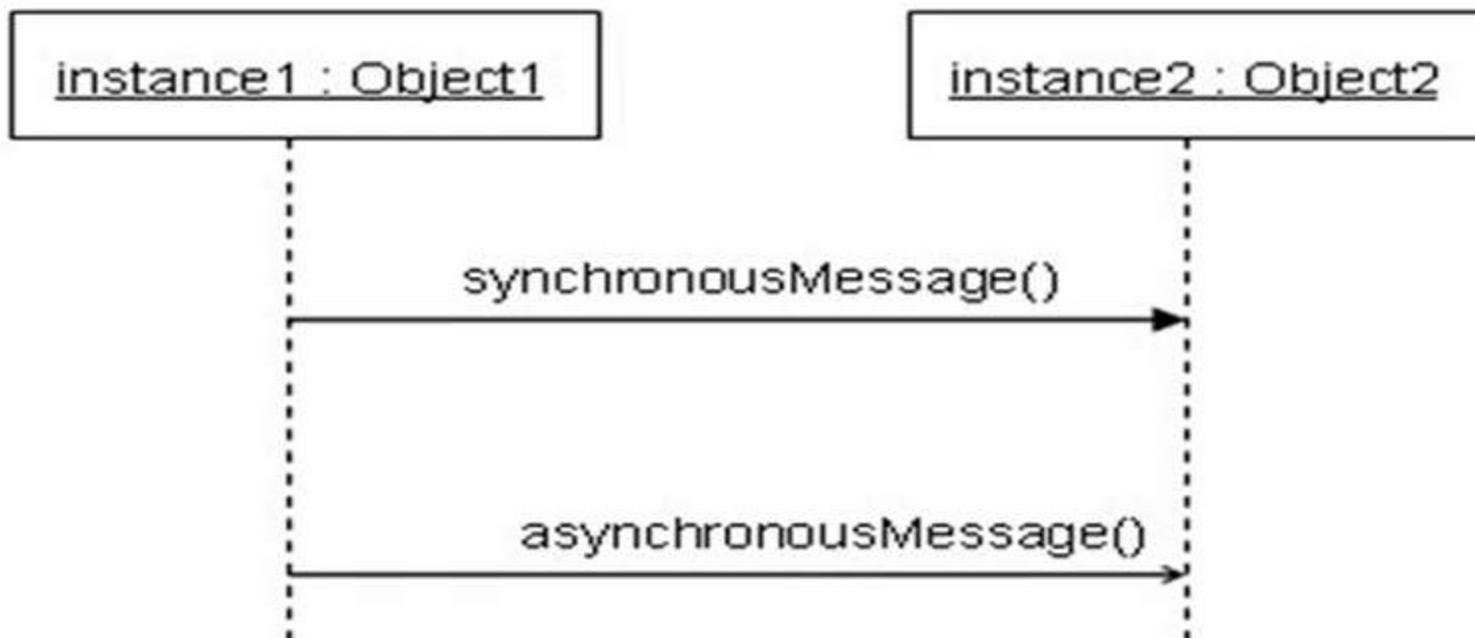
Sequence Diagram - Notation

Synchronous call

- It must wait until the response is received , such as invoking a subroutine.

