

Software Engineering (315323)

Model Question Paper — Set A · Model Answers

Q.1

a) Define software engineering.

2 M

Software engineering is the application of a **systematic, disciplined, quantifiable approach** to the development, operation and maintenance of software; that is, the application of engineering to software. (*IEEE*)

OR: Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. (*Fritz Bauer*)

(Correct definition 2 M)

b) Attributes of a good software.

2 M

1. **Maintainability:** software should be written so that it can evolve to meet the changing needs of customers.
2. **Dependability (reliability, security, safety):** it should not cause physical or economic damage in case of failure and must be protected from malicious users.
3. **Efficiency:** it should not waste system resources such as memory and processor time; response time should be good.
4. **Usability (acceptability):** it must be understandable, easy to use and compatible with the other systems the users work with.

(Any four, ½ M each)

c) Principles of software deployment.

2 M

1. **Customer expectations must be managed:** Do not promise more than you can deliver; tell the customer clearly what the release will and will not contain.
2. **A complete delivery package should be assembled and tested:** All executable files, data files, support documents and installation scripts are put together and beta-tested with real users.

(Others: support regime established before delivery; instructional materials provided; buggy software fixed first, delivered later.)

(Any two, 1 M each)

d) Abstraction and information hiding.

2 M

i) Abstraction: Abstraction means concentrating on the essential features of a problem at a given level and ignoring irrelevant (lower-level) details. Types: procedural abstraction and data abstraction.

ii) Information hiding: Modules are designed so that the information (data and algorithms) contained in one module is inaccessible to other modules that have no need for it; modules interact only through interfaces.

(Each definition 1 M)

e) Risk and types of software risks.

2 M

Risk: A risk is a potential problem — an uncertain event that may or may not occur, but if it occurs it causes loss (to schedule, cost or quality). Two characteristics: *uncertainty* and *loss*.

- **Project risks:** threaten the project plan — budget, schedule, staff, resources.
- **Technical risks:** threaten quality and timeliness — design, implementation, interface, verification problems.
- **Business risks:** threaten the viability of the software — market risk, strategic risk, sales risk, management risk, budget risk.

(Definition 1 M, any two types 1 M)

f) Uses of Gantt chart.

2 M

1. Shows the project schedule as horizontal bars on a time scale — start date, duration and end date of every task.
2. Shows tasks that run in parallel and overlapping tasks.
3. Used to track project progress against plan (planned vs actual), and to show milestones.
4. Simple to understand; used to communicate the schedule to the team and customer.

(Any two uses, 1 M each)

g) Define SQA; goals of SQA.

2 M

Software Quality Assurance (SQA) is a planned and systematic pattern of actions required to give adequate confidence that the software and its process conform to the established technical requirements and standards. It is an *umbrella activity* applied throughout the software process.

Goals:

1. **Requirements quality** — correctness, completeness and consistency of the requirements model.
2. **Design quality** — every design element is assessed to check that it meets the requirements.

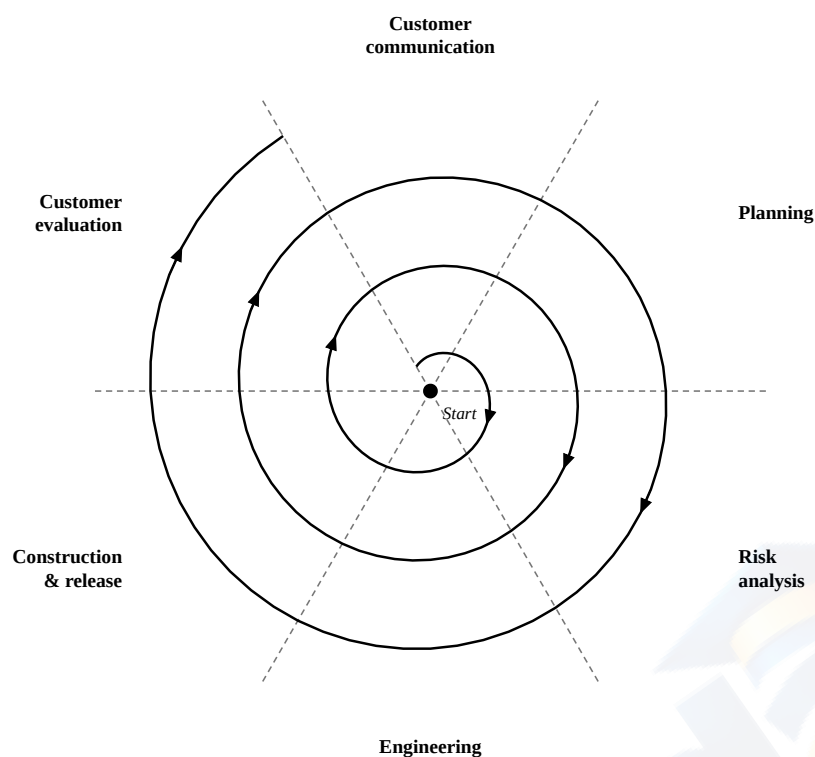
(Definition 1 M, two goals 1 M)

Q.2

a) Spiral model.

4 M

The **spiral model** (Barry Boehm) is an evolutionary process model that combines the iterative nature of prototyping with the controlled, systematic aspects of the waterfall model. Software is developed in a series of evolutionary releases; each loop of the spiral passes through a set of task regions:



Each loop (circuit) of the spiral = one iteration / one more complete version

Fig.: Spiral model (Boehm) with six task regions

1. **Customer communication:** tasks to establish effective communication between developer and customer.
2. **Planning:** define resources, time-lines and other project-related information.
3. **Risk analysis:** assess technical and management risks and plan how to reduce them.
4. **Engineering:** build one or more representations (models, prototypes) of the application.
5. **Construction and release:** construct, test, install the software and provide user support.
6. **Customer evaluation:** obtain customer feedback on the software produced in the engineering and installation stages.

