UNIT-1 : SOFTWARE PRODUCT & SOFTWARE PROCESS

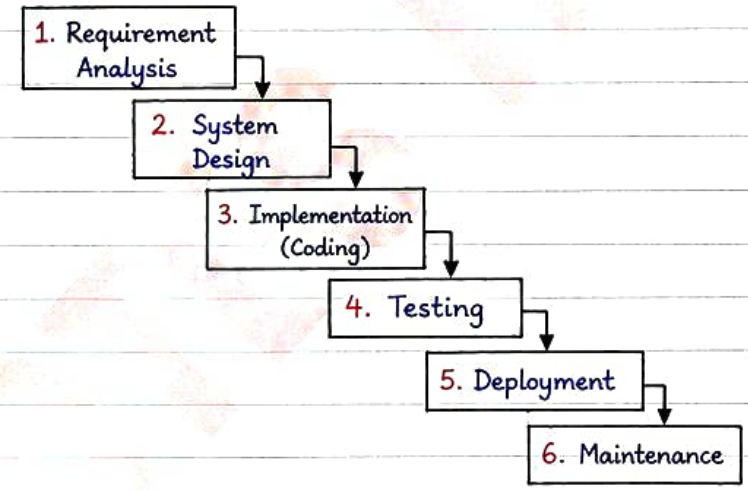
1.1 WATERFALL / LINEAR SEQUENTIAL MODEL

Waterfall Model is the earliest and simplest model of Software Development Life Cycle (SDLC). It is also called Linear Sequential Model.

Definition : Waterfall Model is a linear approach in which each phase is completed one after another in a sequential manner. The output of one phase becomes the input for the next phase. Once a phase is completed, we cannot go back to the previous phase.

Phases of Waterfall Model :

1. Requirement Analysis
2. System Design
3. Implementation (Coding)
4. Testing
5. Deployment
6. Maintenance



Explanation of Phases :

1. Requirement Analysis : In this phase, all the requirements are collected from the client and documented properly.
2. System Design : The overall system design is prepared. This includes hardware, software, database, interfaces, etc.
3. Implementation : In this phase, actual coding is done according to the design.
4. Testing : The developed software is tested to find errors and ensure that it meets the requirements.
5. Deployment : The software is delivered to the client and installed in the real environment.
6. Maintenance : After deployment, software is maintained. Errors are fixed and enhancements are done.

Advantages :	Disadvantages :
1. Simple and easy to understand.	1. No going back to previous phase.
2. Each phase has specific deliverables.	2. Not suitable for big and complex projects.
3. Easy to manage due to rigid structure.	3. Working software is seen late.
4. Works well for small projects.	4. Not flexible to changes.

When to use :

Waterfall model is used when :

- Requirements are clear and fixed.
- Technology is well understood.
- Project is small and time is short.

RGPV NOTES HUB

UNIT-3 : SOFTWARE DESIGN

8. SA/SD (STRUCTURED ANALYSIS / STRUCTURED DESIGN)

1. Introduction

Structured Analysis (SA) and Structured Design (SD) are top-down approaches to develop a system by decomposing the system into smaller modules and functions.
They use a set of techniques and tools to represent and design the system in a systematic and disciplined way.

Key Points

- ★ SA focuses on WHAT the system should do.
- ★ SD focuses on HOW the system will do.
- ★ Easy to understand and implement.
- ★ Based on modularization and function orientation.
- ★ Uses DFD, DD, ERD, Structure Chart, etc.

2. Need for SA/SD

- To handle system complexity.
- To improve system understandability.
- To facilitate communication.
- To provide a clear structure for coding.
- To ease testing and maintenance.

3. Structured Analysis (SA)

SA is the process of collecting requirements and understanding the problem domain. It answers the question "WHAT is to be done?".

SA Activities / Steps

1. Study the current system.
2. Elicit user requirements.
3. Model the system using appropriate tools.
4. Validate the model with users.
5. Prepare the SRS (Software Requirement Specification).

