

Software Engineering (315323)

Model Question Paper — Set C · Model Answers

Q.1

a) Characteristics of software.

2 M

1. Software is developed or engineered; it is not manufactured in the classical sense.
2. Software does not “wear out”, but it deteriorates due to changes.
3. Most software is custom built rather than assembled from existing components (though reuse is increasing).
4. Software is intangible, flexible and complex.

(½ M each)

b) IEEE definition of 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)

It covers both the engineering approach and the study of such approaches.

(Definition 2 M)

c) Two principles of deployment practice.

2 M

1. **A support regime must be established before delivery:** Help-desk, contact person, problem-reporting method and support documents must be ready before release.
2. **Buggy software should be fixed first, delivered later:** Under time pressure some teams ship low-quality software; customers forget a late delivery but not a poor-quality product.

(Any two principles with explanation, 1 M each)

d) Modularity and concurrency.

2 M

i) Modularity: Software is divided into separately named and addressable components (modules) that are integrated to satisfy requirements. Example: an online shop divided into Login, Cart, Payment and Delivery modules.

ii) Concurrency: Independent processes are activated simultaneously when the system has multiple processors/threads; the design must handle synchronisation and mutual exclusion. Example: many users booking tickets at the same time.

(Each 1 M)

e) Risk and its characteristics.

2 M

Risk is an uncertain future event or condition that, if it occurs, has a negative effect on the project (cost, schedule, quality).

1. **Uncertainty:** the risk may or may not happen (probability less than 100 %).
2. **Loss:** if the risk becomes a reality, unwanted consequences or losses occur.

(Definition 1 M, two characteristics 1 M)

f) Critical path and total float.

2 M

i) Critical path: the longest path (chain of activities) through the project network from start to end; it decides the minimum project duration. Activities on it have zero float — any delay in them delays the project.

ii) Total float (slack): the amount of time an activity can be delayed without delaying the project completion. $\text{Total float} = \text{LS} - \text{ES} = \text{LF} - \text{EF}$.

(Each 1 M)

g) Activities of the SQA group.

2 M

1. Prepare an SQA plan for the project.
2. Participate in the development of the project's software process description.
3. Review software engineering activities to verify compliance with the defined process.
4. Audit designated software work products to verify compliance.

(Any four, $\frac{1}{2}$ M each)

Q.2

a) Adaptive Software Development.

4 M

ASD (Jim Highsmith) is an agile technique for building complex software and systems. **Focus of ASD:** human collaboration and team self-organization; it treats development as an adaptive (not predictive) activity.

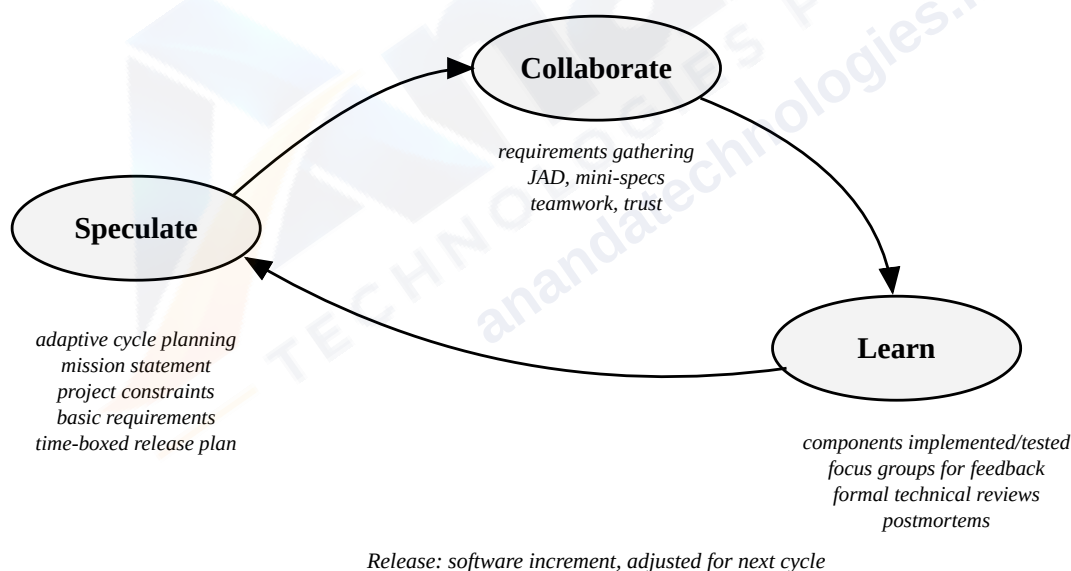


Fig.: Adaptive Software Development (ASD) life cycle

1. **Speculation:** the project is initiated and *adaptive cycle planning* is done using the customer's mission statement, project constraints (delivery dates, user descriptions) and basic requirements to define the set of release cycles.
2. **Collaboration:** motivated people work together in a way that multiplies their talent; it requires trust, the ability to criticise without animosity, assistance, hard work and communication. JAD-

style requirements gathering and mini-specs are used.

3. **Learning:** team members learn through focus groups (customer feedback on increments), formal technical reviews and postmortems; the learning is used to adjust the next cycle.

(Focus 1 M, diagram 1 M, phases 2 M)

b) Structure of the analysis model.