The first circuit may produce a product specification, the next a prototype, and later circuits more complete versions. Cost and schedule are adjusted after each circuit.

(Diagram 2 M, description 2 M)

b) Elements of analysis model.

4 M

The analysis (requirements) model describes what the customer requires, establishes a basis for design and defines requirements that can be validated. Its elements are:

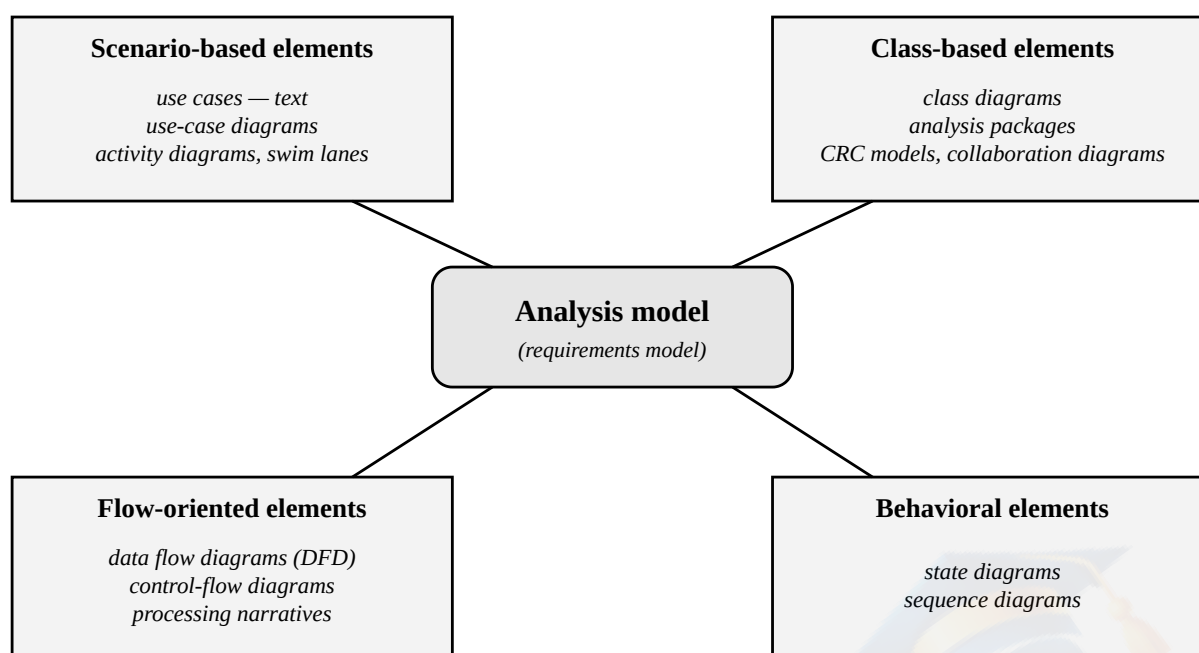


Fig.: Elements of the analysis model

1. **Scenario-based elements:** describe the system from the user's point of view — use cases (text), use-case diagrams, activity and swim-lane diagrams. Usually the first part of the model to be developed.
2. **Class-based elements:** each usage scenario implies objects; these are grouped into classes with attributes and operations — class diagrams, analysis packages, CRC models, collaboration diagrams.
3. **Behavioral elements:** show how the system behaves in response to external events — state diagrams and sequence diagrams.
4. **Flow-oriented elements:** show how data is transformed as it flows through the system — data flow diagrams, control-flow diagrams, processing narratives.

(Diagram 1 M, four elements 3 M)

c) Core principles of software engineering.

4 M

1. **The reason it all exists:** A software system exists for one reason — to provide value to its users. Every decision should be checked against this.
2. **KISS (Keep It Simple, Stupid!):** Design should be as simple as possible, but not simpler; simple designs are easier to understand and maintain.
3. **Maintain the vision:** A clear vision (architecture) is essential; without it the system becomes a patchwork of incompatible designs.
4. **What you produce, others will consume:** Always specify, design and code so that others (maintainers, testers, users) can understand your work.

(Also: be open to the future; plan ahead for reuse; think!)

(Any four principles, 1 M each)

d) Function point computation.

4 M

FP = Count total $\times$ $[0.65 + 0.01 \times \Sigma F_i]$. All parameters are of average complexity, so average weights are used (EI 4, EO 5, EQ 4, ILF 10, EIF 7).

Information domain value	Count	Weight (average)	Count $\times$ weight
External inputs (EI)	24	4	96
External outputs (EO)	16	5	80
External inquiries (EQ)	22	4	88
Internal logical files (ILF)	4	10	40
External interface files (EIF)	2	7	14
Count total (UFP)			318

$$\Sigma F_i = 42$$

$$\text{CAF (value adjustment)} = 0.65 + 0.01 \times 42 = \mathbf{1.07}$$

$$\text{FP} = \text{UFP} \times \text{CAF} = 318 \times 1.07 = \mathbf{340.26}$$

Function points $\approx$ 340

(Count table 2 M, CAF 1 M, FP 1 M)

Q.3

a) Principles of communication practice.

4 M

- 1. Listen:** Concentrate on the speaker's words; ask for clarification when needed and do not interrupt.
- 2. Prepare before you communicate:** Understand the problem and business domain before meeting; prepare an agenda.
- 3. Someone should facilitate the activity:** A leader (facilitator) keeps the meeting productive and resolves conflicts.
- 4. Face-to-face communication is best:** It works best when supported by some representation of the information, e.g. a drawing or document.

(Any four principles, 1 M each)

b) Use case diagram — Online Shopping System.