4. Tools Used in Structured Analysis

Tool	Purpose
DFD (Data Flow Diagram)	Shows how data flows in the system.
DD (Data Dictionary)	Describes all data items.
ERD (Entity Relationship Diagram)	Shows data entities and their relationships.
State Transition Diagram	Shows different states of the system.
Decision Table / Tree	Represents business rules and logic.
Process Specification (PSPEC)	Describes the logic of a process.

5. Data Flow Diagram (DFD)

DFD shows the flow of data in the system.

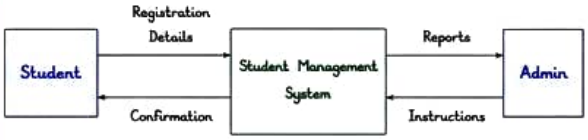
Components of DFD

- Process : Transforms input data to output data.
- Data Flow : Movement of data.
- Data Store : Stores data.
- Source/Sink : External entities.

DFD Levels

- Context Diagram (Level 0)
  - Shows the whole system as a single process.
- Level 1 DFD
  - Decomposes the system into major processes.
- Level 2, 3 ... DFD
  - Further decomposition of processes.

DFD Example - Level 0 (Context Diagram)

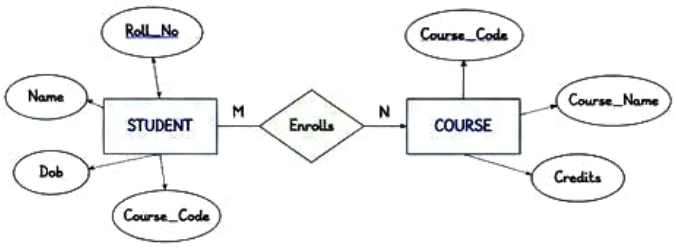


6. Data Dictionary (DD)

DD contains detailed information about all data items used in the system.

Data Item	Description	Type	Length	Alias	Example
Roll_No	Roll number of student	Numeric	6	Roll	230101
Name	Name of student	Alphanumeric	30	Std_Name	Rohan
Dob	Date of birth	Date	-	Birth_Date	12/05/2004
Course_Code	Course code	Alphanumeric	6	C_Code	CS101
Marks	Marks obtained	Numeric	3	Score	85

7. Entity Relationship Diagram (ERD)



8. Structured Design (SD)

SD is the process of converting the requirements (from SA phase) into a blueprint for implementation. It answers the question "HOW it will be done?".

SD Activities / Steps

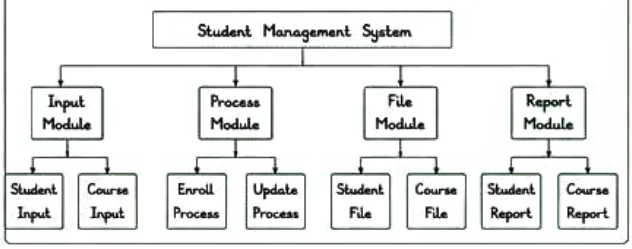
1. Transform SA model into design model.
2. Design the architecture and modules.
3. Design data structures.
4. Design interface and database.
5. Prepare structure charts and detailed design.

9. Tools Used in Structured Design

Tool	Purpose
Structure Chart	Shows the module structure and calling relationships.
Module Specification	Describes the logic of each module.
Data Structure Description	Defines the data structures used.
Interface Design	Designs user interface and I/O formats.
Pseudo Code	Describes logic in a programming like language.

10. Structure Chart

Structure chart shows the hierarchy of modules in a system.



11. SA vs SD

Basis	Structured Analysis (SA)	Structured Design (SD)
Focus	What the system should do.	How the system will do.
Phase	Earlier phase of development.	Later phase of development.
Input	Requirements from users.	SA models and SRS.
Output	DFD, DD, ERD, etc.	Structure charts, module specs, etc.
Goal	Understand the problem.	Design the system solution.
Concern	Data and processes.	Modules and data structures.