4 M

The analysis model must achieve three objectives: describe what the customer requires, establish a basis for software design, and define requirements that can be validated. Its structure is:

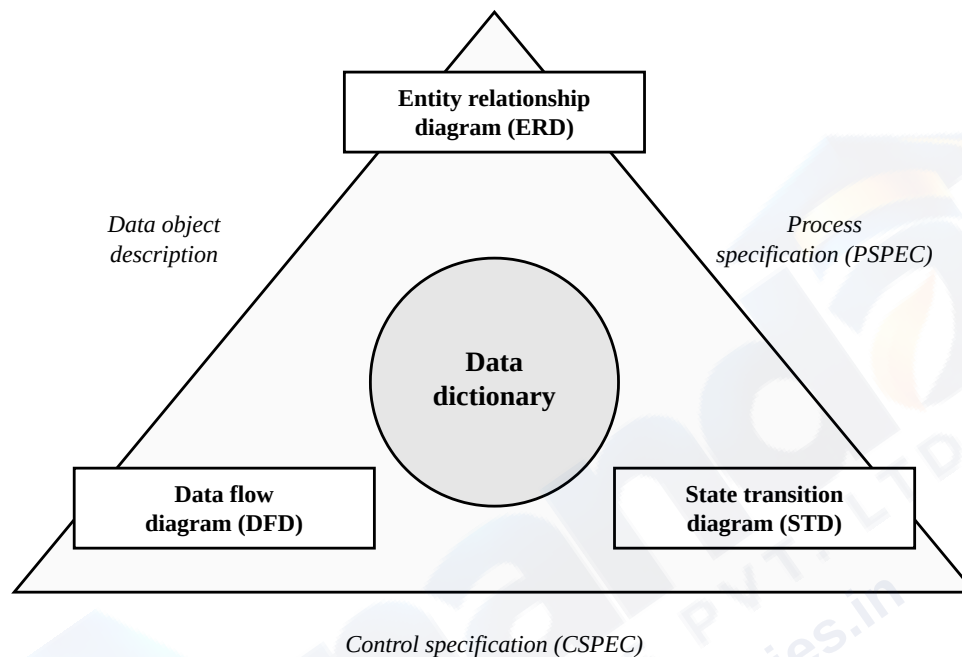


Fig.: Structure of the analysis model (data, functional and behavioral models around the data dictionary)

- **Data dictionary (core):** repository containing descriptions of all data objects used or produced by the software.
- **Entity Relationship Diagram (ERD):** shows relationships between data objects; attributes are described in the *data object description* — the **data model**.
- **Data Flow Diagram (DFD):** shows how data is transformed as it moves through the system; each process is described in a *process specification (PSPEC)* — the **functional model**.
- **State Transition Diagram (STD):** shows system behaviour as a consequence of external events; control is described in the *control specification (CSPEC)* — the **behavioural model**.

(In Pressman's newer edition the same model is organised as scenario-based, class-based, behavioural and flow-oriented elements.)

(Diagram 2 M, explanation 2 M)

c) Gantt chart.

4 M

A **Gantt chart** (timeline chart) shows tasks on the vertical axis and time on the horizontal axis; each task is a bar from its start to its finish. Start of each task = latest finish of its predecessors:

Task	Duration (weeks)	Predecessor	Starts after week	Finish (end of week)
T1 Requirement analysis	2	—	0	2
T2 SRS preparation	1	T1	2	3
T3 System design	2	T2	3	5
T4 Database design	1	T2	3	4
T5 Front-end coding	3	T3	5	8
T6 Back-end coding	3	T3, T4	5	8
T7 Integration & testing	2	T5, T6	8	10
T8 User manual	2	T3	5	7
T9 Deployment	1	T7, T8	10	11

