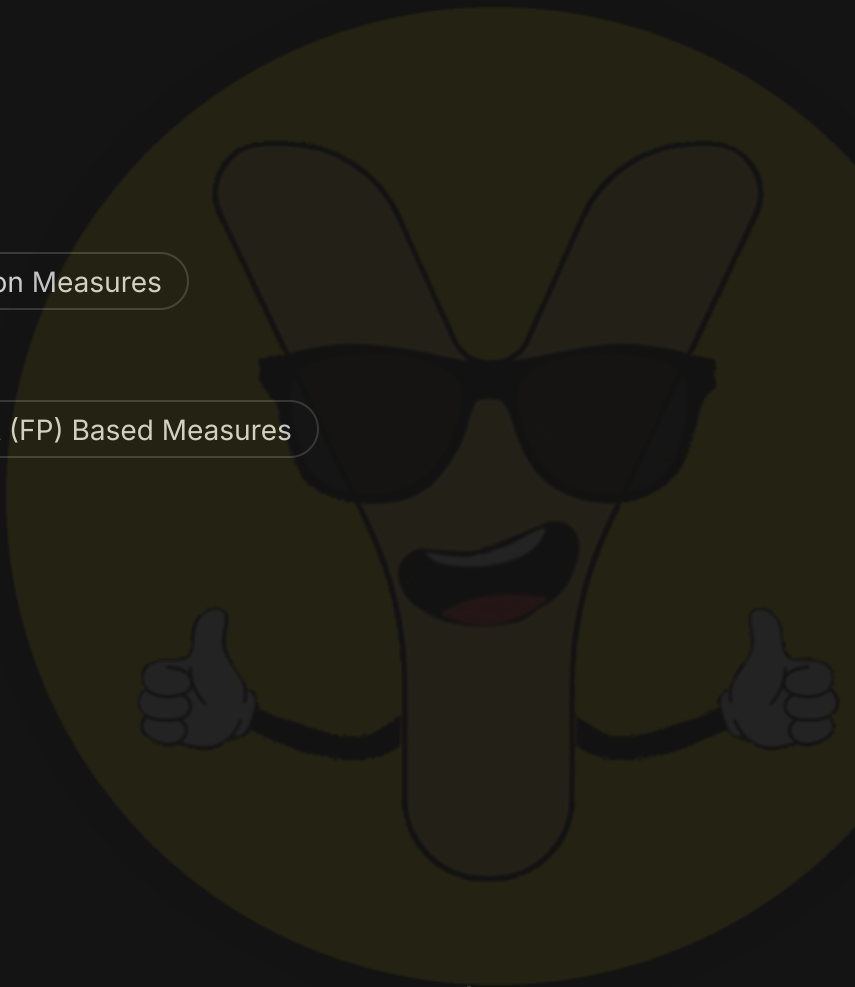
AKTU · 3RD YEAR · SEMESTER 6 · BCS601

Software Engineering

Unit 3 — Software Design & Software Metrics

- Basic Concept of Software Design
- Architectural Design
- Low Level Design: Modularization, Design Structure Charts, Pseudo Codes, Flow Charts
- Coupling and Cohesion Measures
- Design Strategies: Function Oriented Design, Object Oriented Design, Top-Down and Bottom-Up Design
- Software Measurement and Metrics
- Size Oriented Measures: Halstead's Software Science
- Function Point (FP) Based Measures