12. Advantages

- Easy to understand and communicate.
- Improves modularity.
- Supports top-down approach.
- Helps in early detection of errors.
- Easier maintenance and enhancement.

In Short

SA identifies WHAT the system needs.
SD decides HOW it will be built.
Together, SA/SD provide a clear roadmap from requirements to implementation.

13. Limitations

- Not suitable for very large and complex systems.
- Time consuming in analysis and design.
- Requires skilled analysts and designers.

10. DESIGN METRICS

1. Introduction

Design Metrics are quantitative measures used to evaluate the quality of a software design. They help in comparing alternative designs, predicting attributes of the software product and guiding the design process.

Key Points

- ★ Quantitative measurement of design attributes.
- ★ Help in objective evaluation and comparison.
- ★ Predict maintainability, complexity, reliability, etc.
- ★ Used in design improvement and quality assurance.

2. Need for Design Metrics

- To evaluate design quality objectively.
- To compare alternative design solutions.
- To predict product attributes early.
- To identify design weaknesses.
- To guide the designer for improvements.
- To support estimation of effort, cost and risk.

3. Characteristics of Good Metrics

- ☑ Simple and deasy to understand.
- ☑ Quantifiable and objective.
- ☑ Consistent and repeatable.
- ☑ Sensitive to design changes.
- ☑ Correlated with quality attributes.
- ☑ Easy to compute with available data.

4. Categories of Design Metrics

Category	Focus	Examples
Structural Metrics	Measure the structural properties of the design.	Size, Complexity, Coupling, Cohesion
Data Metrics	Measure data related to the design.	Data Abstraction, Data Structure Complexity
Interface Metrics	Measure properties of the interfaces.	Interface Size, Parameter Count
Component Metrics	Measure properties of components or modules.	Component Size, Fan-in, Fan-out
System Metrics	Measure overall system level attributes.	System Complexity, Reusability

5. Common Design Metrics

Metric	Formula / Definition	Measures	Better if
Size	LOC (Lines of Code) or Function Points	Amount of software	As per requirement (optimal)
Cyclomatic Complexity (V(G))	V(G) = E − N + 2P (E = edges, N = nodes, P = number of connected components)	Control flow complexity	Low
Coupling (C)	Degree of interdependence between modules	Inter-module dependence	Low
Cohesion (H)	Degree to which elements inside a module belong together	Intra-module relatedness	High
Fan-in	Number of modules that call a given module	Reusability of a module	High
Fan-out	Number of modules called by a given module	Dependency of a module	Low
Interface Size (I)	Number of parameters + control flags in an interface	Complexity of interface	Low
Depth of Inheritance Tree (DIT)	Longest path from a class to the root class	Inheritance complexity	Low
Data Abstraction (DA)	Ratio of data items hidden from other modules	Information hiding	High

6. Relation of Metrics with Quality Attributes

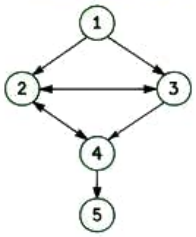
Quality Attribute	Related Metrics	Effect
Maintainability	High Cohesion, Low Coupling, Low Complexity	Improves maintainability
Reusability	High Fan-in, High Cohesion, Low Coupling	Promotes reusability
Reliability	Low Complexity, Low Coupling, High Cohesion	Increases reliability
Testability	Low Complexity, Low Coupling, Small Interfaces	Easier to test
Flexibility	Data Abstraction, Modularity, Low Coupling	Easier to modify and extend
Understandability	High Cohesion, Low Complexity, Good Modularity	Easier to understand

7. Metric Evaluation Scale (General Guideline)

Scale	Value	Interpretation
Good	0 – 3	Acceptable / Good
Average	4 – 7	May need improvement
Poor	8 – 10 (or more)	Needs significant improvement

8. Example – Metric Calculation

Control Flow Graph



In the given graph:

Nodes (N) = 5

Edges (E) = 7

Number of connected components (P) = 1

Cyclomatic Complexity,

V(G) = E − N + 2P

= 7 − 5 + 2(1)

= 4

So, Cyclomatic Complexity V(G) = 4

9. Guidelines for Using Design Metrics

- ➤ Use a set of metrics, not a single metric.
- ➤ Collect metrics early and regularly.
- ➤ Use metrics as indicators, not absolute values.
- ➤ Compare metrics within the same project.
- ➤ Combine quantitative data with expert judgment.
- ➤ Use metrics for improvement, not for punishment.