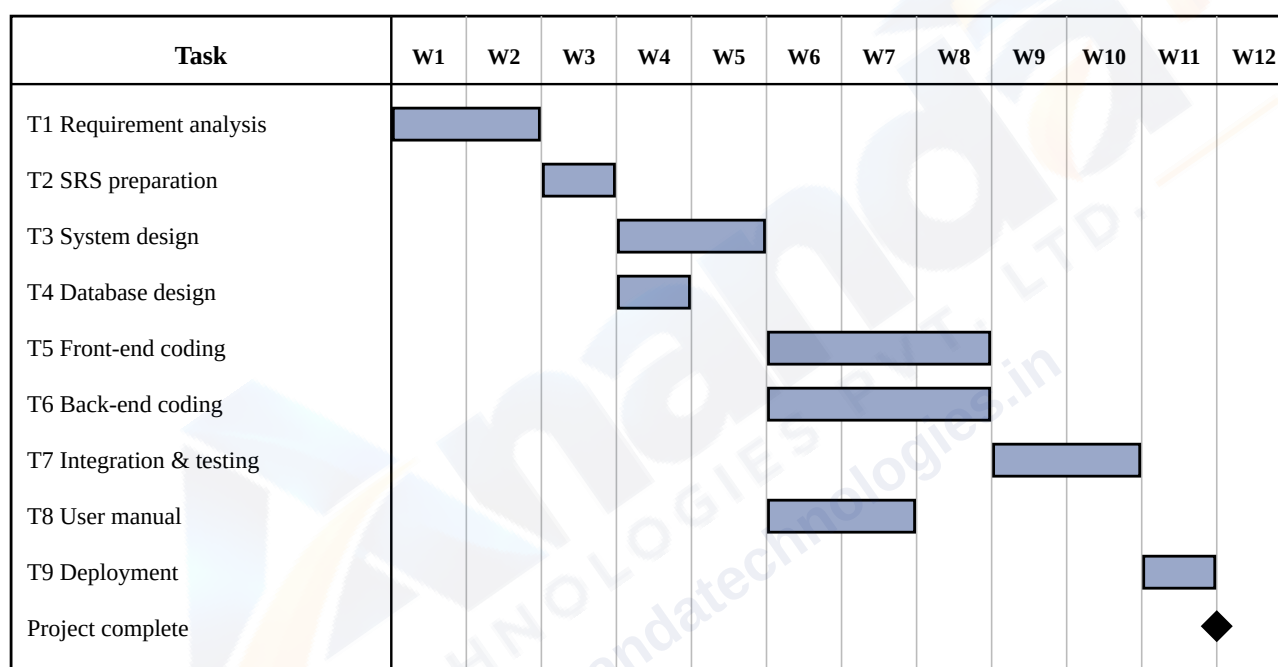


Fig.: Gantt chart (total duration 11 weeks; ♦ = milestone)

The project finishes at the end of week **11**. Tasks T5, T6 and T8 run in parallel after system design.
(Schedule 1 M, chart 3 M)

d) Principles of modelling practice.

4 M

Models help to understand the problem (requirements/analysis model) and to describe the solution (design model). Principles:

- The information domain of the problem must be represented and understood** — data flowing in, out and stored in the system.
- The functions that the software performs must be defined** — functions transform input data to output.
- The behaviour of the software must be represented** — how the system responds to external events.

4. **Models must be partitioned to uncover detail in a layered (hierarchical) fashion** — complex problems are divided into sub-problems.

(Design modelling: design traceable to requirements, interfaces designed with care, components functionally independent and loosely coupled.)

(Any four, 1 M each)

Q.3

a) Principles of construction practice.

4 M

Construction includes **coding** and **testing**. The principles are:

1. **Preparation principles:** Before coding: understand the problem, understand basic design principles, choose a suitable language and environment, and prepare unit tests.
2. **Programming principles:** Use structured programming, choose proper data structures, keep conditional logic simple, write nested loops so they are easily testable, use meaningful variable names and self-documenting code, and use a clear visual layout (indentation).
3. **Validation principles:** After coding: conduct a code walkthrough, perform unit tests and correct errors, and refactor the code.
4. **Testing principles:** All tests should be traceable to customer requirements; tests should be planned long before testing begins; the Pareto principle applies (80 % of errors come from 20 % of components); testing should begin “in the small” and progress “in the large”; exhaustive testing is not possible.

(Coding principles 2 M, testing principles 2 M)

b) Use case diagram — ATM system.

4 M

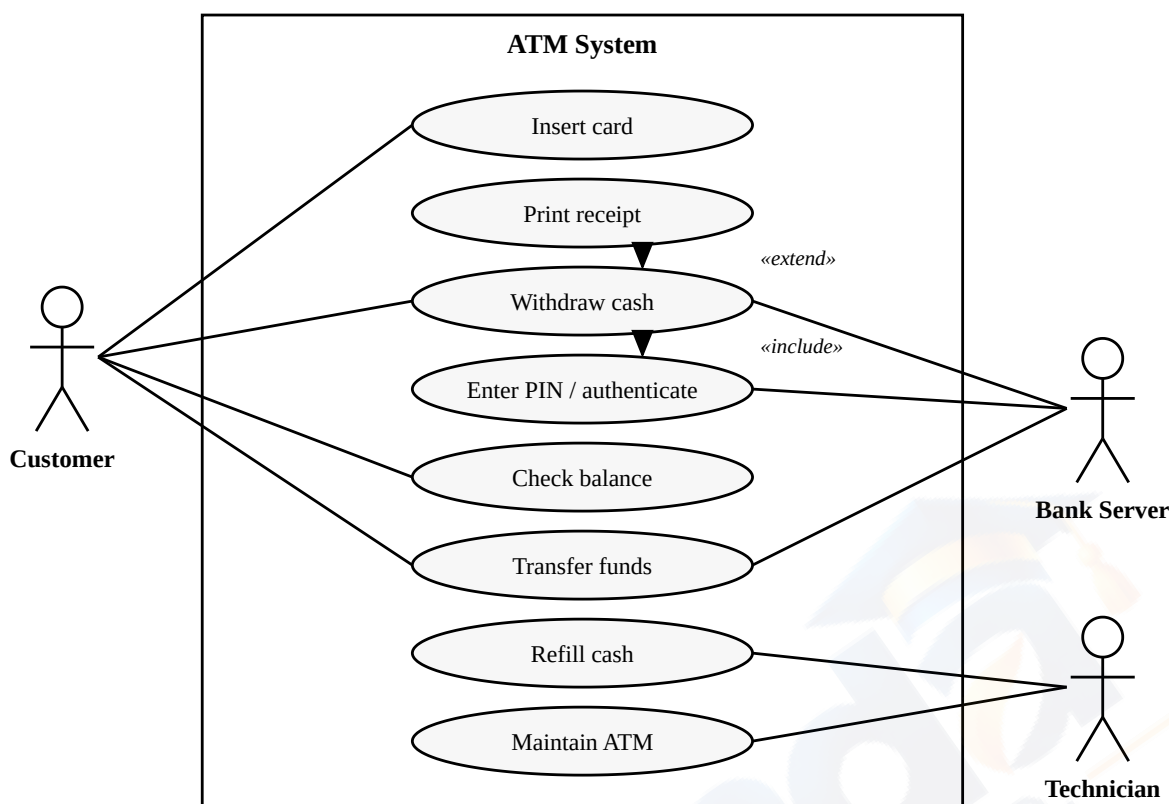


Fig.: Use case diagram — ATM System

Actors: Customer (primary), Bank Server (secondary system), Technician. **Use cases:** Insert card, Enter PIN/authenticate, Withdraw cash, Check balance, Transfer funds, Print receipt (extends withdraw — optional), Refill cash, Maintain ATM. Withdraw cash «include» authentication.