Asynchronous call

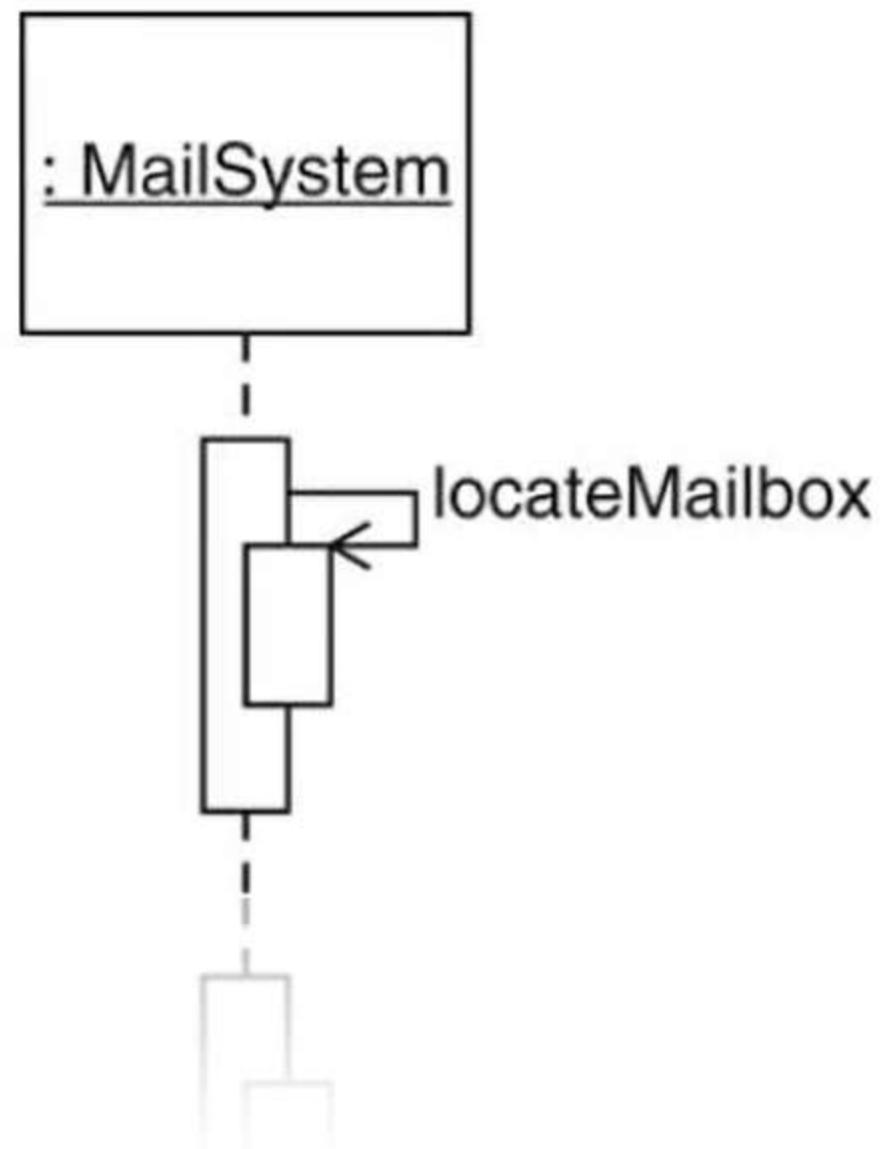
- It can continue processing and doesn't have to wait for a response.



Sequence Diagram - Notation

Self call

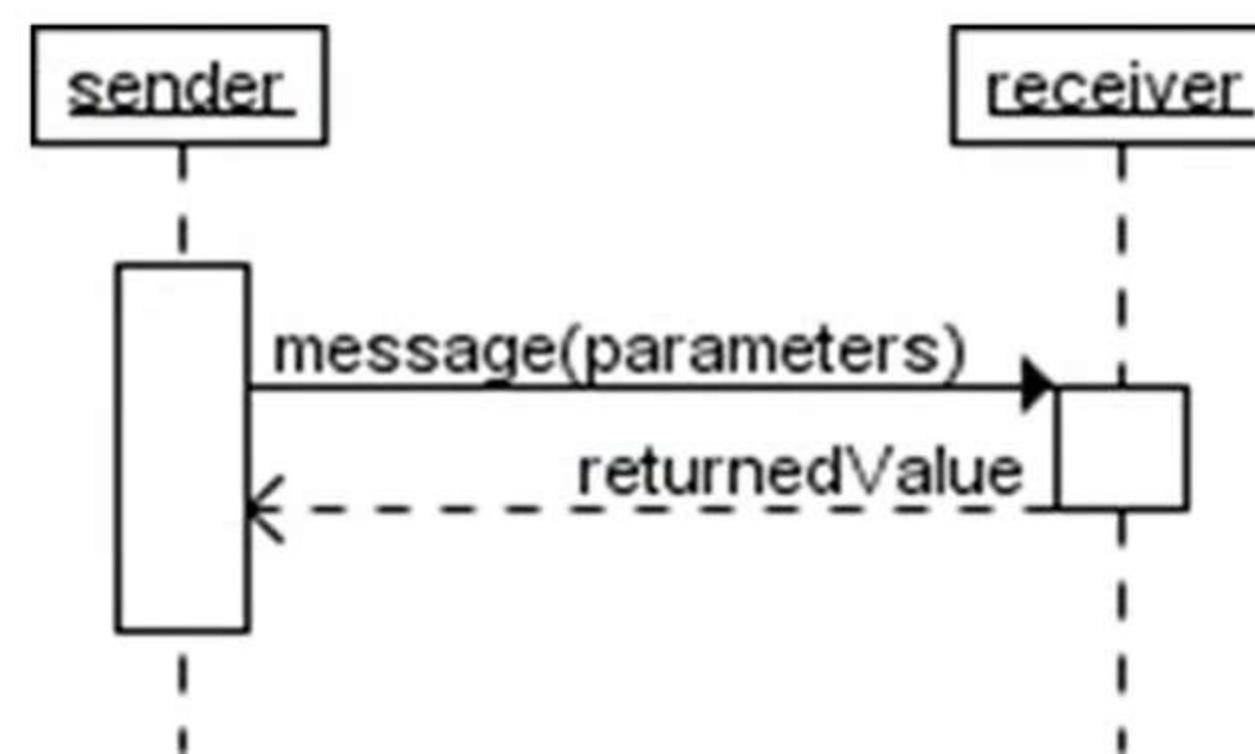
- A self message can represent a recursive call of an operation, or one method calling another method belonging to the same object.