10. Benefits of Using Design Metrics

- ➤ Provide objective basis for evaluating design.
- ➤ Help in early detection of design problems.
- ➤ Improve quality and productivity.
- ➤ Help in estimating cost, effort and schedule.
- ➤ Support continuous improvement.

11. Limitations of Design Metrics

- ➤ Not all quality attributes can be measured.
- ➤ Metrics may not capture human factors.
- ➤ Overemphasis on metrics can be harmful.
- ➤ Values depend on the context and environment.
- ➤ Not a substitute for good design principles.

In Short

Design Metrics help in measuring and evaluating the quality of a design using quantitative values. They support better decision making and lead to high quality software.

Good Design + Appropriate Metrics + Continuous Improvement = Quality Software

RGPV NOTES HUB
Smart Notes, Better Scores

UNIT - 4 : SOFTWARE TESTING

1. TESTING LEVELS

RGPV (CS/IT)
UNIT - 4, TOPIC - 1
DATE : __/__/__

1. Introduction

Software testing is performed at different levels of abstraction to uncover errors as early as possible and to ensure that the complete system meets the specified requirements.

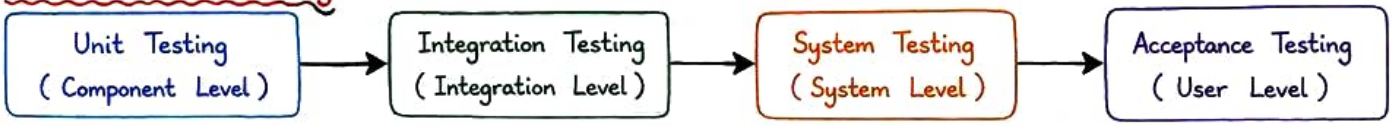
Testing levels help in managing the testing process in an organized manner.

The main testing levels are :
Unit Testing, Integration Testing, System Testing and Acceptance Testing.

2. Key Points

- ★ Testing is carried out at different levels of the software.
- ★ Each level has specific objectives and entry / exit criteria.
- ★ Defects found in early levels are cheaper and easier to fix.
- ★ Testing levels move from low level (components) to high level (complete system).
- ★ All levels together ensure quality software.

3. Levels of Testing - Overview



Individual units / modules are tested.

Combination of modules are tested.

Complete system is tested.

System is validated by user / client.

4. Details of Each Testing Level

Level	Also Known As	Objective	Focus	Performed By	When Performed	Example
1. Unit Testing	Component Testing	To verify each unit/module of the code works correctly as per its specification.	Internal logic, functions, methods, algorithms, local data structures.	Developers	After coding of individual unit.	Testing a function that calculates factorial.
2. Integration Testing	Module Integration Testing	To verify the interfaces and interaction between integrated modules.	Data flow, control flow, interfaces, database connections.	Testers	After unit testing is completed.	Testing interaction between Login module and Database module.
3. System Testing	End-to-End Testing	To verify the complete and integrated system against the requirements.	Functional and non-functional requirements, usability, performance, security, etc.	Testers	After integration testing.	Testing the entire online banking system.
4. Acceptance Testing	User Acceptance Testing (UAT)	To validate the system for user needs and to obtain acceptance.	Business requirements, real world scenarios, user satisfaction.	End Users / Clients	After system testing.	Bank staff using the system and approving it for live use.