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

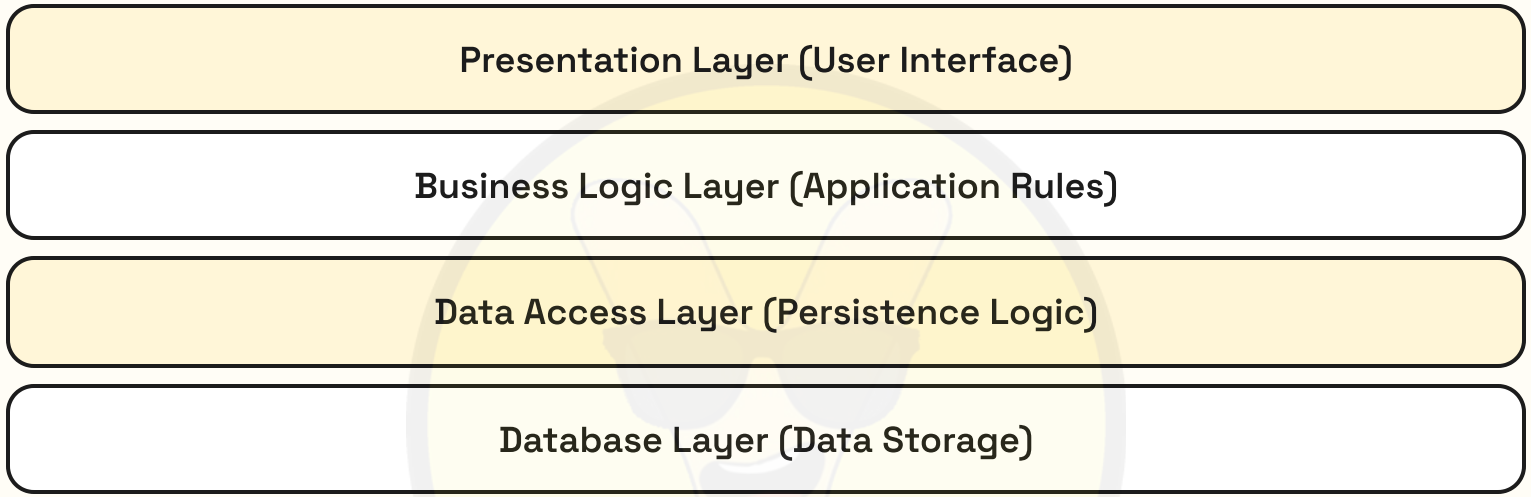
Basic Concepts of Software Design

SOFTWARE DESIGN

Software design is the iterative process of transforming user requirements (SRS) into a blueprint for constructing the software system.

- Abstraction allows focusing on essential features without including background details.
- Refinement systematically decomposes high-level functions into lower-level detailed steps.
- Modularity divides software into independently named and addressable components.
- Information hiding conceals internal module details behind explicit interfaces.

Architectural Design - Layered Pattern



Architectural design defines the high-level structure, subsystem organization, and communication rules between components.

Architectural Design Styles

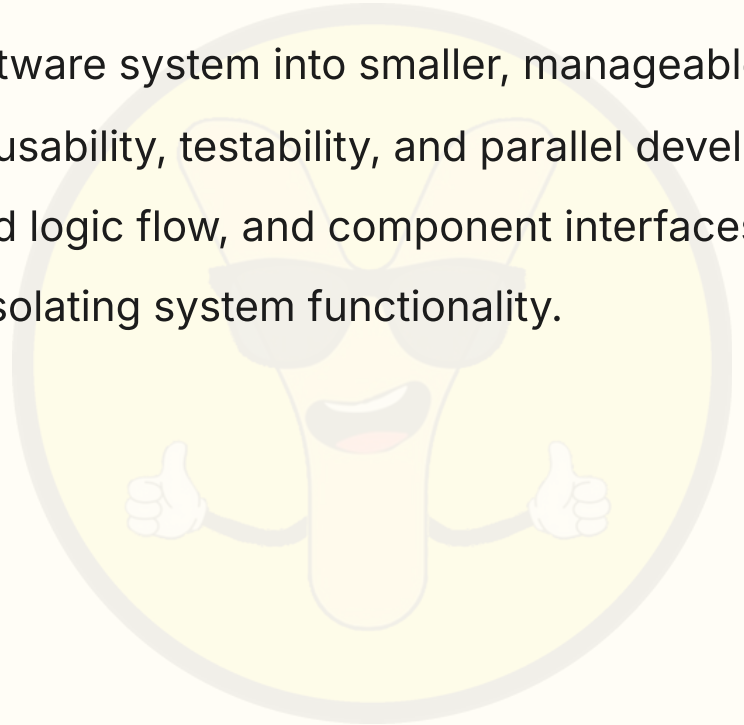
Architectural styles define overall system structures and control mechanisms.