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

c) Levels of testing.

4 M



Fig.: Levels of testing (testing in the small → testing in the large)

- 1. Unit testing:** each module/component is tested separately by the developer, using white-box techniques; drivers and stubs are used. Checks interface, local data structures, boundary conditions, independent paths and error handling.
- 2. Integration testing:** tested modules are combined and tested to find interface errors.
Approaches: top-down, bottom-up, sandwich and regression testing.
- 3. System testing:** the complete integrated system is tested against the SRS — recovery, security, stress, performance and deployment testing.
- 4. Acceptance testing:** performed by the customer/users to decide whether to accept the software — *alpha testing* (at developer's site) and *beta testing* (at customer's site).

(Each level 1 M)

d) Decision table — loan sanction.

4 M

Three conditions give $2^3 = 8$ rules:

Conditions / Actions	R1	R2	R3	R4	R5	R6	R7	R8
C1: Age between 21 and 60?	Y	Y	Y	Y	N	N	N	N
C2: Annual income $\geq$ ₹ 3 lakh?	Y	Y	N	N	Y	Y	N	N
C3: Guarantor available?	Y	N	Y	N	Y	N	Y	N
A1: Approve loan	X	X						
A2: Approve loan with guarantor			X					
A3: Reject loan				X	X	X	X	X

Y = Yes, N = No, X = action. Rules R5–R8 (age not in range) all lead to rejection, so they can be merged into one rule with “–” (don’t care) entries for C2 and C3; similarly R1 and R2 can be merged.

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

Q.4

a) Incremental process model.

4 M

The **incremental model** combines elements of linear and parallel process flows. It applies linear sequences in a staggered fashion as calendar time progresses; each linear sequence produces a deliverable **increment** of the software.

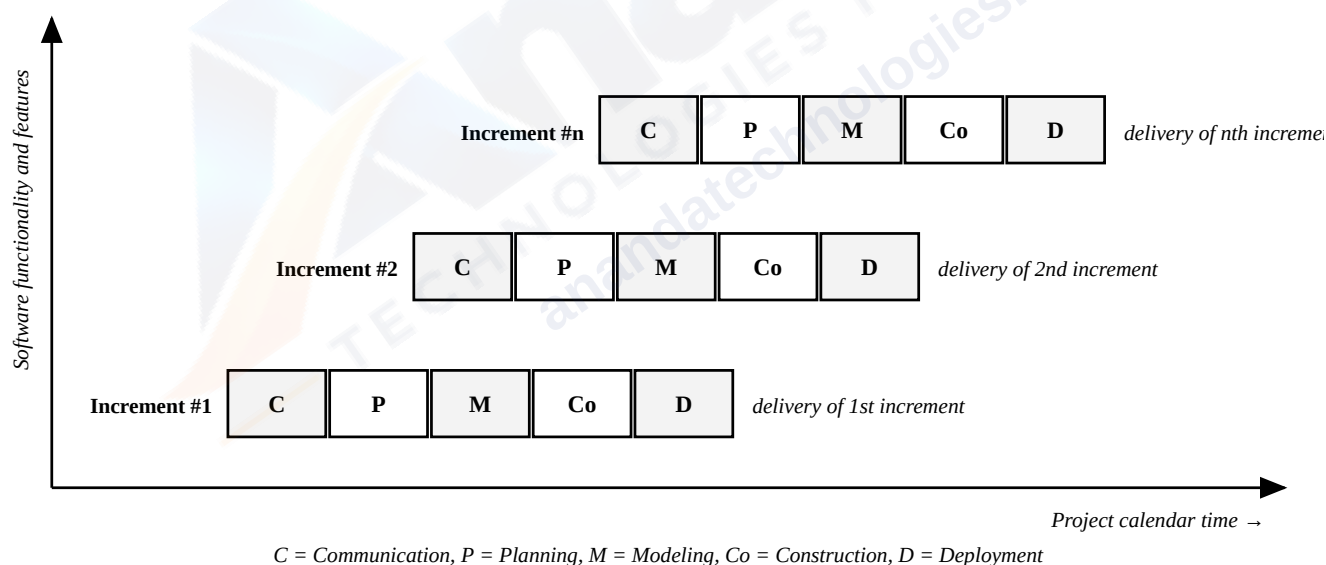


Fig.: Incremental process model

- The first increment is often the **core product** — basic requirements are addressed; supplementary features are delivered later.
- The core product is used and evaluated by the customer; a plan for the next increment is made based on feedback.

- Example: a word-processor — increment 1: basic file management, editing; increment 2: advanced editing; increment 3: spelling and grammar check; increment 4: page layout.
- **Advantages:** working software early; useful when staffing is not available for complete implementation; technical risks can be managed; easier to test smaller increments.

(Diagram 2 M, explanation 2 M)

b) Heuristic, analytical and empirical estimation.