Sequence Diagram - Notation

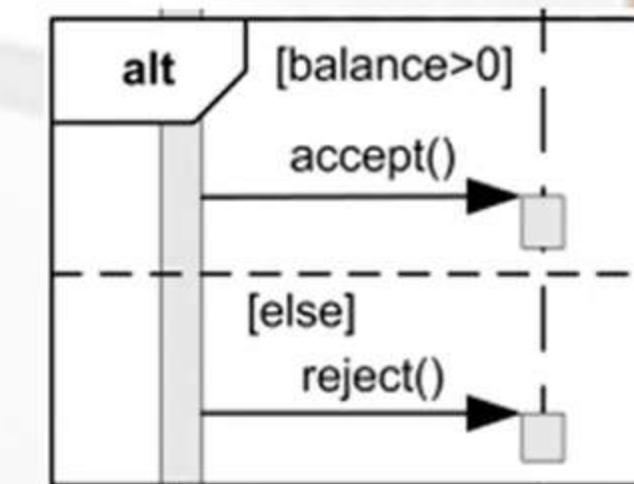
Return message

- Return message is a kind of message that represents the passing of information back to the caller to a corresponding former message.

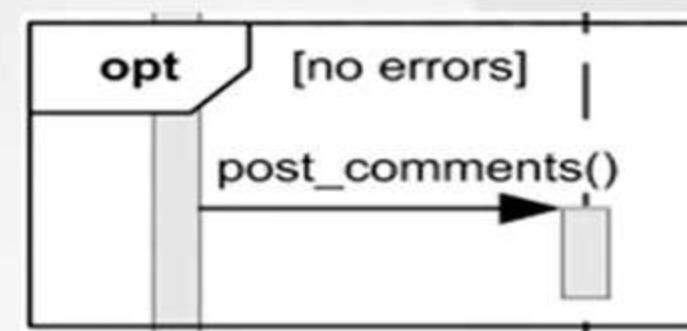


Sequence Diagram - Notation

- Alt – shows the true and the false logic flow