4 M

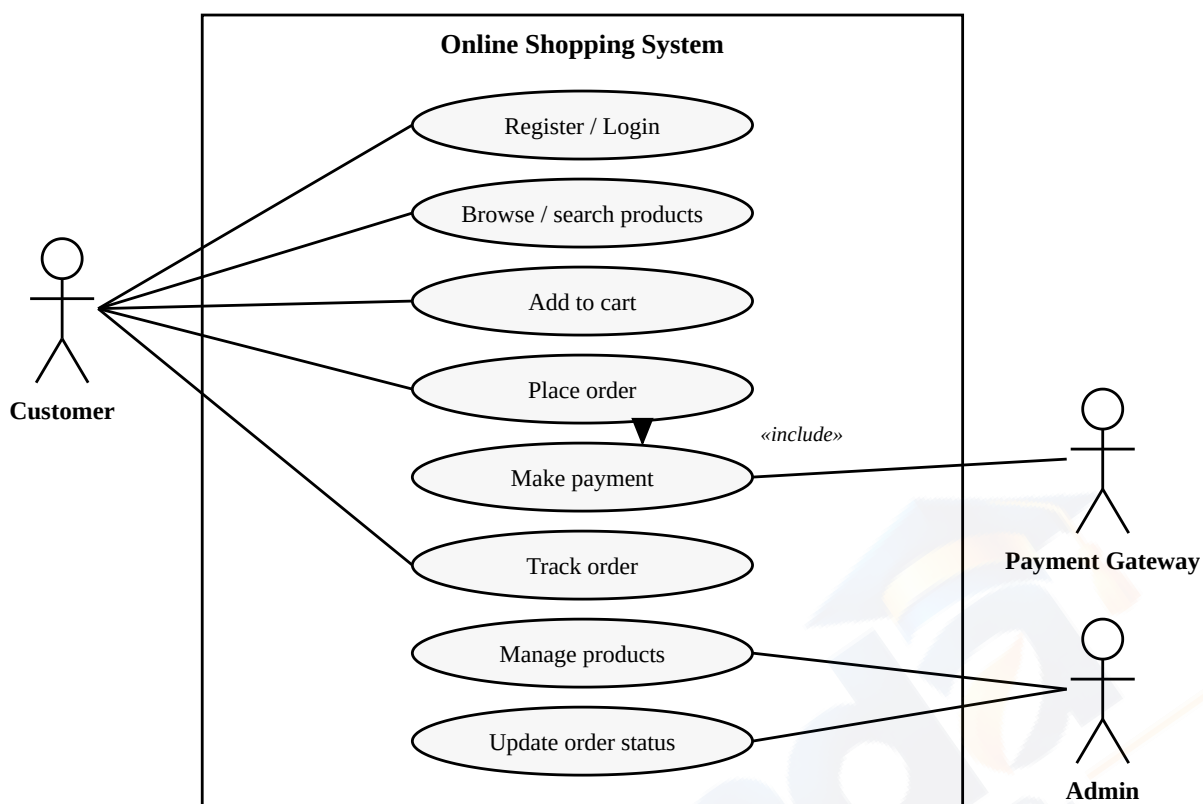


Fig.: Use case diagram — Online Shopping System

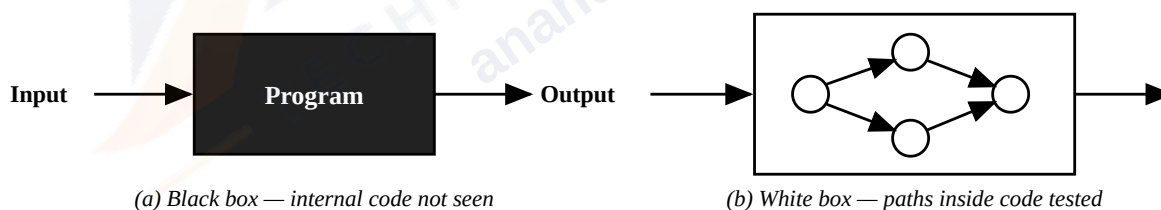
Actors: Customer, Admin (primary), Payment Gateway (secondary, external system).

Use cases: Register/Login, Browse/search products, Add to cart, Place order (includes Make payment), Track order, Manage products, Update order status. The rectangle is the system boundary.

(Actors 1 M, use cases 2 M, relationships/boundary 1 M)

c) Black box vs white box testing.

4 M



Black box testing	White box testing
Tests the functionality of software without looking at the internal code (behavioural testing).	Tests the internal logic, code structure and paths of the program (glass box / structural testing).
Knowledge of programming/implementation is not required.	Knowledge of programming and implementation is required.
Test cases are derived from the requirements/SRS.	Test cases are derived from the source code / design.
Done mainly by testers; applied at system and acceptance level.	Done mainly by developers; applied at unit and integration level.
Techniques: equivalence partitioning, boundary value analysis, cause-effect graphing.	Techniques: basis path testing, statement/branch coverage, loop testing, condition testing.
Finds missing/incorrect functions, interface and performance errors.	Finds logical errors, unreachable code, wrong loops and conditions.

(Any four points, 1 M each)

d) Decision table — shop discount.

4 M

A **decision table** has four parts: condition stub (top-left), condition entries (top-right), action stub (bottom-left) and action entries (bottom-right). With 2 conditions there are $2^2 = 4$ rules.

Conditions / Actions	R1	R2	R3	R4
C1: Customer is a member?	Y	Y	N	N
C2: Bill amount > ₹ 5000?	Y	N	Y	N
A1: 15 % discount	X			
A2: 10 % discount		X		
A3: 5 % discount			X	
A4: No discount				X

Y = Yes, N = No, X = action taken.

(Conditions 1 M, actions 1 M, correct rules 2 M)

Q.4

a) Scrum.

4 M

Scrum is an agile process model (Schwaber and Beedle) in which work is done in short, time-boxed iterations called **sprints**. Its principles: small working teams, flexibility to change, frequent software increments, and testing/documentation done as the product is built.

