Software Engineering

Short notes for B.Tech CSE / IT · BCA · MCA · GATE CS

Uncodemy Handwritten Notes · free for students

1. Introduction

- **Software** = programs + documentation + data needed to run them.
- **Software engineering** = a systematic, disciplined and measurable way to develop, run and maintain software.
- **Software crisis**: projects late, over budget, low quality and hard to maintain. Software engineering came as the answer.

➤ Characteristics of software

- Software is developed, not manufactured.
- Software does not wear out, but it deteriorates because of changes (bathtub curve does not apply directly).
- Most software is still custom-built, though reuse is growing.

➤ Layers of software engineering

- Quality focus (base) → Process → Methods → Tools.

2. Software Process Models (SDLC)

- **SDLC** phases: requirement analysis → design → coding → testing → deployment → maintenance.

Model	Idea	Best when
Waterfall	Phases one after another, no going back	Requirements are clear and fixed
V-model	Each development phase has a matching test phase	Safety-critical, testing planned early
Incremental	Build in small working pieces	Early partial delivery needed
Prototyping	Build a quick model to clarify needs	Requirements are unclear
Spiral	Loops of planning, risk analysis, engineering, evaluation	Large, high-risk projects
RAD	Very fast, component-based, short cycles	Modular systems, tight time
Agile (Scrum, XP)	Short iterations, working software, customer feedback	Changing requirements

EXAM POINT Spiral model (Boehm) is called a meta-model because it can include other models. Its key feature is risk handling.

➤ Agile basics

- Agile values: individuals and interactions, working software, customer collaboration, responding to change.
- **Scrum**: sprints (usually 1–4 weeks), product backlog, sprint backlog, daily stand-up, sprint review, retrospective.
- Roles: Product Owner, Scrum Master, Development Team.
- **XP (Extreme Programming)**: pair programming, test-driven development, continuous integration, small releases.

3. Requirement Engineering

- Steps: elicitation → analysis → specification → validation → management.
- Elicitation methods: interviews, questionnaires, observation, brainstorming, use cases.

Functional requirement	Non-functional requirement
What the system must do	How well it must do it
"User can reset password"	"Page loads in under 2 seconds"
Features, services	Performance, security, usability, reliability

➤ SRS (Software Requirement Specification)

- A document that describes what the software will do. Basis of agreement between client and developer.
- Good SRS is: correct, complete, unambiguous, consistent, verifiable, modifiable, traceable.

➤ Analysis models

- **DFD (Data Flow Diagram)**: shows how data moves. Level 0 DFD = **context diagram** (whole system as one process).
- DFD symbols: process (circle), data flow (arrow), data store (open rectangle), external entity (rectangle).
- **Data dictionary**: definitions of all data items in the DFD.
- **ER diagram** for data, **state diagram** for behaviour.

4. Software Design

- Converts requirements into a blueprint. Levels: architectural design → high-level design → detailed design.
- Principles: abstraction, modularity, information hiding, refinement, separation of concerns.

➤ Cohesion and coupling

- **Cohesion** = how strongly the parts inside one module belong together. Want high cohesion.
- **Coupling** = how much modules depend on each other. Want low coupling.

Cohesion (worst to best)	Coupling (worst to best)
Coincidental	Content
Logical	Common
Temporal	External
Procedural	Control
Communicational	Stamp
Sequential	Data
Functional (best)	(No coupling is ideal but rare)

REMEMBER Good design = high cohesion + low coupling. Functional cohesion and data coupling are the best.

Design approaches

- **Function-oriented**: top-down decomposition, structure charts.
- **Object-oriented**: classes, objects, UML diagrams.

UML diagrams

Structural	Behavioural
Class diagram	Use case diagram
Object diagram	Sequence diagram
Component diagram	Activity diagram
Deployment diagram	State machine diagram

5. Software Metrics and Estimation

Size metrics

- **LOC (Lines of Code)**: simple but language dependent.
- **Function Points (FP)**: based on functionality. Counts inputs, outputs, inquiries, internal files, external interfaces.

FORMULA $FP = UFP \times CAF$, where $CAF = 0.65 + 0.01 \times \Sigma Fi$ (14 factors, each rated 0 to 5).

EXAMPLE UFP = 200, sum of 14 factors = 35. CAF = 0.65 + 0.35 = 1.00. FP = 200 x 1.00 = 200.