- Data-Centered: Central repository shared by multiple client components.
- Data-Flow (Pipe and Filter): Inputs are transformed through a series of processing components.
- Call and Return: Main program invokes sub-programs hierarchically.
- Layered Architecture: Modules organised into hierarchical layers offering services to adjacent layers.
- Object-Oriented: Components encapsulate data and operations; communicate via message passing.

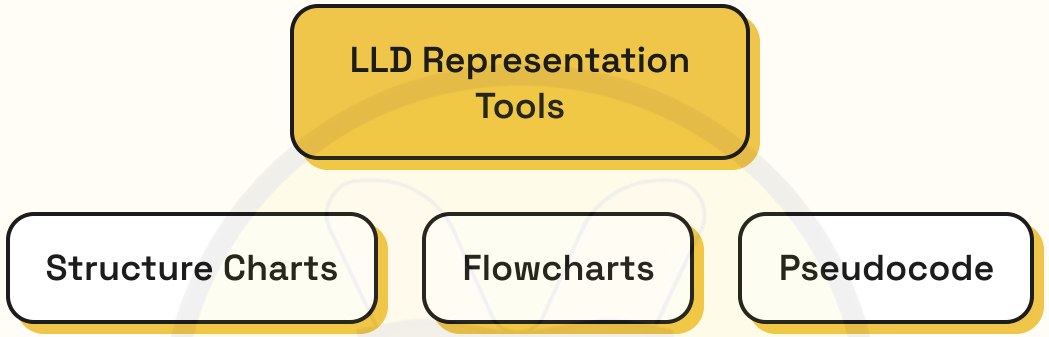
Low-Level Design & Modularization

Low-Level Design (LLD) focuses on detailed algorithmic design and internal structures.

- Modularization splits a complex software system into smaller, manageable units.
- Improves system maintainability, reusability, testability, and parallel development.
- Focuses on data structures, detailed logic flow, and component interfaces.
- Reduces mental cognitive load by isolating system functionality.



Low-Level Design Tools Overview



Structure Charts show structural control hierarchy; Flowcharts map visual execution flow; Pseudocode provides logic specification.

Cohesion in Software Design

Cohesion measures the strength of functional relationships among elements inside a single module.

- High Cohesion is desirable as it indicates a single, well-defined task.
- Functional (Best): All elements perform a single, targeted task.
- Sequential: Output of one element acts as input to the next.
- Communicational: Elements operate on identical input or output data.
- Procedural: Elements execute in a specific sequence of steps.
- Coincidental (Worst): Elements are grouped with no meaningful relation.

Coupling in Software Design

Coupling measures the degree of interdependence between two distinct software modules.

- Low Coupling is desirable to minimize ripple effects during software modification.
- Data Coupling (Best): Modules communicate passing simple primitive parameters.
- Stamp Coupling: Modules share complex composite data structures.
- Control Coupling: Module controls internal flow of another passing flag bits.
- Common Coupling: Modules share global data areas.
- Content Coupling (Worst): One module directly accesses or modifies another module's code.