4 M

1. **Empirical estimation techniques:** based on making an educated guess of project parameters using past experience. *Expert judgement* — an expert estimates cost after analysing the problem; *Delphi cost estimation* — a group of experts estimate anonymously under a coordinator, and estimates are refined over several rounds.
2. **Heuristic techniques:** assume that the relationship among project parameters can be modelled by mathematical expressions derived from historical data. *Single-variable models:* Estimated parameter = $c_1 \times e^{d_1}$ (e.g. basic COCOMO: effort from KLOC). *Multivariable models:* depend on several parameters (e.g. intermediate COCOMO with cost drivers).
3. **Analytical techniques:** derive the required results starting from basic assumptions about the problem, with scientific basis. Example: *Halstead's software science* — uses the number of operators and operands to compute program length, volume and effort.

Empirical and heuristic techniques are used for project-level estimates; analytical techniques are useful mainly for estimating software maintenance and program-level metrics.

(Each technique with example, about 1½ M each)

c) Principles of project scheduling.

4 M

Scheduling is the activity of distributing estimated effort across the planned project duration by allocating effort to specific tasks. Basic principles:

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.
5. **Defined responsibilities:** Every scheduled task is assigned to a specific team member.
6. **Defined outcomes:** Every task has a defined outcome (work product), e.g. a design document or module.
7. **Defined milestones:** Every task or group of tasks is associated with a milestone; a milestone is reached when work products are reviewed and approved.

(Any four principles, 1 M each)

d) CMMI with five levels.

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:

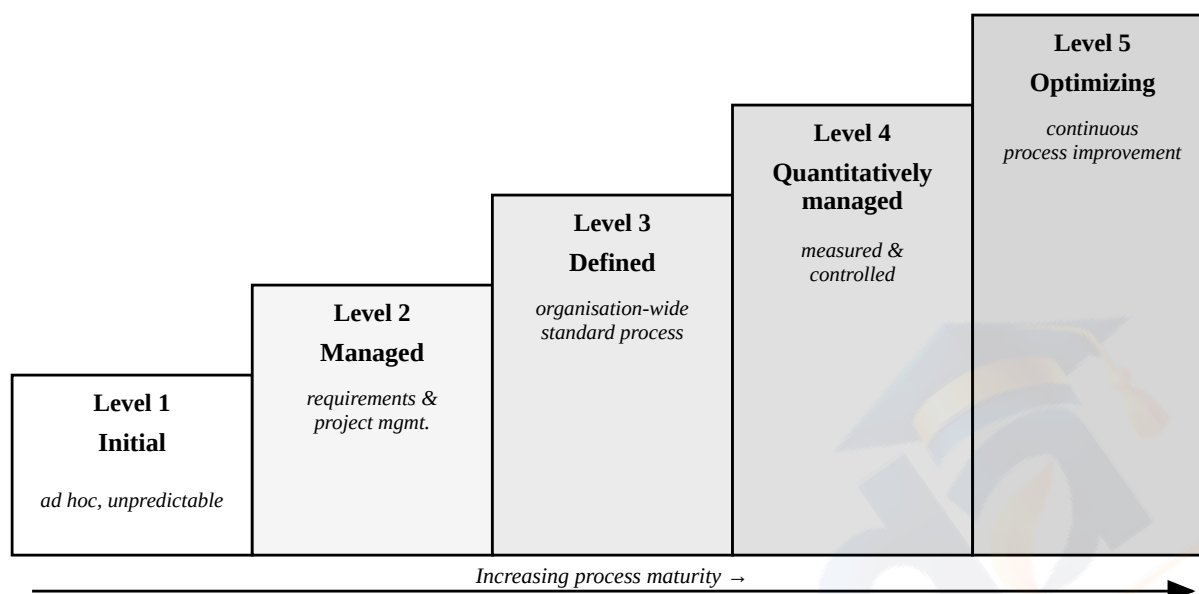


Fig.: CMMI maturity levels (staged representation)

- 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.

CMMI also has a **continuous representation** in which each process area is rated by *capability levels* (0 incomplete to higher levels).

(Diagram 2 M, levels 2 M)

e) Activity diagram — online course registration.

An **activity diagram** shows the flow of control from activity to activity. Notation: filled circle = start, rounded rectangle = activity, diamond = decision/merge with guard conditions [], thick bar = fork/join (parallel activities), bull's-eye = end.