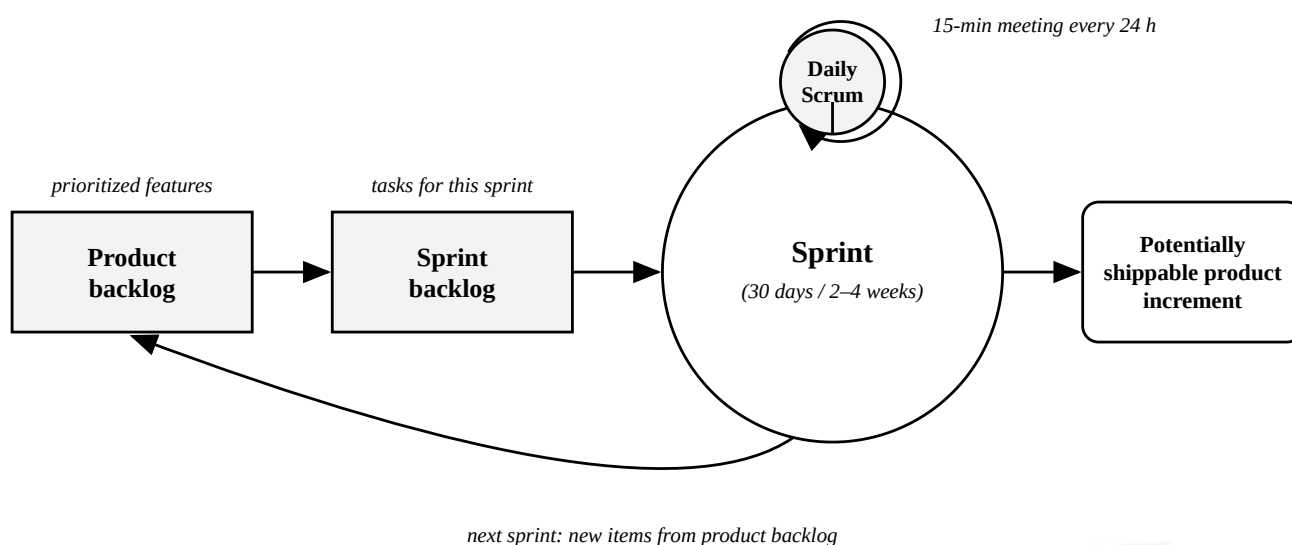


Fig.: Scrum process flow

- **Product backlog:** a prioritized list of features/requirements that provide business value; items can be added at any time by the product owner.
- **Sprint:** work units required to achieve a requirement in the backlog, fitted into a fixed time-box (typically 2–4 weeks / 30 days); no changes are made during a sprint.
- **Daily Scrum meeting:** a short (about 15 minute) daily meeting led by the Scrum master where each member answers: What did you do since the last meeting? What obstacles are you facing? What will you do by the next meeting?
- **Demo / increment:** at the end of each sprint the software increment is demonstrated to the customer for evaluation.
- **Roles:** Product owner, Scrum master, development team.

(Diagram 2 M, explanation 2 M)

b) Management spectrum.

4 M

Effective software project management focuses on the **four P's**: People, Product, Process and Project.

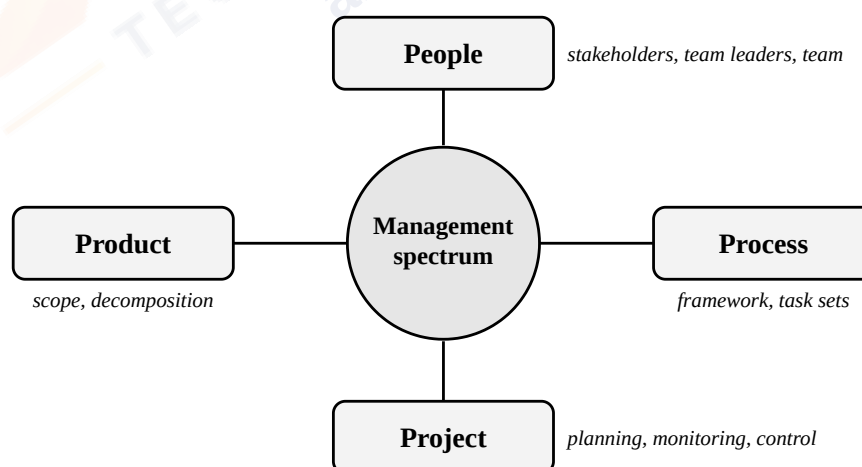


Fig.: The 4 P's of the management spectrum

- 1. People:** The most important element. Stakeholders are senior managers, project (technical) managers, practitioners, customers and end users. Team leaders must motivate, organise and innovate; the team structure (democratic, controlled, agile team) and coordination/communication issues are decided.
- 2. Product:** The software to be built. Before planning, the *software scope* (context, information objectives, function and performance) is established and the problem is *decomposed* into smaller functions so that estimation is possible.
- 3. Process:** The software process provides the framework (communication, planning, modeling, construction, deployment) from which a comprehensive plan is made. A suitable process model and task set is chosen, and umbrella activities (SQA, configuration management, measurement) are added.
- 4. Project:** A planned and controlled software project is the only known way to manage complexity. The project manager plans, monitors and controls work, avoids the common signs of failure and uses the W5HH principle (Why, What, When, Who, Where, How, How much).

(Each P, 1 M)

c) Principles of project scheduling.

4 M

- 1. Compartmentalization:** The project is divided into a number of manageable activities and tasks (product and process are decomposed).
- 2. Interdependency:** The interdependency of each activity/task is determined — some tasks must occur in sequence, others can run in parallel.
- 3. Time allocation:** Each task is allocated a number of work units (person-days) with a start date and a completion date.
- 4. Effort validation:** The project manager checks that no more than the allocated number of people are scheduled at any time.

(Also: defined responsibilities, defined outcomes, defined milestones.)

(Any four, 1 M each)

d) CMMI maturity levels.

