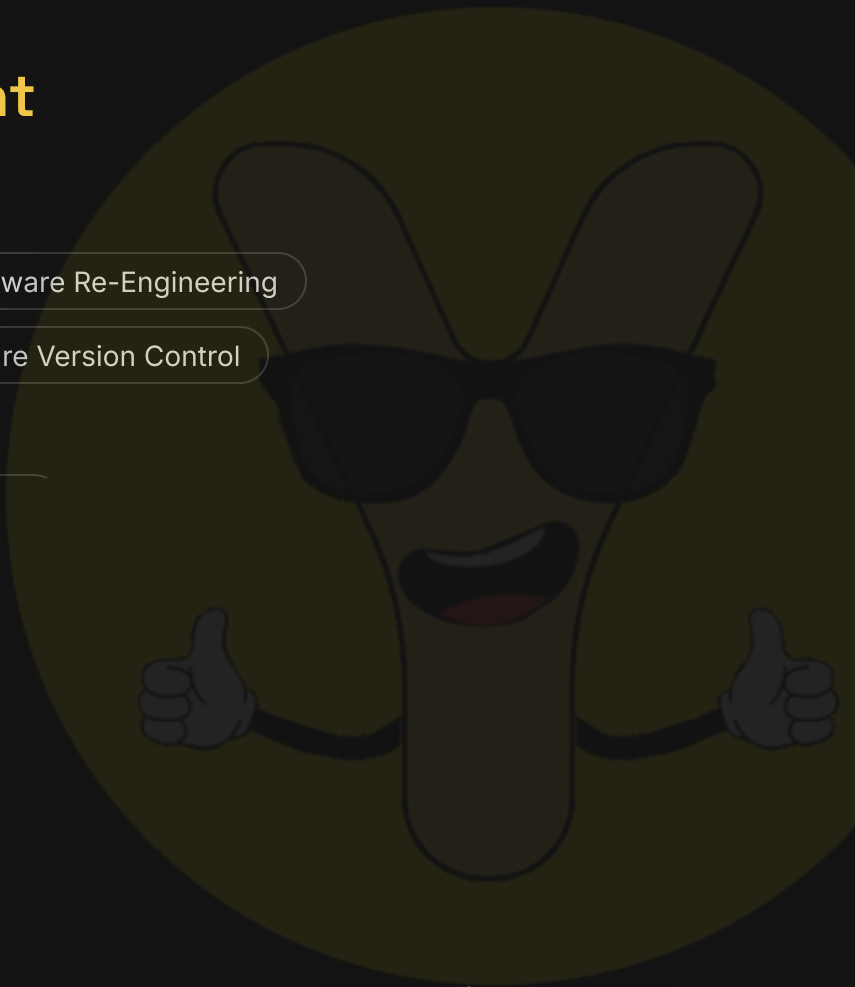
AKTU · 3RD YEAR · SEMESTER 6 · BCS601

Software Engineering

Unit 5 — Software Maintenance & Project Management

- Software as an Evolutionary Entity
- Need for Maintenance
- Categories of Maintenance: Preventive, Corrective and Perfective Maintenance
- Cost of Maintenance
- Software Re-Engineering
- Reverse Engineering
- Software Configuration Management Activities
- Change Control Process
- Software Version Control
- An Overview of CASE Tools
- Estimation of Various Parameters: Cost, Efforts, Schedule/Duration



DECODED BY

Chirag Jain

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

Software as an Evolutionary Entity

Software systems must continually evolve to remain useful in changing operational environments.

- Lehman's First Law (Continuing Change): A system must undergo continuous adaptation or become progressively less useful.
- Lehman's Second Law (Increasing Complexity): As a system evolves, its complexity increases unless work is done to maintain or reduce it.
- Software decay occurs when system structures degrade due to repeated emergency fixes.
- Evolutionary view treats maintenance not as a post-project phase, but as continuous development.