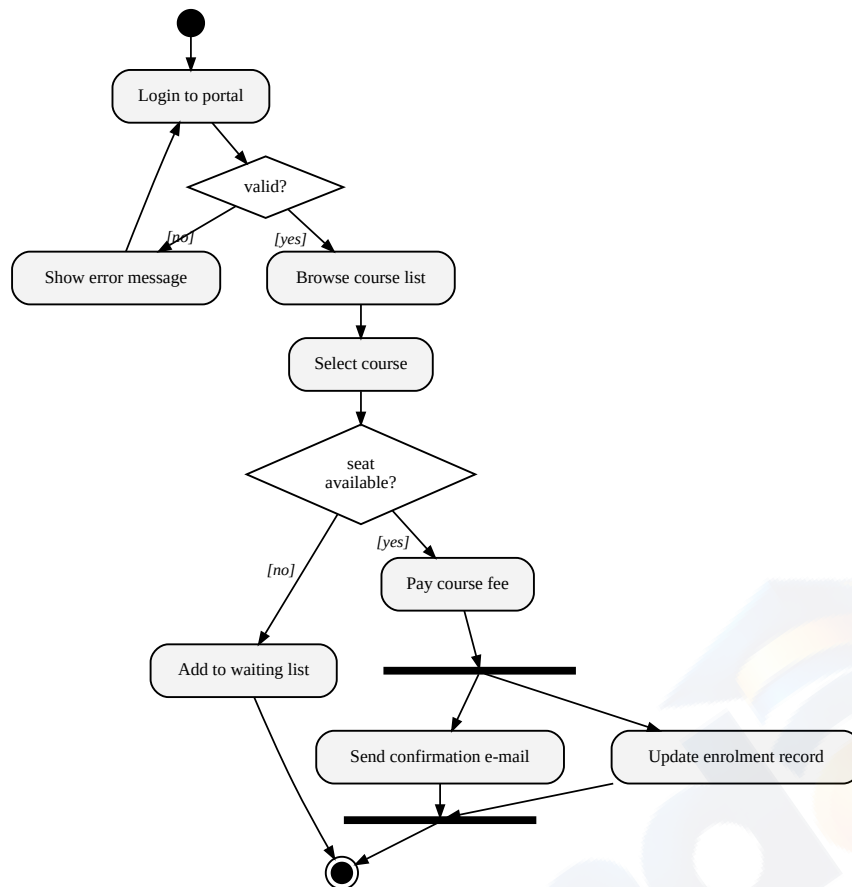


Fig.: Activity diagram — online course registration

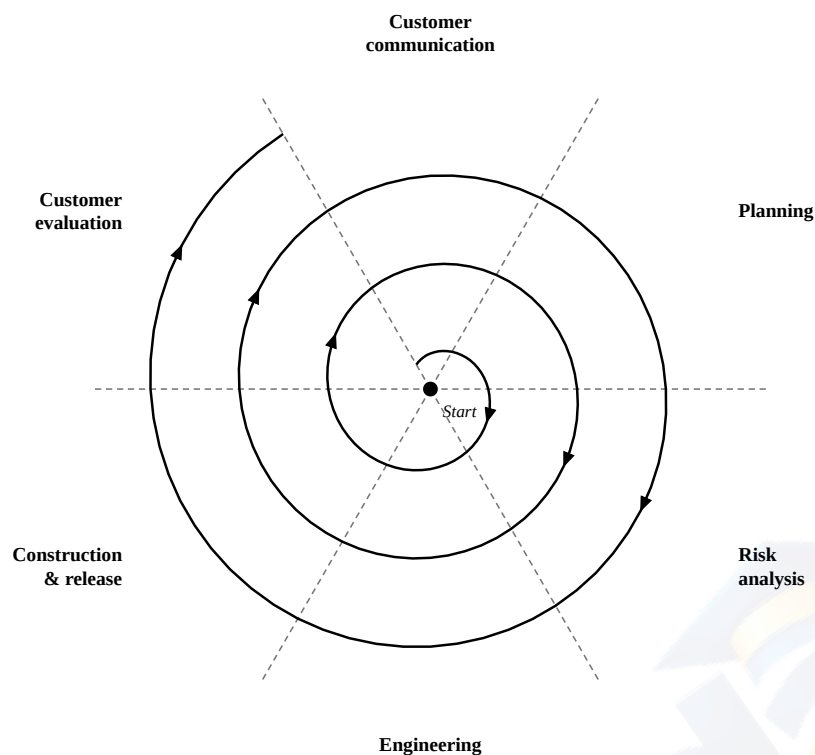
(Correct flow 3 M, notation 1 M)

Q.5

a) **Spiral model with six task regions; advantages and disadvantages.**

6 M

The **spiral model**, proposed by Barry Boehm, is an evolutionary, **risk-driven** process model. It couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. The software is developed in a series of evolutionary releases; the project moves around the spiral in a clockwise direction starting at the centre.



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

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

1. **Customer communication** — establish effective communication between developer and customer.
2. **Planning** — define resources, time-lines and other project information.
3. **Risk analysis** — assess technical and management risks.
4. **Engineering** — build one or more representations (models/prototypes) of the application.
5. **Construction and release** — construct, test, install and provide user support.
6. **Customer evaluation** — obtain customer feedback on the evaluation of the software.

Advantages:

- Risk analysis at every loop reduces project risk; suitable for large and high-risk projects.
- Software is produced early; customer sees and evaluates each release.
- Changes in requirements can be accommodated in later loops.
- Combines the strengths of waterfall (control, documentation) and prototyping (iteration).

Disadvantages:

- Needs considerable risk-assessment expertise; success depends on it.
- Costly for small projects.
- Difficult to convince customers that the evolutionary approach is controllable; the end of the project may not be known early.

(Diagram 2 M, regions 2 M, advantages 1 M, disadvantages 1 M)

b) Level 0 and Level 1 DFD — Online Food Ordering System.

6 M

The **context diagram** shows the system as one process (bubble 0) interacting with external entities. The **Level 1 DFD** decomposes it into major processes, data stores and data flows.