4 M

CMMI (Capability Maturity Model Integration), developed by the SEI (Carnegie Mellon University), is a process meta-model that defines the characteristics an organisation's software process must have to be mature. In the **staged** representation it has five maturity levels:

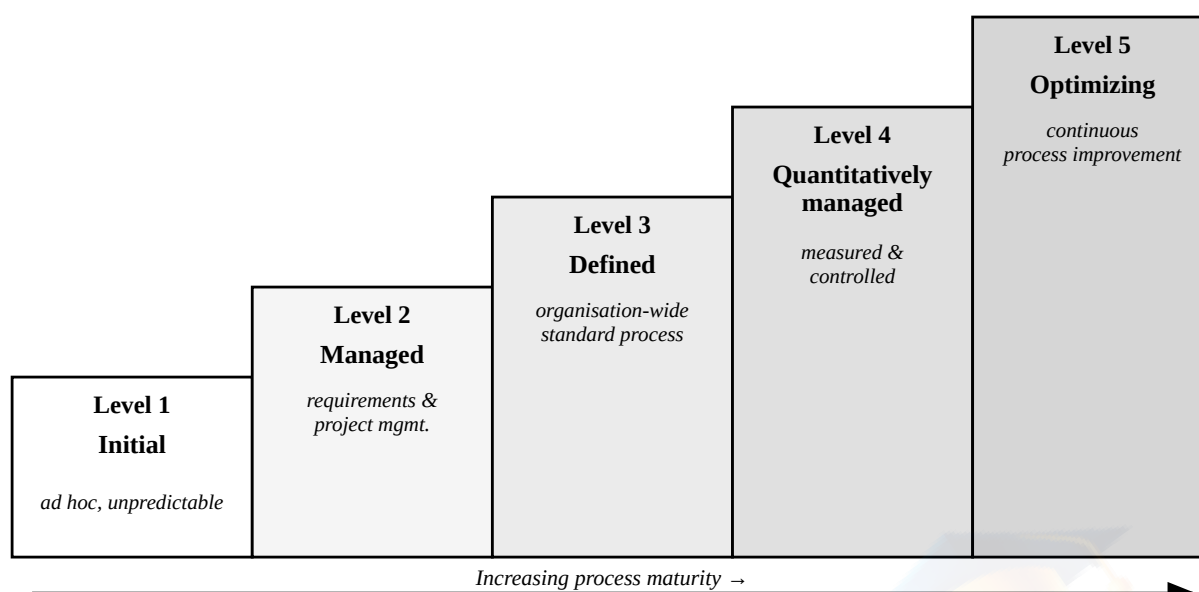


Fig.: CMMI maturity levels (staged representation)

- Level 1 — Initial:** Processes are ad hoc and chaotic; success depends on individual heroics. Projects often exceed budget and schedule.
- Level 2 — Managed:** Basic project management is established: requirements are managed, projects are planned, monitored and controlled; work products are under configuration management.
- Level 3 — Defined:** A standard, documented process is defined for the whole organisation and tailored for each project; risk management and verification/validation are practised.
- Level 4 — Quantitatively managed:** Detailed measures of process and product quality are collected; processes are controlled using statistical and quantitative techniques.
- Level 5 — Optimizing:** Focus on continuous process improvement through innovative ideas, technology and causal analysis of defects.

(Diagram 2 M, levels 2 M)

e) Requirement elicitation techniques.

4 M

Requirement elicitation means drawing out the requirements from customers, users and other stakeholders. Common techniques:

- Interviews:** one-to-one or group discussion with stakeholders using open (free) or closed (pre-defined) questions.
- Collaborative requirements gathering (FAST):** a meeting of developers and customers with a facilitator and an agenda; participants list objects, services, constraints and performance criteria, then a combined list and mini-specifications are made.
- Quality Function Deployment (QFD):** translates customer needs into technical requirements; identifies normal, expected and exciting requirements.
- Usage scenarios (use cases):** scenarios describing how end users will use the system are developed to understand functions and features.
- Questionnaires / surveys, observation, brainstorming and prototyping** are also used.

Elicitation work products: statement of need and feasibility, bounded scope, list of stakeholders, description of technical environment, list of requirements and constraints, usage scenarios, prototypes.

(Any four techniques, 1 M each)

Q.5

a) Waterfall model.

6 M

The **waterfall model** (classic life cycle / linear sequential model) suggests a systematic, sequential approach to software development. Each phase starts only after the previous phase is completed, like water flowing down steps.