Need and Cost of Software Maintenance

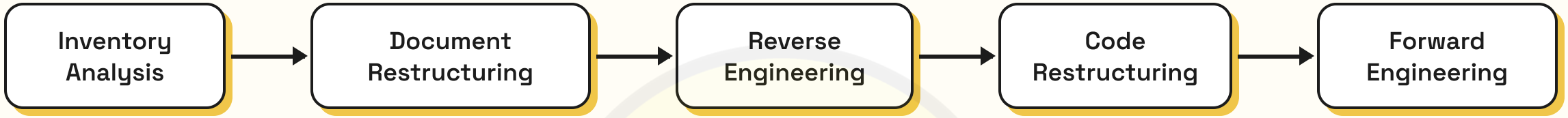
Software maintenance accounts for 60% to 80% of total software lifecycle costs.

- Correcting system bugs and defects reported after delivery.
- Adapting software to updated operating systems, hardware, or third-party APIs.
- Enhancing system features to fulfill evolving business requirements.
- Cost Drivers: Poor documentation, complex legacy code, high staff turnover, and lack of design standards.
- Maintenance Cost Formula Concept: $Maintenance\ Cost = Development\ Cost \times Multiplier$, where multiplier grows exponentially with legacy age.

Categories of Software Maintenance

Category	Objective	Trigger / Cause	Example
Corrective	Fix reported faults/bugs	System crash, logic error	Fixing a memory leak or null pointer exception
Adaptive	Adapt to environment changes	OS upgrade, hardware change	Porting a legacy application from Windows 10 to 11
Perfective	Enhance performance/features	User request for improvements	Optimizing SQL query execution time
Preventive	Prevent future failures	Code smells, structural debt	Refactoring monolithic code to modular functions

Software Re-Engineering Process



Sequential phases involved in re-engineering legacy software to modern standards.

Reverse Engineering

REVERSE ENGINEERING

The process of analyzing a target system to identify its components and interrelationships, creating representations of the system at higher levels of abstraction.

- Extracts design specifications directly from existing source code.
- Does not alter the target system or generate new code.
- Levels of abstraction: Program structure, Data structure, System architecture.
- Primary Goal: Enhance system understandability for maintenance and re-documentation.

Comparison of Software Engineering Approaches

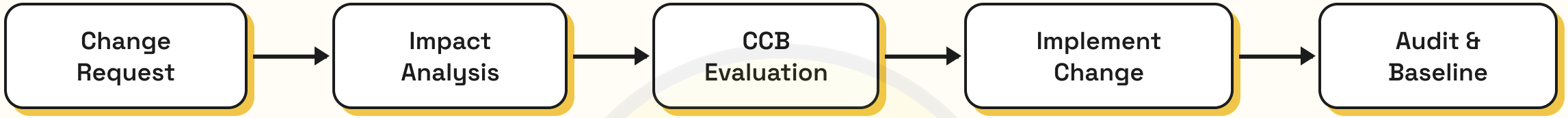
Parameter	Forward Engineering	Reverse Engineering	Re-Engineering
Direction	High-level specs to Code	Code to High-level design	Code → Design → New Code
Starting Point	System Requirements	Existing Source Code	Legacy Software System
Primary Output	Executable Software System	Design Models & Specs	Modernized Target System
Core Objective	Build a new software system	Understand existing code	Improve quality & maintainability

Software Configuration Management (SCM) Activities

SCM provides systematic control over changes to software artifacts throughout the lifecycle.

- Configuration Identification: Defining baselines and labeling Configuration Items (SCIs).
- Version Control: Managing multiple variants and tracking change history.
- Change Control: Formal evaluation, approval, and tracking of modification requests.
- Configuration Auditing: Ensuring implementation matches functional and technical specifications.
- Status Reporting: Recording and reporting the status of proposed and implemented changes.