5. Comparison

Basis	Unit	Integration	System	Acceptance
Test Object	Unit / Module	Modules	Whole System	Business Need
Performed By	Developers	Testers	Testers	Users / Clients
Purpose	Verify code	Verify interfaces	Validate system	Validate with real use
Scope	Small	Medium	Large	Business oriented
Execution	Early	Next	Later	Last
Example	Test a method	Test module interaction	Test all features	User checks and accepts

6. Importance of Different Levels

- Early defect detection reduces cost and time.
- Each level ensures quality at a different granularity.
- Improves reliability, usability and performance.
- Builds confidence for users and stakeholders.
- Final acceptance ensures that system meets business goals.

7. In Short

Testing levels provide a step-by-step approach to validate software from individual components to the final system used by the customer.

Remember : Early Testing → Less Cost → Better Quality → Happy Customer ☺

RGPV NOTES HUB
Smart Notes, Better Scores

UNIT - 4 : SOFTWARE TESTING

3. WHITE BOX TESTING

RGPV (CS/IT)
UNIT - 4, TOPIC - 3
DATE : __ / __ / __

1. Introduction

White Box Testing is a software testing technique in which the internal structure / working / logic of the system is known to the tester. Tester checks the internal code, structure, paths, conditions, loops, etc.
It is also called Structural Testing or Clear Box Testing.

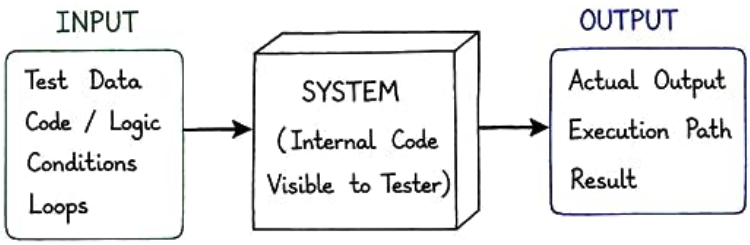
2. Key Points

- ★ Tester has full knowledge of internal code / structure.
- ★ Focus is on how the system does.
- ★ Test cases are designed using code, logic and control flow.
- ★ Helps in finding errors in logic, design and coding.
- ★ Usually performed by developers or test engineers.
- ★ Done at Unit / Component level.

3. Characteristics

- → Internal structure / code is visible to the tester.
- → Based on program logic, control flow and paths.
- → Test cases are derived from source code.
- → Helps in verifying internal working and hidden errors.
- → Measures coverage (statement, branch, path, etc.).
- → Usually performed at early stages (Unit Testing).

4. How White Box Testing Works



Internal code / path is visible and tested.

5. Techniques of White Box Testing

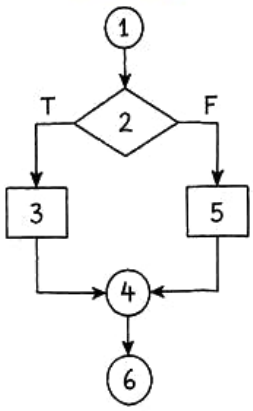
Technique	Description
Statement Coverage	Every executable statement in the program is executed at least once.
Branch Coverage	Every branch (True / False path) is executed at least once.
Condition Coverage	Every condition in decision is evaluated to both True and False.
Path Coverage	Every possible path in the program is executed.
Loop Testing	Loops are tested for 0, 1, 2, ..., n times.
Data Flow Testing	Based on definition and use of variables.
Control Flow Testing	Based on flow of control (sequence, selection, iteration).

6. Example

Program Code :

```
if (A > B)
  C = A - B;
else
  C = A + B;
D = C * 2;
return D;
```

Flow Graph :



Test Cases (White Box):

TC ID	A	B	Path Covered
TC01	10	5	1-2(T)-3-4-6
TC02	2	8	1-2(F)-5-4-6
TC03	5	5	1-2(F)-5-4-6

(T = True , F = False)

7. Coverage Criteria (Brief)

Coverage	What it Means
Statement	Each statement is executed at least once.
Branch	Each branch (True/False) is executed.
Condition	Each condition is True and False at least once.
Path	All possible independent paths are executed.