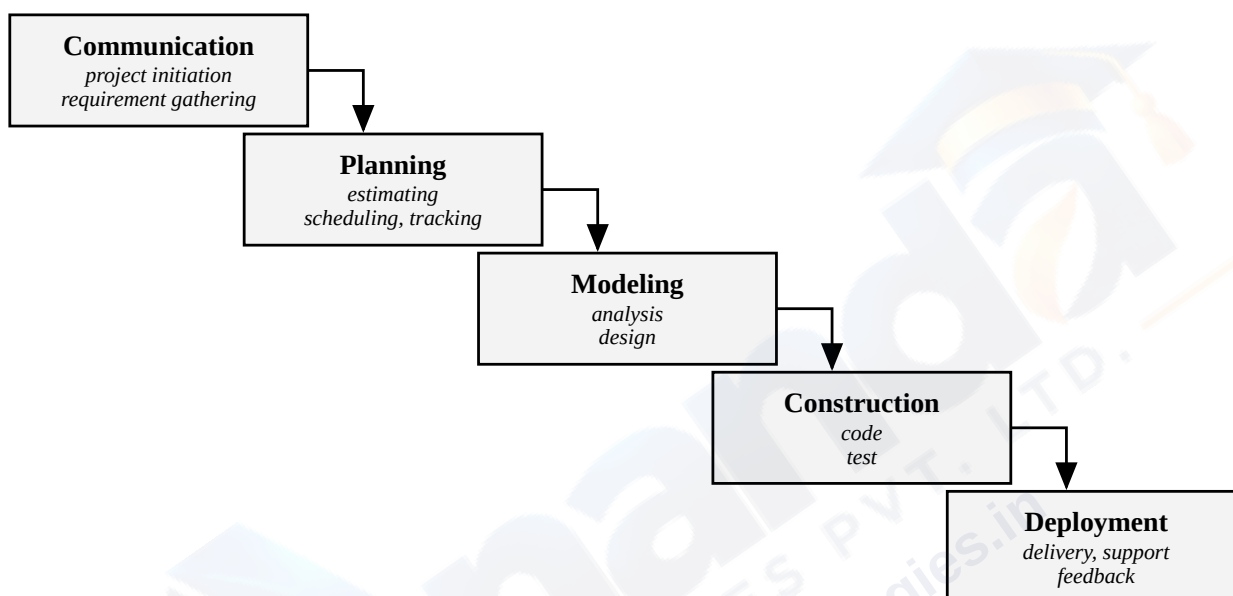


Fig.: Waterfall model (classic life cycle)

1. **Communication:** project initiation and requirement gathering from the customer; requirements are documented (SRS).
2. **Planning:** estimating cost and effort, scheduling and tracking the project.
3. **Modeling:** analysis and design — architecture, data structures, interfaces and algorithms are designed.
4. **Construction:** code generation and testing to uncover errors.
5. **Deployment:** delivery of the software to the customer, support and feedback.

Advantages (merits):

- Simple and easy to understand and use; phases are distinct.
- Each phase has specific deliverables and a review — easy to manage.
- Works well for small projects where requirements are well understood and stable.
- Good documentation is produced.

Disadvantages (demerits):

- Real projects rarely follow a sequential flow; changes cause confusion.

- It is difficult for the customer to state all requirements explicitly at the beginning.
- A working version is available only late in the project; a major error found late can be disastrous.
- Leads to “blocking states” — team members wait for others to finish.

(Diagram 2 M, phases 2 M, merits 1 M, demerits 1 M)

b) Level 0 and Level 1 DFD — Hospital Management System.

6 M

A **context level (Level 0) DFD** shows the whole system as a single process with its external entities. The **Level 1 DFD** expands it into major sub-processes and data stores.