SCM Change Control Process Flow

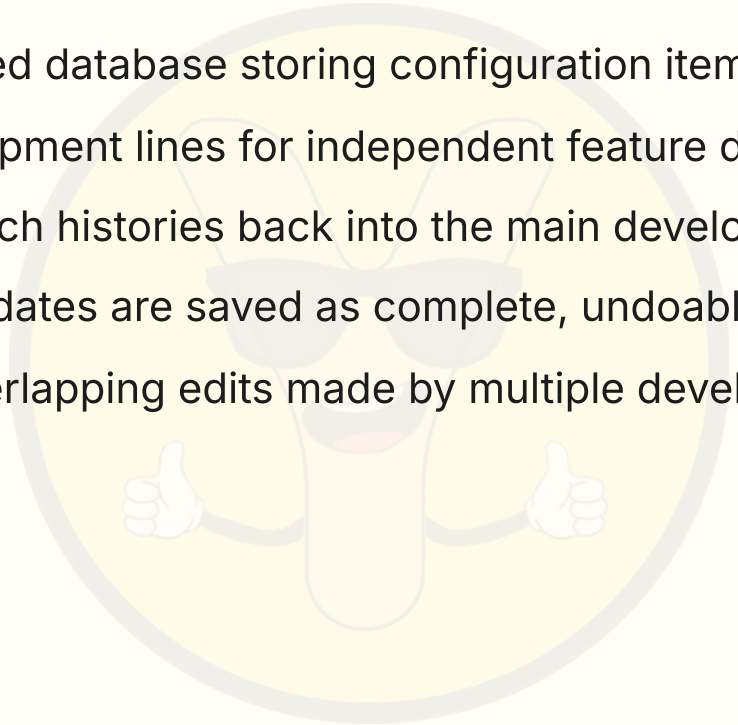


Standard workflow of the Change Control Board (CCB) for processing software modifications.

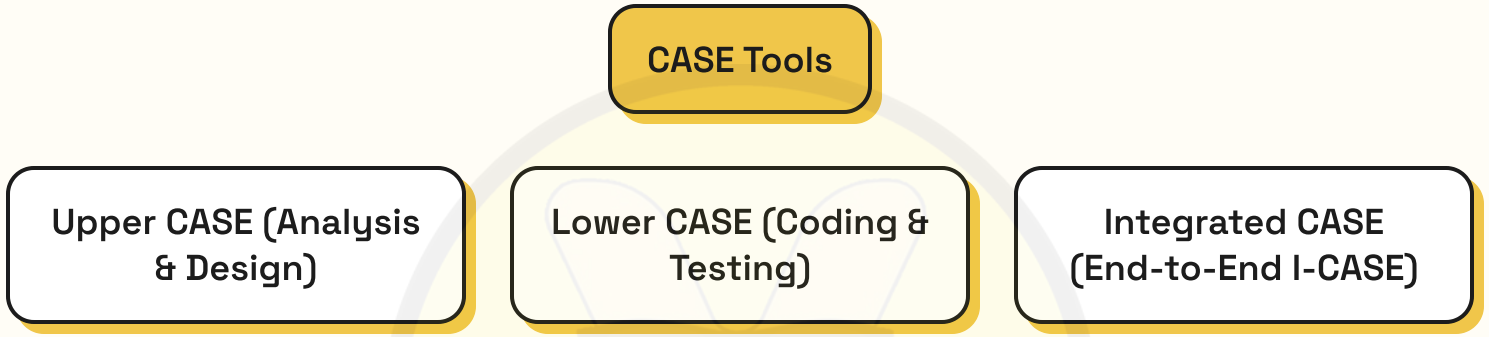
Software Version Control

Version control systems manage concurrent development and maintain revision history.

- Repository: Centralized or distributed database storing configuration items and commit history.
- Branching: Creating isolated development lines for independent feature development.
- Merging: Combining divergent branch histories back into the main development trunk.
- Atomic Commits: Ensuring code updates are saved as complete, undoable historical units.
- Conflict Resolution: Reconciling overlapping edits made by multiple developers to identical files.



Classification of CASE Tools



Categorization of Computer-Aided Software Engineering tools across the development lifecycle.

Software Project Estimation Parameters

Software project management relies on accurate estimation of Effort, Duration, and Cost.

- Effort: Total labor required, expressed in Person-Months (PM).
- Duration / Schedule: Total calendar time required to complete the project, in Months.
- Cost: Computed as $Cost = Total\ Effort \times Average\ Staff\ Salary$.
- Software Size: Measured in Source Lines of Code (SLOC/KLOC) or Function Points (FP).
- Productivity Rate: Measured in \(KLOC / Person\-Month\) or \(FP / Person\-Month\).