8. Advantages

- ✓ Finds errors in logic and design.
- ✓ Checks internal security and invalid paths.
- ✓ Improves code quality.
- ✓ Helps in removing dead code.
- ✓ High fault detection capability.
- ✓ Can test hidden functionality.

9. Disadvantages

- × Requires knowledge of programming.
- × Time consuming.
- × Not suitable for large codes (all paths).
- × High effort and cost.
- × Cannot be used without code.

10. When to Use

- • During Unit / Component Testing.
- • When code / design is available.
- • To test internal logic, loops, conditions.
- • To find hidden errors.

11. In Short

White Box Testing is a structural testing technique where tester verifies internal code, logic, paths and design. It ensures that each part of the code works correctly and all paths are properly executed.

Remember : Test the Code Inside → Improve Logic → Remove Defects Early → Quality Software ☺

RGPV NOTES HUB

Smart Notes, Better Scores

UNIT - 4 : SOFTWARE TESTING

5. INTEGRATION TESTING

RGPV (CS/IT)
UNIT - 4, TOPIC - 5
DATE : __/__/__

1. Introduction

Integration Testing is a level of software testing where individual modules or components are combined and tested as a group.

The main objective is to check the interaction between integrated modules.

It is done after Unit Testing and before System Testing.

2. Key Points

- ★ Combines modules and tests their interfaces.
- ★ Detects interface defects in data flow or control flow.
- ★ Helps to identify defects in communication between modules.
- ★ Testing is done incrementally.
- ★ Ensures that integrated modules work together as expected.

3. Objectives

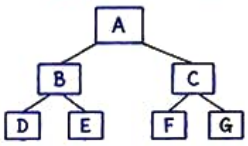
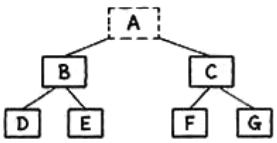
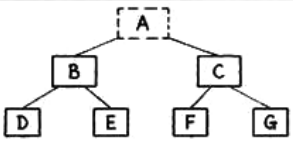
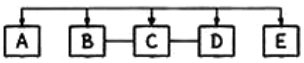
- To test the interface between integrated modules.
- To check the data flow and control flow.
- To find defects in module interaction.
- To ensure that modules work together as intended.

4. When Performed ?

Unit Testing → Integration Testing → System Testing → Acceptance Testing

(After Unit Testing and before System Testing)

5. Types of Integration Testing

Type	Description	Approach	Diagram (Example)	Advantages
1. Top-Down	Testing starts from Top (main module) and moves down to lower modules.	Top level modules are tested first by using stubs.		• Early test of important control paths. • Less number of stubs required.
2. Bottom-Up	Testing starts from lower modules and moves up to higher modules.	Lower level modules are tested first by using drivers.		• Early test of utility modules. • More number of drivers required.
3. Sandwich / Hybrid	Combination of Top-Down and Bottom-Up approach.	Testing is done both from top and bottom till they meet in the middle.		• Overcomes drawbacks of both approaches. • More effective.
4. Big Bang	All modules are combined together and tested.	No need of stubs or drivers.		• Simple to use. • Difficult to find defects.

6. Advantages

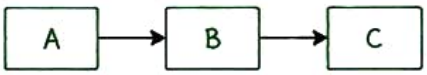
- ✓ Defects in interface are identified.
- ✓ Ensures proper interaction between modules.
- ✓ Early detection reduces cost and time.
- ✓ Improves reliability of the system.

7. Limitations

- × More time consuming.
- × Requires planning and effort.
- × Many test cases are needed.
- × Complex for large systems.

8. Example

Consider 3 modules A, B and C.
A sends data to B and B sends data to C.
Integration Test checks whether data is passed correctly from A → B → C.



9. In Short

Integration Testing ensures that when individual modules are combined, they interact and work together as expected.

TOPIC 5 : PROJECT SCHEDULING

Project Scheduling is the process of determining the order of activities, estimating the time required and allocating resources to complete the project within the given time.