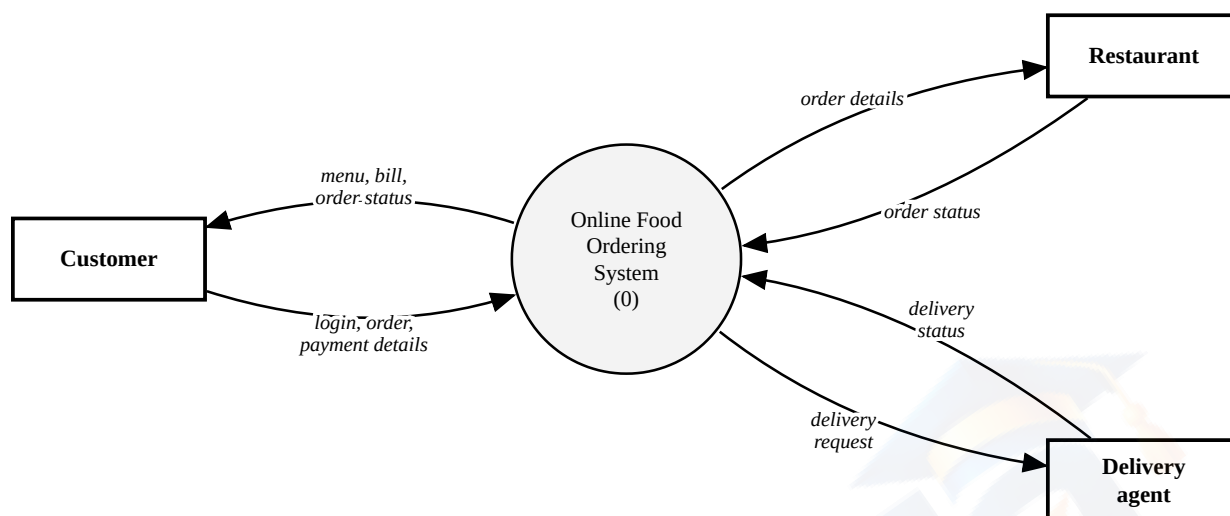


Fig.: Context level (Level 0) DFD — Online Food Ordering System

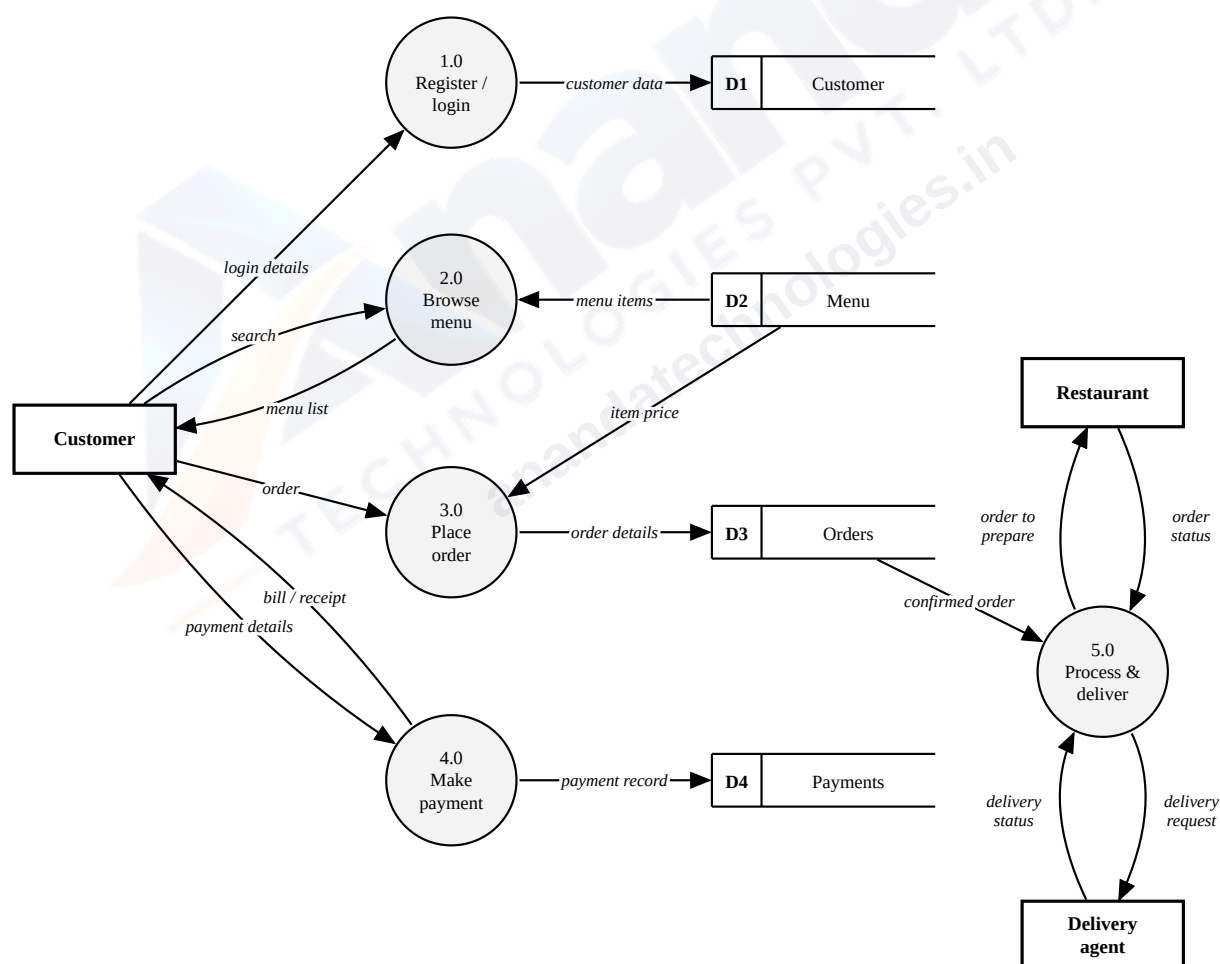
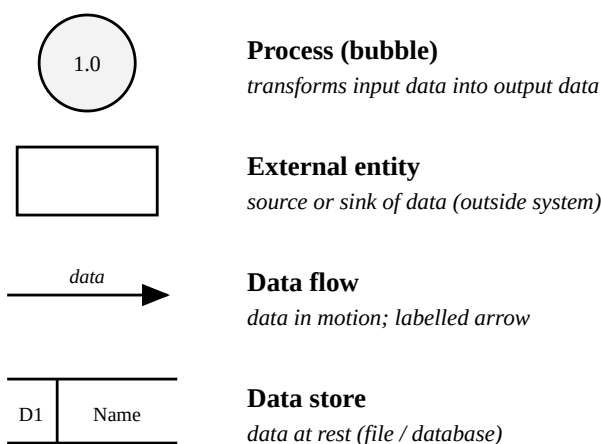