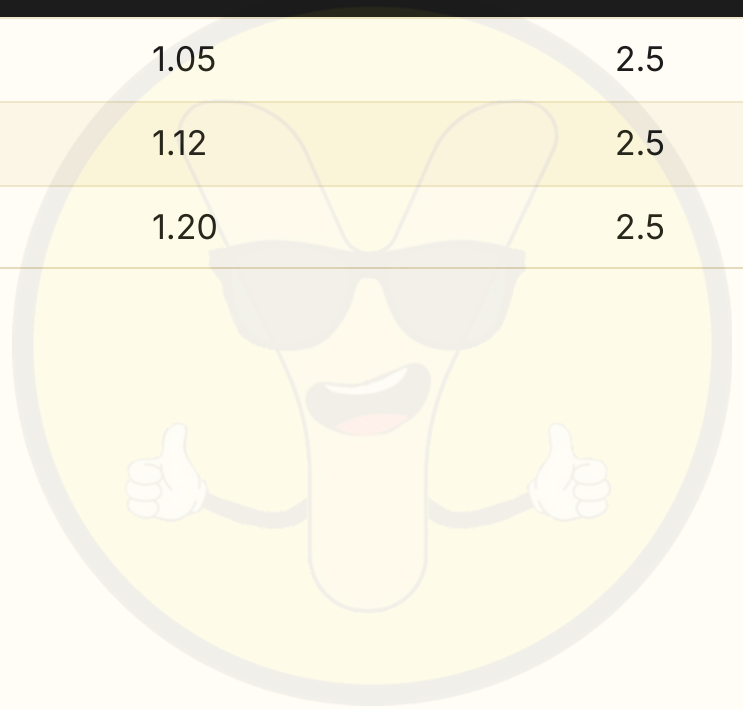
COCOMO Model: Modes and Basic Formulas

Boehm's Constructive Cost Model (COCOMO) estimates effort and schedule using KLOC.

- Organic Mode: Small team, well-understood domain, flexible requirements.
- Semi-Detached Mode: Medium team, mixed experience, medium complexity.
- Embedded Mode: Tight hardware/software constraints, highly complex requirements.
- Basic Effort Equation: $E = a \times (KLOC)^b$ in Person-Months.
- Basic Schedule Equation: $T_{dev} = c \times (E)^d$ in calendar Months.

COCOMO Modes and Coefficients

Project Mode	a (Effort)	b (Exponent)	c (Schedule)	d (Exponent)
Organic	2.4	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32



Worked Example: Basic COCOMO Calculation

Estimate the Effort, Development Time, and Staff Size for an Organic mode software system estimated at 32 KLOC.

1. Step 1: Identify Organic mode constants: $a = 2.4, b = 1.05, c = 2.5, d = 0.38$.
2. Step 2: Calculate Effort: $E = 2.4 \times (32)^{1.05} = 2.4 \times 37.89 = 90.93$ Person-Months.
3. Step 3: Calculate Development Time: $T_{dev} = 2.5 \times (90.93)^{0.38} = 2.5 \times 5.55 = 13.88$ Months.
4. Step 4: Calculate Staff Size: $Persons = \frac{E}{T_{dev}} = \frac{90.93}{13.88} \approx 6.55$ Persons.

Effort = 90.93 Person-Months, Duration = 13.88 Months, Team Size = ~7 Persons.

Intermediate and Detailed COCOMO Models

Advanced COCOMO models refine basic estimates using cost driver attributes.