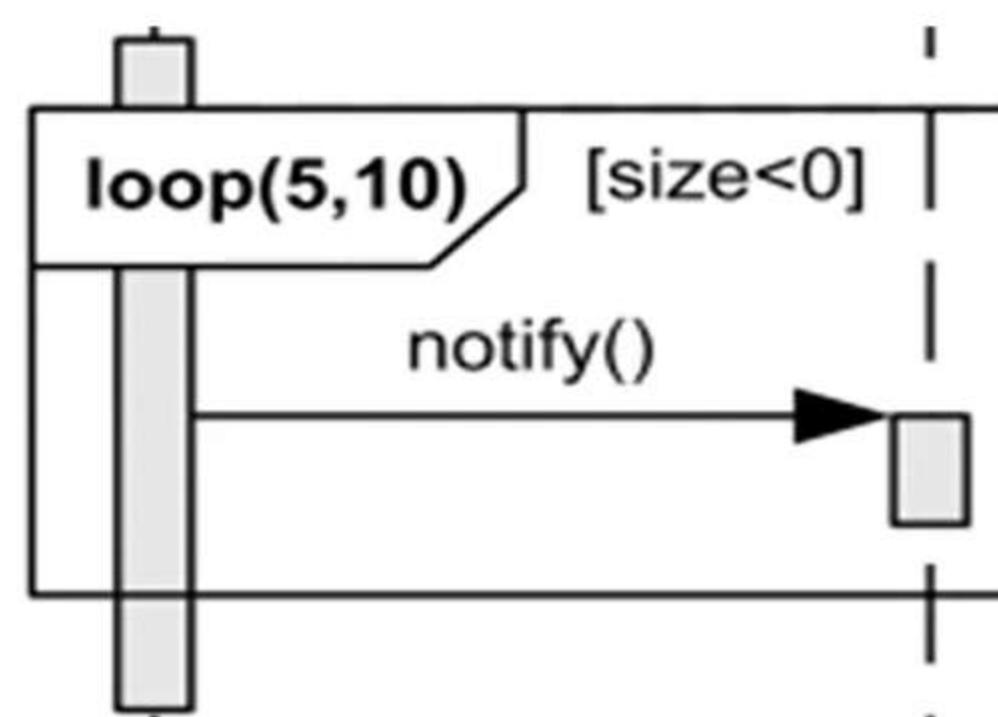
Opt - The interaction operator opt means that the combined fragment represents a choice of behavior where either the (sole) operand happens or nothing happens.



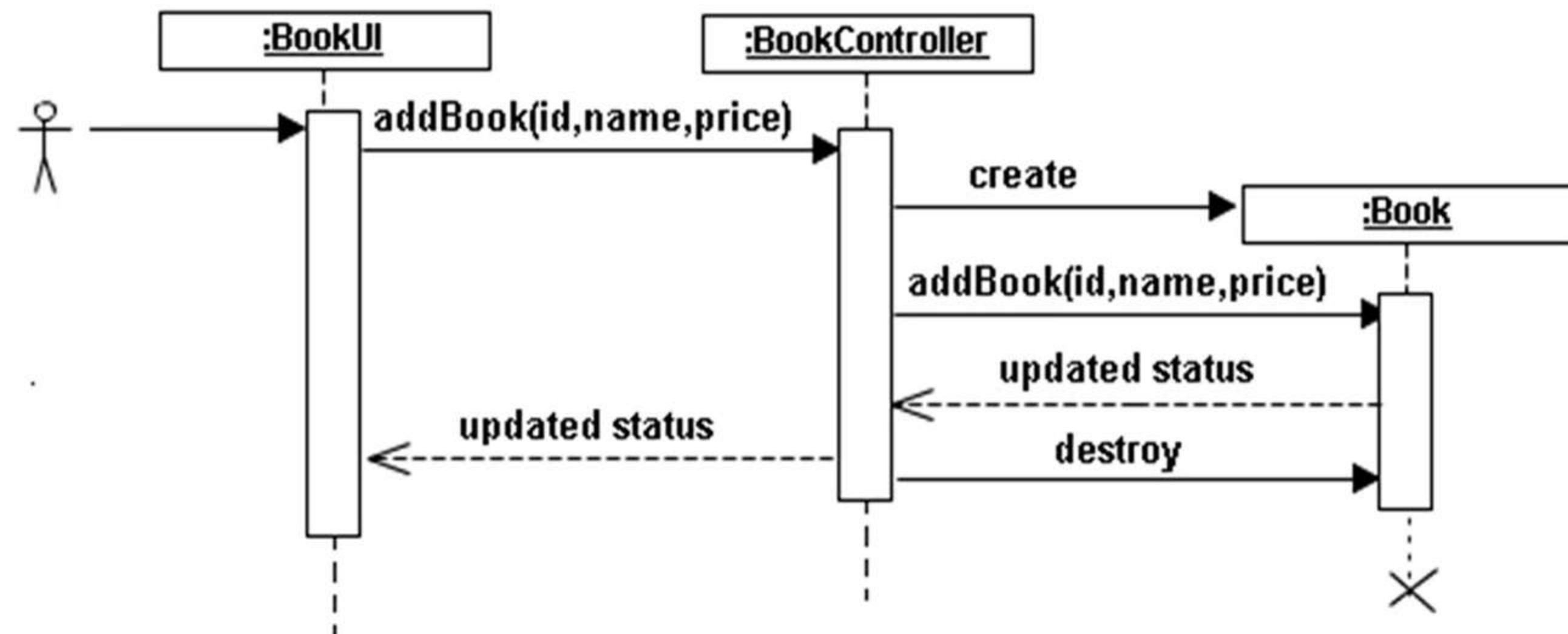
Sequence Diagram - Notation

The loop operand will be repeated a number of times.