Cyclomatic complexity (McCabe)

FORMULA $V(G) = E - N + 2P$ (P = connected components, usually 1) = number of decision points + 1 = number of regions in a planar flow graph.

EXAMPLE A flow graph has 9 edges and 7 nodes. $V(G) = 9 - 7 + 2 = 4$. So 4 independent paths, and at least 4 test cases for basis path testing.

➤ COCOMO (Basic)

> **FORMULA** Effort E = a x (KLOC)^b person-months. Development time D = c x (E)^d months.

Project type	a	b	c	d
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

- Organic = small team, familiar problem. Embedded = tight hardware / rules constraints.
- Intermediate COCOMO multiplies effort by an **EAF** (from 15 cost drivers).

6. Software Testing

- **Verification**: "Are we building the product right?" (reviews, inspections).
- **Validation**: "Are we building the right product?" (testing against user needs).
- Error (human mistake) → fault / defect (in code) → failure (wrong behaviour at run time).

Black-box testing	White-box testing
Based on requirements, no code view	Based on internal code
Equivalence partitioning, boundary value analysis, decision table	Statement, branch, path coverage, basis path
Done by testers	Done mostly by developers

➤ Levels of testing

1. **Unit testing**: one module. Uses drivers (call the module) and stubs (replace called modules).
2. **Integration testing**: top-down (needs stubs), bottom-up (needs drivers), big bang, sandwich.
3. **System testing**: full system, including performance, load, stress, security.
4. **Acceptance testing**: by the customer. Alpha (at developer site) and beta (at user site).

- **Regression testing**: re-test after changes to make sure old features still work.

EXAMPLE Boundary value analysis for input range 1 to 100: test 0, 1, 2, 99, 100, 101.

- Coverage strength: path coverage > branch coverage > statement coverage.

7. Software Maintenance and Quality

➤ Types of maintenance

Type	Purpose
Corrective	Fix bugs
Adaptive	Adjust to new environment (OS, hardware)
Perfective	Add features, improve performance
Preventive	Make future maintenance easier (refactoring)

- Maintenance usually takes the largest share of total software cost.
- **Re-engineering**, **reverse engineering** (code to design) and **forward engineering** (design to code).

➤ Quality and reliability

- **Software quality**: meets requirements and user expectations. Models: McCall, ISO 9126 / ISO 25010.
- **CMM / CMMI levels**: 1 Initial, 2 Repeatable (Managed), 3 Defined, 4 Quantitatively Managed, 5 Optimizing.
- **Reliability** = probability of failure-free operation for a given time.

FORMULA MTBF = MTTF + MTTR. Availability = MTTF / (MTTF + MTTR).

EXAMPLE MTTF = 95 hours, MTTR = 5 hours. Availability = 95 / 100 = 95%.

8. Project Management and Risk

- Project management activities: planning, scheduling, estimation, staffing, monitoring, risk management.
- Scheduling tools: **Gantt chart** (bars over time), **PERT** (uncertain durations), **CPM** (critical path).

FORMULA PERT expected time = (O + 4M + P) / 6, where O = optimistic, M = most likely, P = pessimistic.

EXAMPLE O = 2, M = 4, P = 12 days. Expected = (2 + 16 + 12) / 6 = 30 / 6 = 5 days.

- **Critical path** = longest path in the activity network. Any delay on it delays the whole project.
- **Risk management**: identify → analyse → plan → monitor. Risk exposure = probability x loss.
- **Configuration management**: version control, change control, baselines.

9. Most Asked Questions (practice)

1. Compare Waterfall, Spiral, Prototyping and Agile models.
2. Functional vs non-functional requirements. What makes a good SRS?
3. Draw level 0 and level 1 DFDs for a library system.
4. Explain types of cohesion and coupling with examples.
5. Calculate effort and time using basic COCOMO.
6. Compute function points for given counts and adjustment factors.
7. Find cyclomatic complexity of a flow graph and list independent paths.
8. Black-box vs white-box testing. Explain boundary value analysis.
9. Explain unit, integration, system and acceptance testing.
10. Types of software maintenance. Explain CMM levels.

TIP University exams ask models, cohesion/coupling, COCOMO and testing levels. GATE (when asked) focuses on cyclomatic complexity, COCOMO and testing concepts.