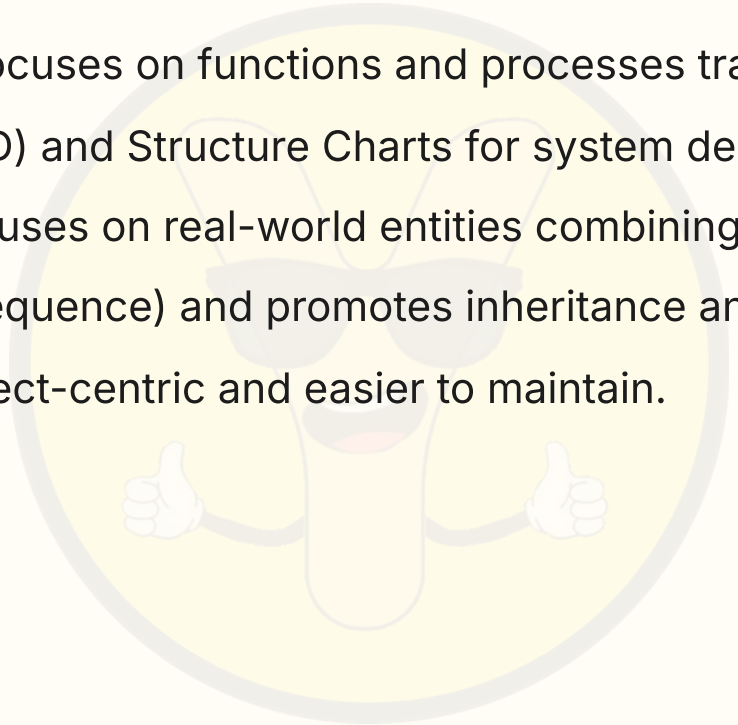
Comparison: Coupling vs Cohesion

Parameter	Cohesion	Coupling
Definition	Intra-module strength (internal binding)	Inter-module dependency between modules
Design Goal	Maximize (High Cohesion)	Minimize (Low Coupling)
Ideal Level	Functional Cohesion	Data Coupling
Worst Level	Coincidental Cohesion	Content Coupling
Focus	Single module's internal purpose	Interaction between different modules
Impact	Increases reusability & clarity	Reduces impact of changes across modules

Design Strategies: Function vs Object Oriented

Design strategies determine how software systems are decomposed.

- Function-Oriented Design (FOD): Focuses on functions and processes transforming data.
- FOD Uses Data Flow Diagrams (DFD) and Structure Charts for system decomposition.
- Object-Oriented Design (OOD): Focuses on real-world entities combining state and operations.
- OOD uses UML diagrams (Class, Sequence) and promotes inheritance and polymorphism.
- FOD is process-centric; OOD is object-centric and easier to maintain.



Design Strategies: Top-Down vs Bottom-Up

Attribute	Top-Down Strategy	Bottom-Up Strategy
Starting Point	High-level system modules down to low-level	Low-level basic modules up to system level
Core Approach	Progressive refinement and decomposition	Composition of existing or base components
Suitability	New systems with well-defined specs	Systems reusing existing component libraries
Testing Tools	Requires Stubs for lower modules	Requires Drivers for higher modules
Redundancy	Low risk of duplicate functionality	Higher risk of creating redundant code

Software Measurement and Metrics

SOFTWARE METRICS

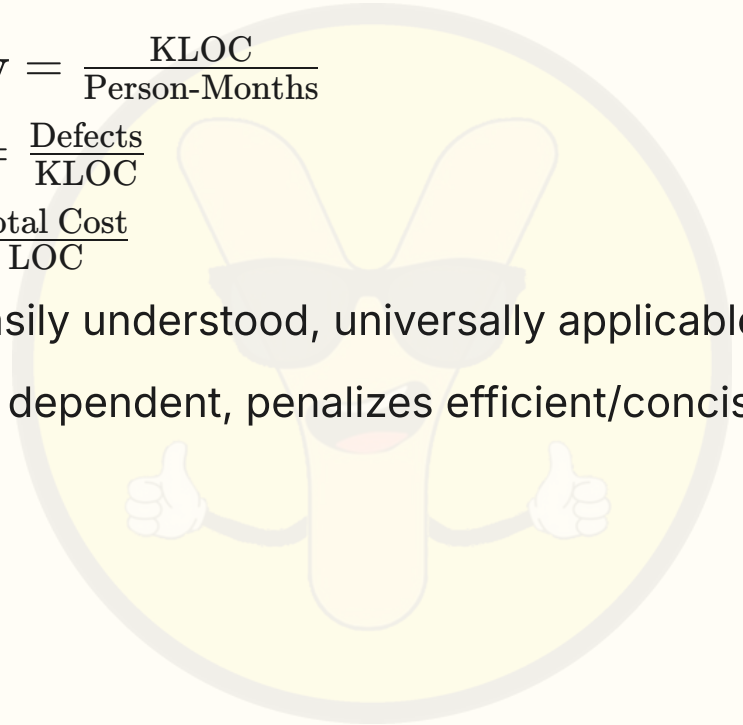
Quantitative measures of software characteristics used to evaluate, estimate, and control software development processes and products.