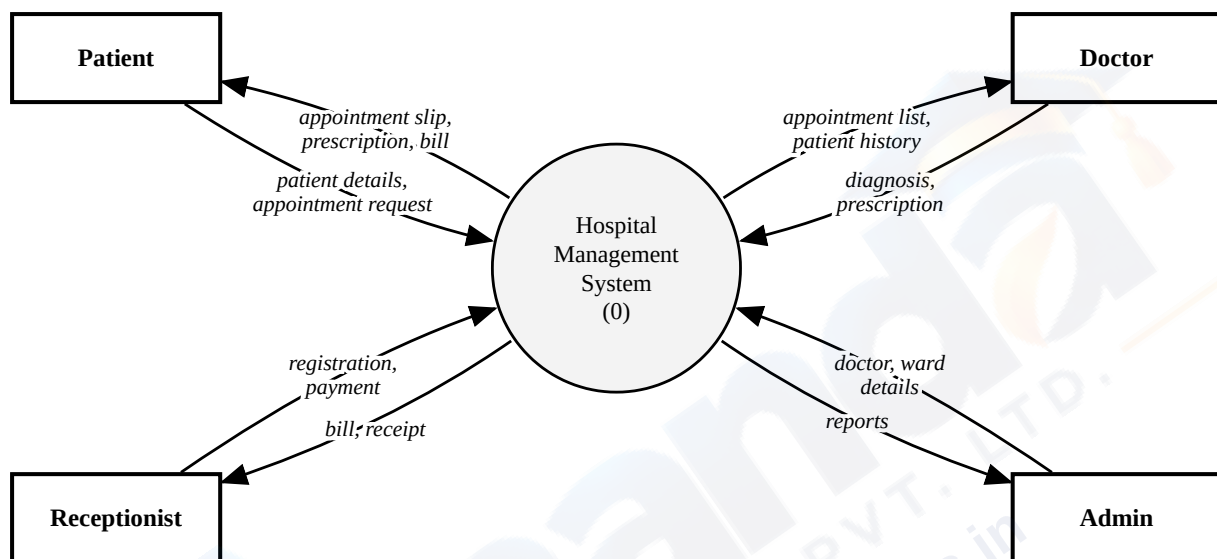


Fig.: Context level (Level 0) DFD — Hospital Management System

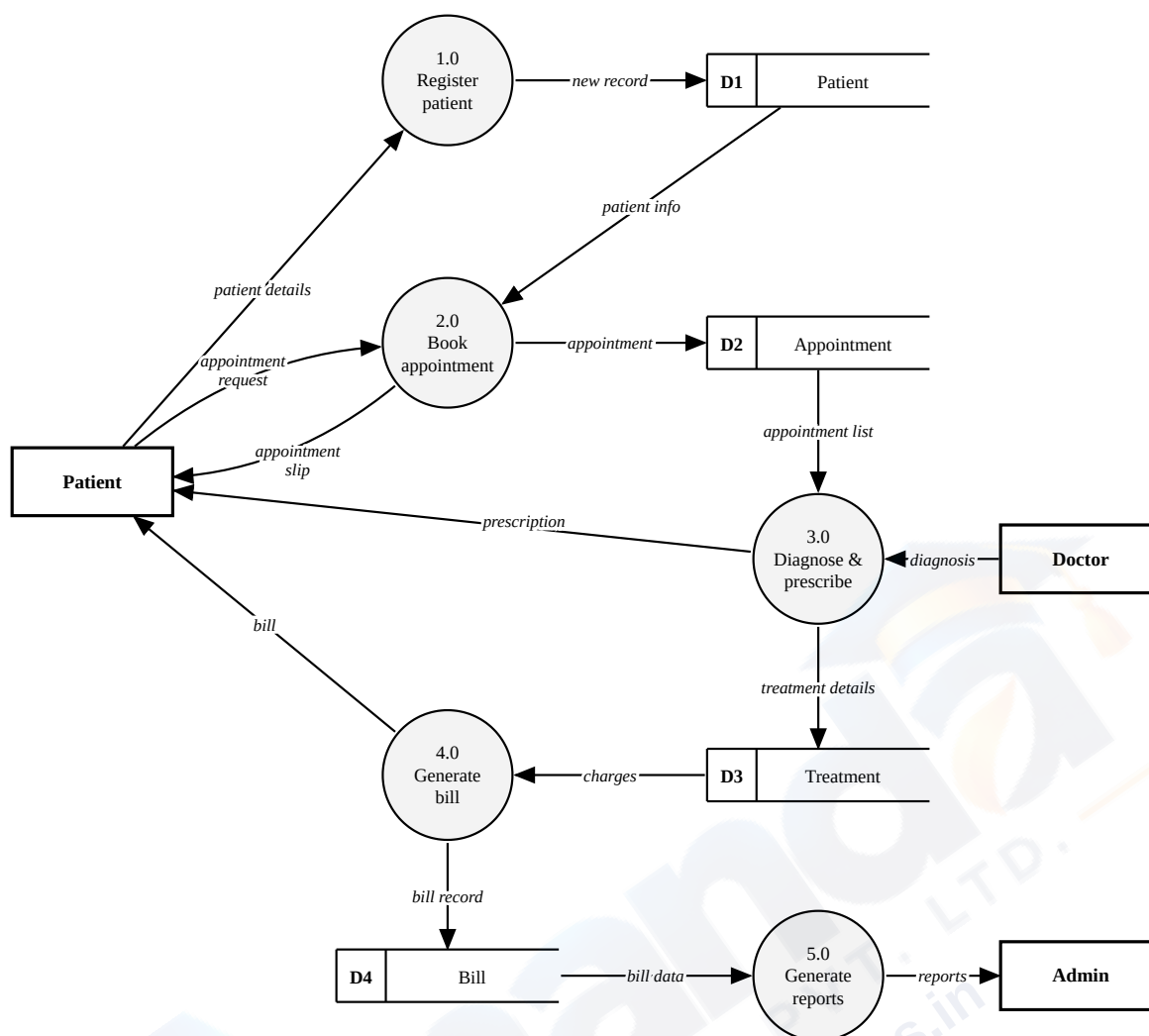


Fig.: Level 1 DFD — Hospital Management System

- Each process is numbered (1.0, 2.0 ... at level 1; 1.1, 1.2 ... at level 2) and named with a verb phrase.
- Data cannot flow directly between two external entities or two data stores; it must pass through a process.
- Data flows are labelled with nouns; the inputs and outputs of the context diagram must balance with the level-1 DFD.

(Level 0 DFD 2 M, Level 1 DFD 4 M)

c) Network diagram, critical path and duration.

6 M

Activities are named A to H in the given order. The network is drawn with events as circles and activities as arrows (activity-on-arrow). The values [E | L] near each event are the earliest and latest event times; thick arrows show the critical path.

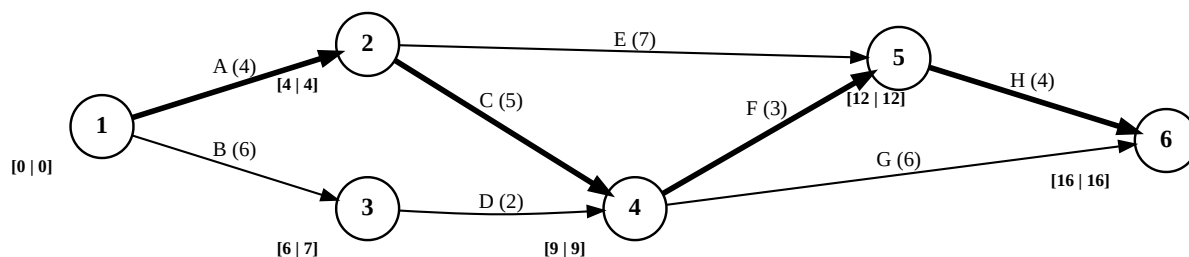


Fig.: Network diagram (critical path shown by thick arrows)

Forward pass (earliest event time, $E_j = \max(E_i + t_{ij})$) and **backward pass** (latest event time, $L_i = \min(L_j - t_{ij})$):

Event	1	2	3	4	5	6
Earliest time E	0	4	6	9	12	16
Latest time L	0	4	7	9	12	16

Activity times ($ES = E_i$, $EF = ES + t$, $LF = L_j$, $LS = LF - t$, Total float = $LS - ES$):

Activity	i-j	Duration	ES	EF	LS	LF	Total float	Free float
A *	1-2	4	0	4	0	4	0	0
B	1-3	6	0	6	1	7	1	0
C *	2-4	5	4	9	4	9	0	0
D	3-4	2	6	8	7	9	1	1
E	2-5	7	4	11	5	12	1	1
F *	4-5	3	9	12	9	12	0	0
G	4-6	6	9	15	10	16	1	1
H *	5-6	4	12	16	12	16	0	0

Activities marked * have zero total float, so they are critical.

Critical path: 1 → 2 → 4 → 5 → 6 (A-C-F-H)

Project duration = 16

(Network 2 M, forward/backward pass 2 M, critical path & duration 2 M)

Q.6

a) SRS — definition, format, characteristics.

6 M

SRS (Software Requirements Specification) is a document that completely describes what the proposed software should do — its functions, performance, interfaces, design constraints and quality attributes — without describing how it will be done. It acts as an agreement between customer and developer and is the basis for design and testing.

