UnCodemy

Software Engineering Important Questions PDF

The most important Software Engineering questions are SDLC models (waterfall, spiral, prototype, incremental, agile), requirement engineering and the SRS document, DFDs, ER and UML diagrams, coupling and cohesion, COCOMO and function point estimation, black box and white box testing, cyclomatic complexity, the levels of testing, software maintenance, re-engineering, risk management and quality standards such as ISO 9001 and CMM.

Key takeaways

- SDLC model comparison, COCOMO and cyclomatic complexity are the most common long questions.
- Draw DFDs, use case diagrams and the spiral model neatly. Diagrams carry many marks.
- Learn the COCOMO constants table and the function point weights.
- These are practice questions, not past papers; check your own syllabus and official previous papers.

About these important questions

This page lists the most important Software Engineering questions for B.Tech CSE and IT exams, arranged unit by unit. The units follow the standard syllabus that most Indian technical universities share, so the order may differ slightly from yours.

Please note: these are practice questions based on common syllabus topics. They are not verified past papers, and we do not claim that any question will appear in your exam. Check your own university syllabus first, then solve the official previous year papers. AKTU students can download them from the AKTU old question paper archive.

Unit 1: Introduction and Software Process Models

Covers software characteristics, the software crisis and SDLC models.

Short questions (2 marks)

- Define software engineering.
- What was the software crisis?
- List the phases of the SDLC.

Long questions

1. Explain the waterfall model, its phases, advantages and limitations.
2. Explain the spiral model with a diagram and its focus on risk.
3. Compare the prototype, incremental, evolutionary and RAD models.
4. Explain the agile approach, Scrum roles and ceremonies, and extreme programming.
5. Which model would you choose for a project with unclear requirements, and why?
6. Explain the V-model and how each development phase maps to a testing phase.

Unit 2: Software Requirement Specification

Covers requirement engineering, analysis tools and the SRS.

Short questions (2 marks)

- Differentiate between functional and non-functional requirements.
- What is a feasibility study?
- List four characteristics of a good SRS.

Long questions

1. Explain the requirement engineering process: elicitation, analysis, documentation, validation and management.
2. Draw level 0 and level 1 DFDs for a library, banking or hospital system.
3. Explain the structure of an SRS document following IEEE 830.
4. Draw an ER diagram and a data dictionary for a given system.
5. Explain decision tables and decision trees with an example.
6. Explain the SEI CMM levels and ISO 9001.

Unit 3: Software Design

Covers design concepts, modularity, coupling, cohesion and UML.

Short questions (2 marks)

- Define coupling and cohesion.
- What is modularity?
- Differentiate between top-down and bottom-up design.

Long questions

1. Explain the types of coupling and cohesion, from best to worst.
2. Explain function-oriented design and object-oriented design.
3. Draw a use case diagram, class diagram and sequence diagram for an ATM or online shopping system.
4. Explain software measurement: size metrics (LOC, function points) and Halstead's metrics.
5. Calculate function points for given counts of inputs, outputs, inquiries, files and interfaces with a given complexity adjustment.

Unit 4: Software Testing

Covers testing levels, techniques and cyclomatic complexity.

Short questions (2 marks)

- Differentiate between verification and validation.
- Define alpha and beta testing.
- What is regression testing?

Long questions

1. Explain unit, integration, system and acceptance testing.
2. Explain black box techniques: equivalence partitioning, boundary value analysis and cause-effect graphs.
3. Explain white box testing: statement, branch and path coverage.
4. Draw the control flow graph for a code fragment and find its cyclomatic complexity in three ways.
5. Explain top-down and bottom-up integration testing with stubs and drivers.
6. Explain static testing: code walkthroughs, inspections and reviews.

Unit 5: Maintenance, Estimation and Project Management

Covers maintenance, re-engineering, cost estimation, risk and configuration management.

Short questions (2 marks)

- List the types of software maintenance.
- What is reverse engineering?
- What is software configuration management?