- Direct Measures: Directly measurable attributes like Lines of Code (LOC) and memory usage.
- Indirect Measures: Derived attributes like software quality, maintainability, and reliability.
- Process Metrics: Track efficiency and performance of development activities.
- Product Metrics: Evaluate state, size, and complexity of software deliverables.

Size-Oriented Measures (LOC)

Size-oriented metrics collect normalized productivity and quality metrics derived from Lines of Code.

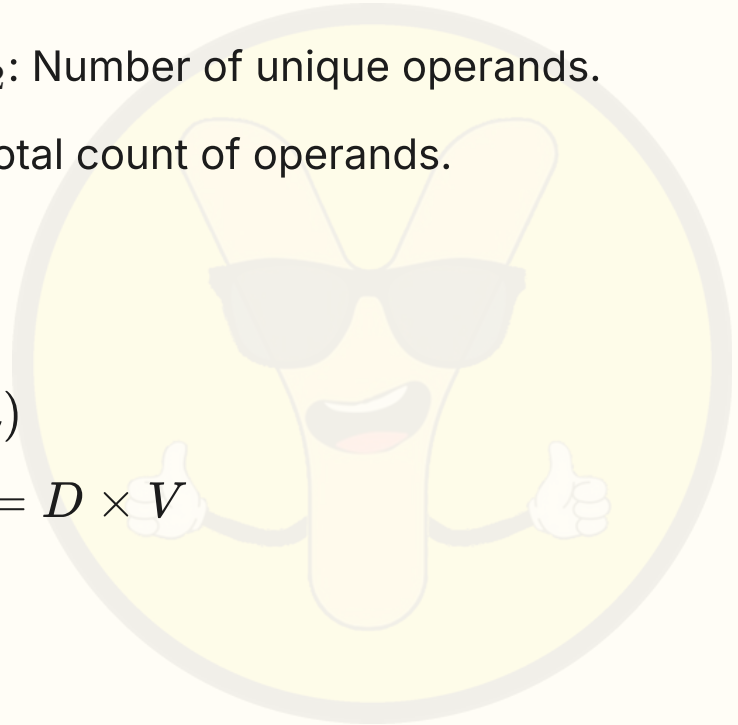
- Productivity Formula: $\text{Productivity} = \frac{\text{KLOC}}{\text{Person-Months}}$
- Quality Formula: $\text{Defect Density} = \frac{\text{Defects}}{\text{KLOC}}$
- Cost Formula: $\text{Cost per LOC} = \frac{\text{Total Cost}}{\text{LOC}}$
- Advantages: Simple to calculate, easily understood, universally applicable.
- Limitations: Programming language dependent, penalizes efficient/concise code.



Halstead's Software Science

Halstead metrics measure program complexity based on counting primitive tokens.

- n_1: Number of unique operators; n_2: Number of unique operands.
- N_1: Total count of operators; N_2: Total count of operands.
- Program Vocabulary: $n = n_1 + n_2$
- Program Length: $N = N_1 + N_2$
- Program Volume: $V = N \times \log_2(n)$
- Difficulty: $D = \frac{n_1}{2} \times \frac{N_2}{n_2}$; Effort: $E = D \times V$



Halstead's Science: Worked Example

Calculate Vocabulary, Length, and Volume for a C snippet: int max(int a, int b) { if (a > b) return a; else return b; }