The loop is expected to execute minimum 5 times and no more than 10 times. If guard condition [size<0] becomes false, loop terminates regardless of the minimum number of iterations specified.



Sample Sequence Diagram



Activity Diagram

An activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity.

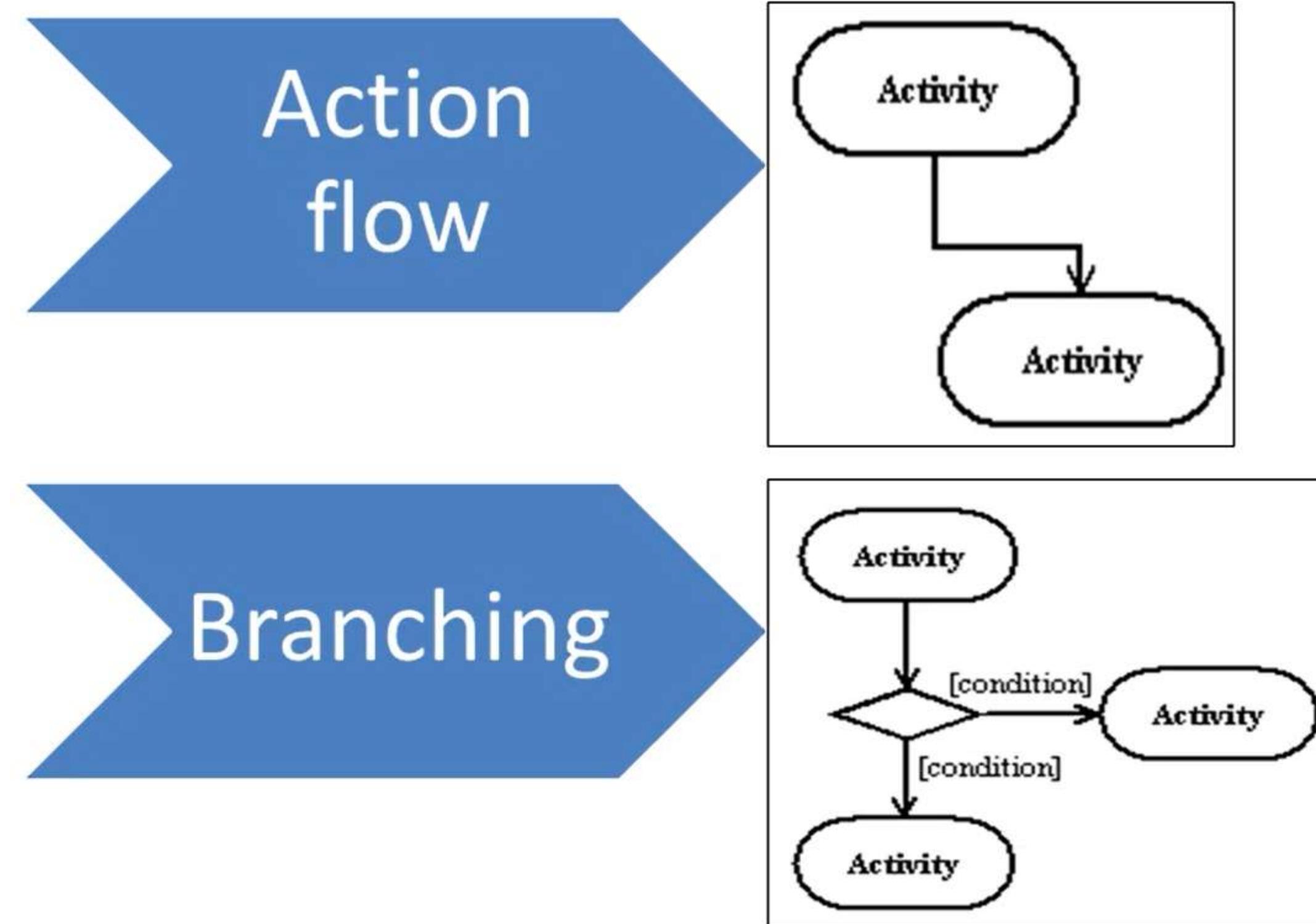
An activity represents an operation on some class in the system that results in a change in the state of the system.