Long questions

1. Explain corrective, adaptive, perfective and preventive maintenance.
2. Explain the basic COCOMO model, and estimate effort and time for a 40 KLOC organic project. *Hint:* effort = 2.4 × (KLOC)^1.05 person-months and time = 2.5 × (effort)^0.38 months.
3. Explain intermediate COCOMO and cost drivers.
4. Explain risk management: identification, analysis, planning and monitoring.
5. Explain re-engineering and the forward and reverse engineering process.
6. Explain project scheduling with Gantt charts, PERT and CPM.

Sample answer outline: Cyclomatic complexity

A common question gives a code fragment and asks for its control flow graph (CFG) and cyclomatic complexity.

1. **Number the statements.** Give each statement or condition a node number. Merge simple sequential statements into one node.
2. **Draw the CFG.** Draw an edge for every possible flow. An if has two outgoing edges, and a loop has a back edge to its condition.

3. **Compute V(G) three ways.** V(G) = E - N + 2, where E is the number of edges and N the number of nodes. V(G) = P + 1, where P is the number of predicate (decision) nodes. V(G) = the number of closed regions + 1.
4. **Check that all three agree.** If they do not, recheck the graph.
5. **List the independent paths.** Write V(G) basis paths, each adding at least one new edge. These become the minimum test cases for basis path testing.

For example, a function with one if-else and one while loop has two decision nodes, so V(G) = 3, and you need at least three test cases.

Marks weightage and how to write answers

A common pattern, used by AKTU and many other universities, has a compulsory short-answer section that covers every unit, followed by long-answer questions with a choice inside each unit. Marks per question change between universities and years, so confirm the scheme on a recent official paper.

- **Short answers (about 2 marks):** a crisp definition, one example or diagram, and nothing more. Three to five lines is enough.
- **Long answers (7 to 10 marks):** start with a definition, add a labelled diagram or table, explain the steps, and finish with an example, advantages or a comparison.
- For model comparison questions, a table with rows for requirements clarity, risk, cost and customer involvement is faster to write and easy to mark.
- For COCOMO, write the formula, the constants for the project type and each calculation step with units.

A 7-day revision plan for Software Engineering

1. **Days 1 to 5:** one unit a day. Read your notes, then write the long answers for that unit from memory and check them against your notes.
2. **Day 6:** draw one DFD, one use case and one class diagram, and redo COCOMO, function points and cyclomatic complexity.
3. **Day 7:** solve one full official previous year paper in three hours, then check which units cost you time.

Keep a single sheet of formulas, definitions and diagrams for the last hour before the exam.

Frequently asked questions

Which Software Engineering topics are most important?

SDLC models, SRS and DFDs, coupling and cohesion, UML diagrams, COCOMO and function points, testing techniques and cyclomatic complexity, and software maintenance are asked most often.

How many diagrams should I draw in a Software Engineering answer?

At least one for every long answer where it fits: a model diagram, a DFD, a UML diagram or a CFG. Diagrams make the answer easier to mark and often carry separate marks.

Is Software Engineering useful in real jobs?

Yes. Agile, requirements, design principles, testing and estimation are part of daily work in developer, tester and analyst roles.

Are these Software Engineering questions from previous year papers?

No. They are practice questions built from topics that appear in most B.Tech Software Engineering syllabi. Use them with your own university's official previous papers, for example the AKTU archive at https://aktu.ac.in/old-question-paper.html.

Is this PDF free?

Yes. Fill in the short form on this page and the PDF downloads right away.

Sources and further reading

- AKTU: old question papers (official archive)
- NPTEL: online courses from IITs (search for Software Engineering)
- The Scrum Guide

About this PDF

These are practice questions for revision, prepared by the Uncodemy trainer team from common B.Tech syllabus topics. They are not past papers, a certificate or a promise of marks. Spot an error? Write to info@uncodemy.com and the team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550