1. Identify unique operators (n1): int, max, (), {}, if, >, return, else, ; → n1 = 9
2. Identify total operators (N1): int(3), max(1), ()(2), {}(1), if(1), >(1), return(2), else(1), ;(2) → N1 = 14
3. Identify unique operands (n2): a, b → n2 = 2
4. Identify total operands (N2): a(3), b(3) → N2 = 6
5. Program Vocabulary (n): n = n1 + n2 = 9 + 2 = 11
6. Program Length (N): N = N1 + N2 = 14 + 6 = 20
7. Program Volume (V): V = N * log2(n) = 20 * log2(11) = 20 * 3.459 = 69.19 bits

Vocabulary (n) = 11, Length (N) = 20, Volume (V) = 69.19 bits

Function Point (FP) Analysis

Function Points measure functional size independent of programming implementation language.

- Based on 5 Information Domain Characteristics:
- External Inputs (EI): User inputs updating internal tables.
- External Outputs (EO): Generated reports or formatted outputs.
- External Inquiries (EQ): Online inputs resulting in simple data retrieval.
- Internal Logical Files (ILF): Logical groups of data inside system.
- External Interface Files (EIF): Data shared from external systems.
- Calculated as: $\mathrm{FP} = \mathrm{UFP} \times (0.65 + 0.01 \times \sum F_i)$

Function Point: Worked Numerical

A software project has Unadjusted Function Points (UFP) = 150. The sum of 14 Value Adjustment Factors (Sum Fi) is 35. Calculate Adjusted FP.

1. Given Unadjusted Function Points (UFP) = 150
2. Given total degree of influence $\sum F_i = 35$
3. Calculate Value Adjustment Factor (VAF): $\text{VAF} = 0.65 + 0.01 \times \sum F_i$
4. $\text{VAF} = 0.65 + 0.01 \times 35 = 0.65 + 0.35 = 1.00$
5. Compute Adjusted Function Point: $\text{FP} = \text{UFP} \times \text{VAF}$
6. $\text{FP} = 150 \times 1.00 = 150$

Adjusted Function Point (FP) = 150

Control Flow Graphs & Cyclomatic Complexity

Cyclomatic Complexity measures the logical complexity of code using Control Flow Graphs (CFG).

- Control Flow Graph (CFG): Graph representing execution paths using nodes (statements) and edges (control transfers).
- Method 1 Formula: $V(G) = E - N + 2P$ (E = Edges, N = Nodes, P = Connected Components).
- Method 2 Formula: $V(G) = P_{pred} + 1$ (P_pred = Predicate/Decision nodes).
- Method 3 Formula: $V(G) = \text{Number of Bounded Regions} + 1$.
- Determines the upper bound for the number of test cases needed for branch coverage.

Cyclomatic Complexity: Worked Numerical

A program graph has 11 nodes, 13 edges, and 3 predicate nodes in a single component (P=1). Compute Cyclomatic Complexity V(G).

1. Method 1: Using Edge-Node formula $V(G) = E - N + 2P$
2. Substitute values: $V(G) = 13 - 11 + 2(1) = 2 + 2 = 4$
3. Method 2: Using Predicate Node formula $V(G) = P_{pred} + 1$
4. Substitute values: $V(G) = 3 + 1 = 4$
5. Result verified: Both methods yield the exact same complexity score.

Cyclomatic Complexity V(G) = 4 (Requires 4 independent test paths)

Exam Corner: High-Yield AKTU Questions

Q1 Explain Coupling and Cohesion. Describe different types of cohesion and coupling with appropriate diagrams.

Q2 Differentiate between Function-Oriented Design and Object-Oriented Design.

Q3 Compare Top-Down and Bottom-Up design strategies with suitable examples.

Q4 Define Halstead's Software Science metrics. Write all formulas for length, volume, effort, and difficulty.

Q5 Explain Function Point Analysis. Discuss the 5 information domain characteristics and compute FP given UFP.

Q6 What is Cyclomatic Complexity? Explain three methods to calculate V(G) using Control Flow Graphs.

Q7 A program has 14 edges and 10 nodes with 1 connected component. Calculate its cyclomatic complexity and state its importance in software testing.

Unit 3 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 4 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