Activity diagrams are used to model workflow or business processes and internal operation

Notations

- Initial state
- Final state
- Action state

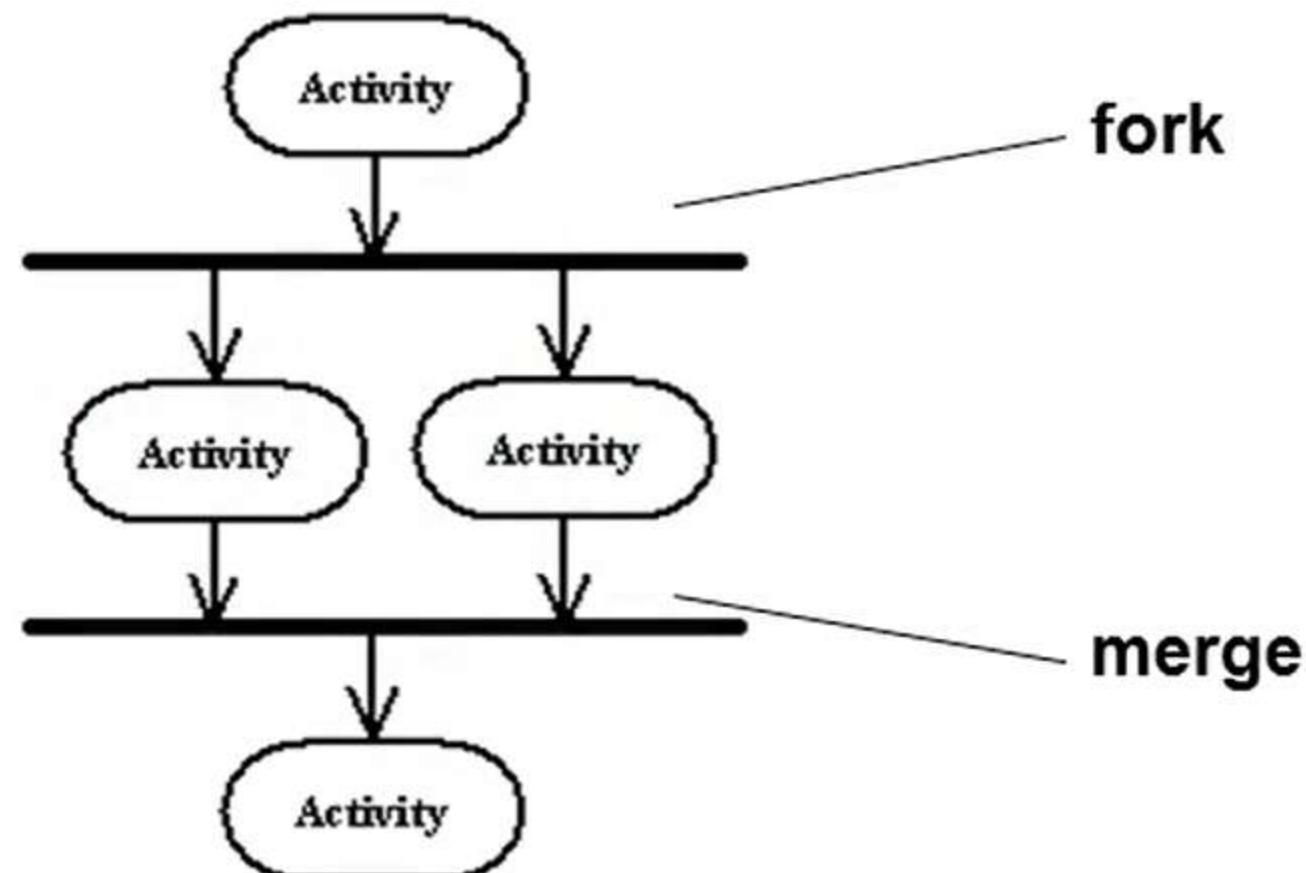
Activity Diagram



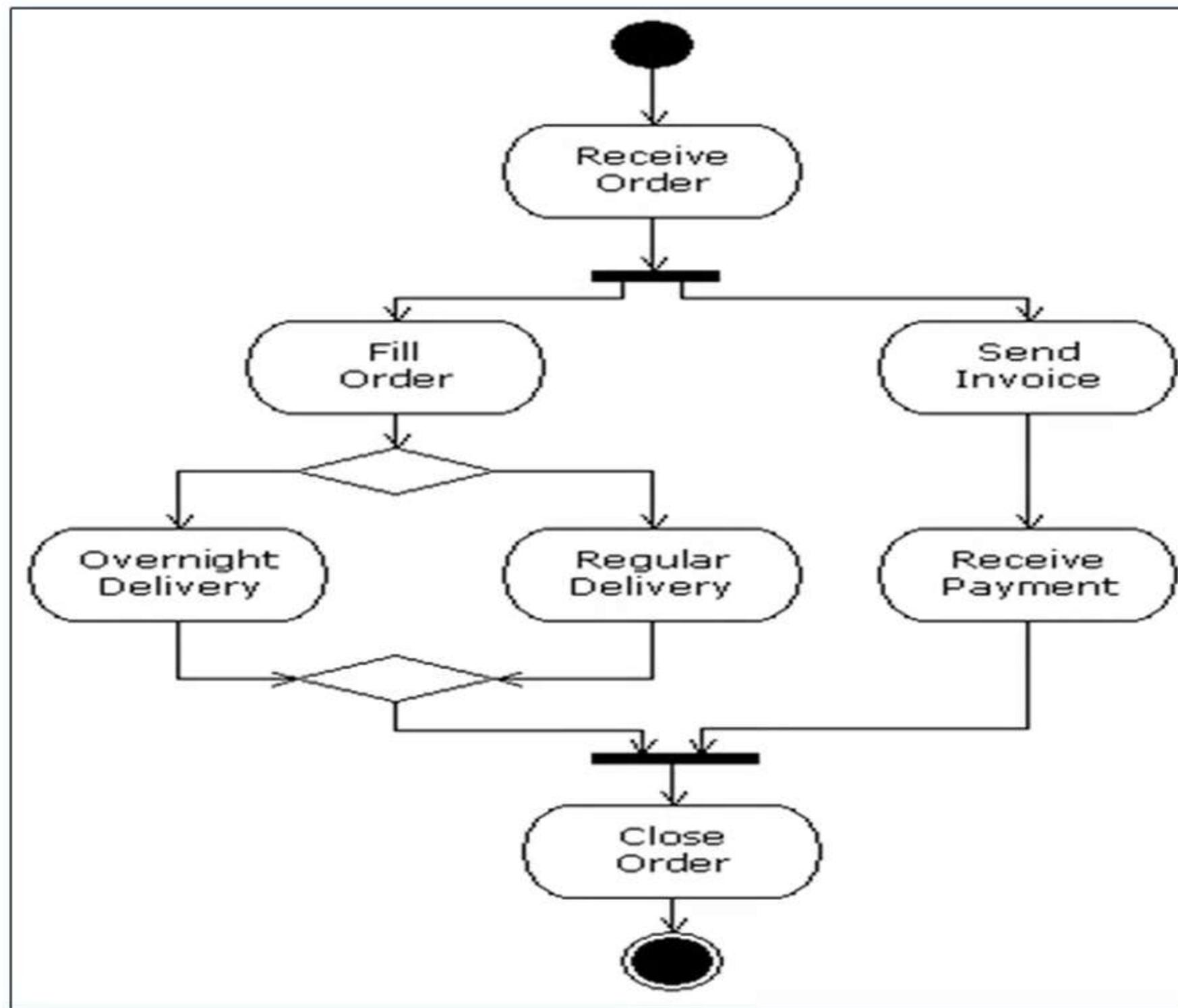
Activity Diagram

Synchronization

- A synchronization bar helps illustrate parallel transitions. Synchronization is also called forking and joining.



Activity Diagram



State Chart Diagram

Used to model dynamic nature of a system.

They define different states of an object during its lifetime.

Purpose of State chart diagram:

- To model dynamic aspect of a system.
- To model life time of a reactive system.
- To describe different states of an object during its lifetime.
- Define a state machine to model states of an object.