Format of SRS (IEEE 830):

- 1. Introduction** — 1.1 Purpose, 1.2 Scope, 1.3 Definitions, acronyms and abbreviations, 1.4 References, 1.5 Overview

2. **Overall description** — 2.1 Product perspective, 2.2 Product functions, 2.3 User characteristics, 2.4 Constraints, 2.5 Assumptions and dependencies
3. **Specific requirements** — 3.1 External interface requirements (user, hardware, software, communication interfaces), 3.2 Functional requirements, 3.3 Performance requirements, 3.4 Design constraints, 3.5 Software system attributes (reliability, availability, security, maintainability, portability), 3.6 Other requirements
4. **Appendices and Index**

Characteristics of a good SRS:

1. **Correct** — every requirement stated is one the software shall meet.
2. **Unambiguous** — every requirement has only one interpretation.
3. **Complete** — includes all significant requirements and responses to all inputs.
4. **Consistent** — no requirements conflict with each other.
5. **Verifiable** — each requirement can be checked by a cost-effective process.
6. **Modifiable and traceable** — easy to change; origin of each requirement is clear.

(Definition 1 M, format 3 M, characteristics 2 M)

b) Basic COCOMO for 250 KLOC.

6 M

Given: Size = 250 KLOC

Organic mode ($a = 2.4$, $b = 1.05$, $c = 2.5$, $d = 0.38$):

$$E = 2.4 \times (250)^{1.05} = 2.4 \times 329.49 = \mathbf{790.77 \text{ person-months}}$$

$$D = 2.5 \times (790.77)^{0.38} = \mathbf{31.56 \text{ months}}$$

$$\text{Average staff} = E / D = 790.77 / 31.56 \approx \mathbf{25.1 \text{ persons}}$$

Semi-detached mode ($a = 3.0$, $b = 1.12$, $c = 2.5$, $d = 0.35$):

$$E = 3.0 \times (250)^{1.12} = 3.0 \times 484.95 = \mathbf{1454.84 \text{ person-months}}$$

$$D = 2.5 \times (1454.84)^{0.35} = \mathbf{31.98 \text{ months}}$$

$$\text{Average staff} = E / D = 1454.84 / 31.98 \approx \mathbf{45.5 \text{ persons}}$$

Embedded mode ($a = 3.6$, $b = 1.2$, $c = 2.5$, $d = 0.32$):

$$E = 3.6 \times (250)^{1.2} = 3.6 \times 754.27 = \mathbf{2715.38 \text{ person-months}}$$

$$D = 2.5 \times (2715.38)^{0.32} = \mathbf{31.39 \text{ months}}$$

$$\text{Average staff} = E / D = 2715.38 / 31.39 \approx \mathbf{86.5 \text{ persons}}$$

Mode	Effort E (PM)	Dev. time D (months)	Staff (persons)
Organic	790.77	31.56	25.1
Semi-detached	1454.84	31.98	45.5
Embedded	2715.38	31.39	86.5

Observation: for the same size, the embedded mode needs the most effort because of tight constraints, while the development time is nearly the same in all modes.

(Formulae 1 M, each mode 1½ M)

c) Risk analysis and management; RMMM sheet.

6 M

Risk analysis and management is a series of steps that help a software team to understand and manage uncertainty. A **proactive** strategy is followed — risks are identified and planned for before problems occur.

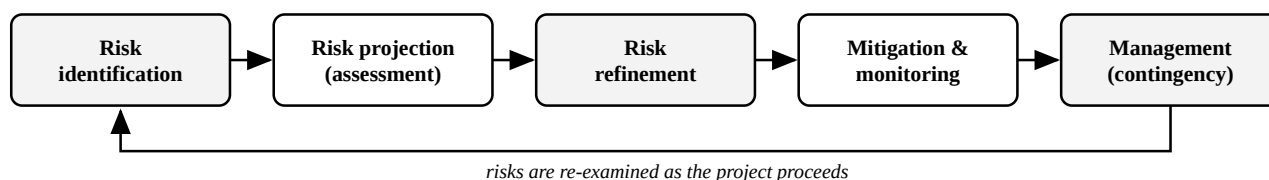


Fig.: Risk analysis and management steps

- 1. Risk identification:** A systematic attempt to list threats to the project plan. A *risk item checklist* is used: product size, business impact, customer characteristics, process definition, development environment, technology to be built, staff size and experience. Risks are generic or product-specific.
- 2. Risk projection (assessment / estimation):** Each risk is rated by (i) its **probability** and (ii) its **impact** (negligible, marginal, critical, catastrophic). A **risk table** is prepared and sorted by probability and impact; a cut-off line separates risks that need management. Risk exposure $RE = P \times C$ (probability $\times$ cost).
- 3. Risk refinement:** A general risk is broken into more detailed risks using the CTC format (Condition–Transition–Consequence) so that it is easier to monitor and manage.
- 4. Risk mitigation, monitoring and management (RMMM):** **Mitigation** — steps to avoid the risk; **monitoring** — tracking factors that show whether the risk is becoming more or less likely; **management** — contingency plan if the risk actually occurs.
- 5. RMMM plan:** All the above work is documented in the RMMM plan (or a *risk information sheet* for each risk) which becomes part of the project plan.

Risk information sheet (example):

Field	Entry
Risk ID	R-01
Description	High staff turnover — experienced developers may leave in the middle of the project.
Probability	70 % (high)
Impact	Critical — schedule slip of 2–3 weeks
Mitigation	Meet staff to find causes of turnover; assign a backup for every critical task; keep documentation up to date.
Monitoring	Watch staff attitude and morale, job-market conditions, number of people leaving.
Management / contingency plan	If a member leaves, the backup person takes over; re-plan the schedule and inform the customer.

(Steps 4 M, RMMM sheet 2 M)

Note: This is a **model question paper prepared by Ananda Technologies** only for practice and exam preparation, based on the pattern of previous MSBTE question papers and the K-scheme syllabus. It is **not an official MSBTE paper**, and it does **not** mean that these questions will come in the exam. Answers follow standard textbook points; marks shown are indicative. Free for students — anandatechnologies.in