1. Objectives

- To complete the project on time.
- To optimize the use of resources.
- To identify the critical activities.
- To monitor and control the project progress.
- To reduce project cost and risk.

2. Steps in Project Scheduling

1. List all activities.
2. Determine the sequence of activities.
3. Estimate the time for each activity.
4. Allocate resources.
5. Prepare the schedule.
6. Monitor and update the schedule.

3. Techniques / Methods

- Gantt Chart
- PERT (Program Evaluation and Review Technique)
- CPM (Critical Path Method)
- Milestone Chart
- Network Diagram

4. Gantt Chart (Example)

Activity	Duration (Days)	Time (Days) 1	2	3	4	5	6	7	8	9	10
A	3	▓	▓	▓							
B	2		▓	▓							
C	4			▓	▓	▓	▓				
D	3					▓	▓	▓			
E	3								▓	▓	▓

5. Important Points

- Critical activities determine the minimum project completion time.
- Delays in critical activities delay the entire project.
- Slack time = Difference between earliest and latest start time.
- Proper scheduling improves quality and productivity.

6. Benefits

- Helps in planning and controlling the project.
- Ensures timely completion.
- Better utilization of resources.
- Helps in identifying risks early.
- Improves communication among team members.

7. In Short

Project Scheduling acts as a roadmap for the project. It helps in completing the project on time, within budget and with proper quality by effective planning, monitoring and control.

★ Plan your work today, schedule it well, succeed tomorrow. ☺

RGPV NOTES HUB

TOPIC 6 : COST ESTIMATION

Cost Estimation is the process of predicting the total cost required to develop, operate and maintain a software system. It helps in budgeting, planning and resource allocation.

1. Objectives

- To predict the total cost of the project.
- To plan the budget and resources.
- To help in decision making.
- To control cost overrun.
- To improve accuracy in future estimation.

2. Cost Components

Component	Description
Development Cost	Cost of analysis, design, coding, testing and integration.
Operational Cost	Cost of running the system in the organization.
Maintenance Cost	Cost of correcting errors, adapting and enhancing the system.
Support Cost	Cost of user support and training.
Hardware & Software Cost	Cost of tools, hardware, software and other resources.

3. Cost Estimation Methods

- Algorithmic (Model Based) Methods
- Expert Judgment Methods
- Estimation by Analogy
- Top-Down and Bottom-Up Estimation
- Parametric Models (e.g., COCOMO)
- Use Case Points

4. Algorithmic (Model Based) Method – COCOMO

COCOMO (Constructive Cost Model) is an algorithmic model developed by Barry Boehm.

Basic COCOMO

$$\text{Effort (PM)} = a \times (\text{KLOC})^{b}$$

$$\text{Time } (T_{DEV}) = c \times (\text{Effort})^{d}$$

Where,
PM = Person Months
KLOC = Thousands of Lines of Code
T_{DEV} = Development Time (Months)
a, b, c, d = Constants

5. Estimation by Analogy

In this method, cost of the current project is estimated by comparing it with similar past projects whose cost is already known.

Steps :

1. Select similar past projects.
2. Compare size, complexity and environment.
3. Adjust for differences.
4. Calculate estimated cost.

6. Top-Down vs Bottom-Up Estimation

Top-Down Estimation	Bottom-Up Estimation
Entire system is considered as a single unit.	System is divided into modules or components.
Quick and easy.	More time consuming.
Less accurate.	More accurate.
Used in early stages.	Used in detailed planning.

7. Factors Affecting Cost Estimation

- Size of the software
- Complexity of the system
- Reliability and quality requirements
- Technology and tools to be used
- Experience of the development team
- Customer requirements and changes
- Platform and hardware constraints

8. Benefits

- ✓ Helps in budgeting and financial planning.
- ✓ Prevents cost overrun.
- ✓ Helps in effective resource allocation.
- ✓ Supports management decision making.
- ✓ Improves estimation accuracy in future projects.

★ Accurate cost estimation leads to successful project management. ☺

RGPV NOTES HUB