State Chart Diagram Notation

- **State** - State represents situations during the life of an object.
- **Transition** - A solid arrow represents the path between different states of an object

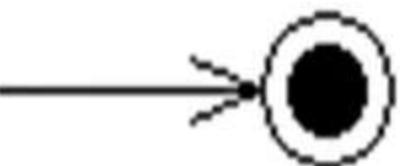


State Chart Diagram

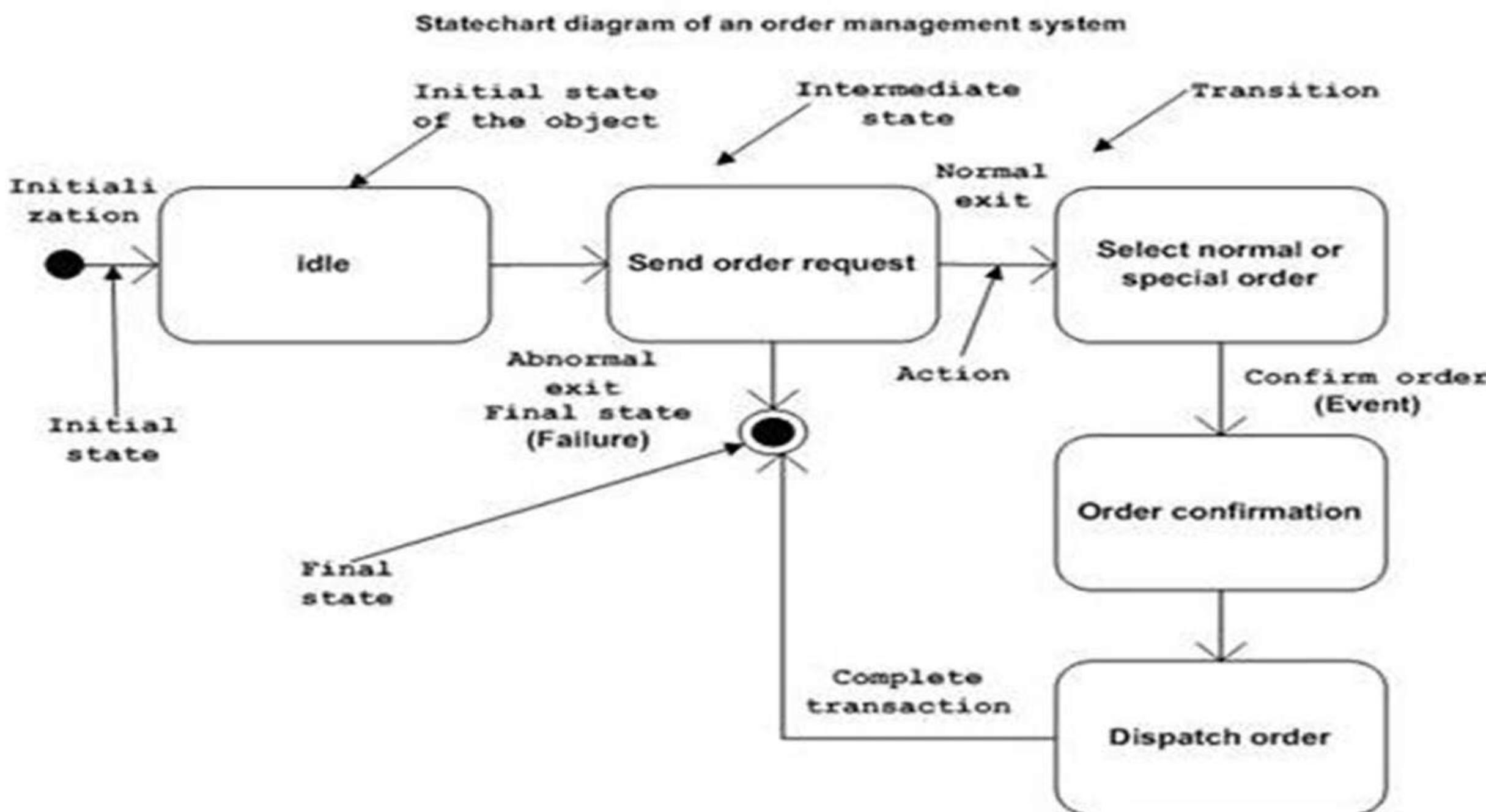
Initial State - A filled circle followed by an arrow represents the object's initial state.



Final State - An arrow pointing to a filled circle nested inside another circle represents the object's final state.



State Chart Diagram



Summary

- UML Architecture
- Use case Diagram
- Class Diagram
- Sequence Diagram
- Activity Diagram
- State chart Diagram