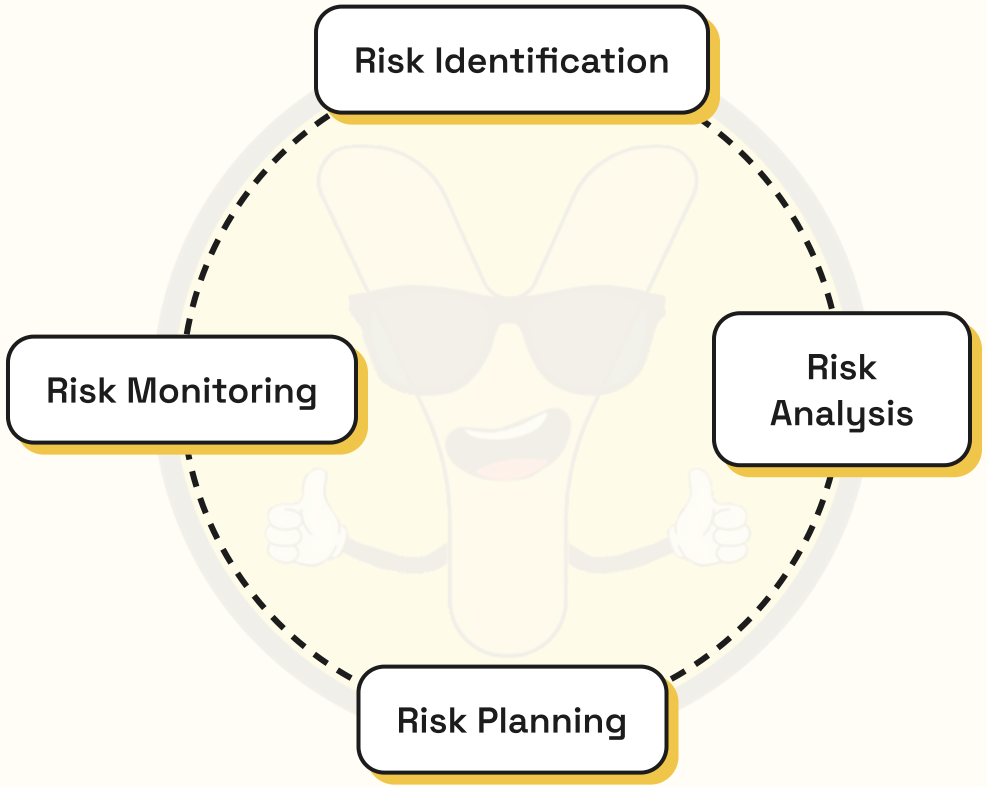
- Intermediate COCOMO incorporates 15 Cost Drivers (Cost-Driver Attributes) across 4 categories.
- Cost Driver Categories: Product, Hardware, Personnel, and Project attributes.
- Effort Adjustment Factor (EAF): Product of all 15 rating values: $EAF = \prod_{i=1}^{15} CD_i$.
- Intermediate Effort Equation: $E = a \times (KLOC)^b \times EAF$.
- Detailed COCOMO applies cost driver multipliers at individual phases (Requirements, Design, Code, Test).

Resource Allocation Models

Resource allocation models map staffing distribution over the project timeline.

- Norden-Rayleigh Model: Describes project manpower distribution over time as a bell curve.
- Rayleigh Curve Equation: $Y(t) = 2Kate^{-at^2}$, where K is effort and t is time.
- Peak Staffing: Manpower peaks during the coding and unit testing phases.
- Putnam's Software Equation: $S = C_k \times K^{1/3} \times t_d^{4/3}$, linking size S, effort K, and time t_d.
- Trade-off Law: Compressing schedule drastically increases required total effort.

Software Risk Management Cycle



Continuous lifecycle steps for identifying and mitigating project risks.

Risk Mitigation, Monitoring, and Management (RMMM)

An RMMM plan establishes proactive procedures to address potential project threats.

- Risk Mitigation: Action taken to reduce the probability or impact of a risk occurring.
- Risk Monitoring: Tracking key indicators to detect whether a risk is becoming more likely.
- Risk Management / Contingency: Execution plan if the risk materializes into an active issue.
- Risk Exposure Formula: $RE = P \times L$, where P is probability and L is potential loss.
- Risk Assessment Matrix: Evaluates risks based on Impact (High/Medium/Low) and Probability.

Exam Corner: AKTU Important Questions

- **Q1** Explain Lehman's laws of software evolution and discuss the need and cost drivers of software maintenance.
- **Q2** Differentiate between Corrective, Adaptive, Perfective, and Preventive software maintenance with suitable examples.
- **Q3** Describe Software Re-Engineering and Reverse Engineering processes with clear block diagrams.
- **Q4** What is Software Configuration Management (SCM)? Explain the change control process in detail.
- **Q5** Explain the Basic, Intermediate, and Detailed COCOMO models. Calculate Effort and Duration for a 50 KLOC Semi-detached system.
- **Q6** Explain Putnam's model and the Norden-Rayleigh resource allocation curve.
- **Q7** What is Risk Management? Explain Risk Identification, Risk Exposure calculation, and RMMM strategy.

Unit 5 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 1 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