Fig.: Level 1 DFD — Online Food Ordering System



Notation used: DFD symbols (Yourdon–DeMarco symbols)

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

c) Basic COCOMO — 45 KLOC.

6 M

Basic COCOMO: $E = a(KLOC)^b$ PM, $D = c(E)^d$ months. Organic: $a = 2.4$, $b = 1.05$, $c = 2.5$, $d = 0.38$.

Organic mode (45 KLOC):

$$E = 2.4 \times (45)^{1.05} = 2.4 \times 54.43 = \mathbf{130.64 \text{ person-months}}$$

$$D = 2.5 \times (130.64)^{0.38} = \mathbf{15.92 \text{ months}}$$

$$\text{Average staff} = E / D = 130.64 / 15.92 \approx \mathbf{8.2 \text{ persons}}$$

$$\text{Productivity} = KLOC / E = 45000 \text{ LOC} / 130.64 \text{ PM} \approx \mathbf{344 \text{ LOC per person-month}}$$

Same project in other modes:

$$\text{Semi-detached: } E = 3.0 \times (45)^{1.12} = 3.0 \times 71.06 = \mathbf{213.17 \text{ PM}}; D = 2.5 \times (213.17)^{0.35} = 16.33 \text{ months}$$

$$\text{Embedded: } E = 3.6 \times (45)^{1.2} = 3.6 \times 96.35 = \mathbf{346.86 \text{ PM}}; D = 2.5 \times (346.86)^{0.32} = 16.25 \text{ months}$$

Mode	Effort (PM)	Time (months)	Staff	Productivity (LOC/PM)
Organic	130.64	15.92	8.2	344
Semi-detached	213.17	16.33	13.1	211
Embedded	346.86	16.25	21.3	130

A 45 KLOC project fits the organic range (2–50 KLOC); treating it as embedded would almost double the estimated effort.

(Organic E, D 2 M, staff & productivity 2 M, other modes 2 M)

Q.6

a) Phases of Software Quality Assurance.

6 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.

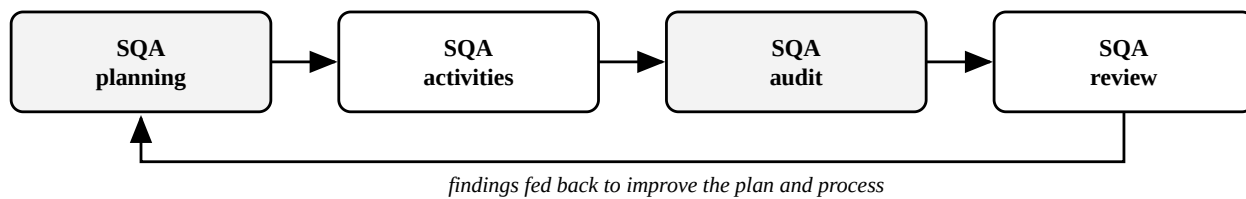


Fig.: Phases of SQA

1. **SQA planning:** an SQA plan is prepared at the start — purpose and scope, work products to be reviewed (SRS, design, code, test plans), standards and practices to be followed, SQA tasks and their schedule, tools and methods, responsibilities, records to be kept, and the defect reporting mechanism.
2. **SQA activities:** the SQA group participates in defining the process; reviews software engineering activities to check compliance with the defined process; applies formal technical reviews, testing and measurement; ensures deviations are documented and handled; manages change (configuration management).
3. **SQA audit:** designated work products are audited — an independent examination to verify that they conform to standards, plans and procedures; non-compliance items are identified, recorded and reported to senior management.
4. **SQA review:** formal technical reviews (walkthroughs, inspections) are conducted at the end of each phase to uncover errors in function, logic or implementation; results are recorded and tracked until closed. Management reviews check the effectiveness of the SQA plan itself.

SQM vs SQA: Software Quality Management is the broader umbrella (quality planning, assurance, control, improvement); SQA is the part that assures the defined process and standards are followed.

(Diagram 1 M, four phases 5 M)

b) CPM and PERT; differences.

6 M

i) Critical Path Method (CPM): developed by DuPont (1957). It is a *deterministic* technique — each activity has one fixed duration. Steps: list activities and dependencies, draw the network, compute earliest times (forward pass) and latest times (backward pass), compute float, and identify the **critical path** (longest path, zero float). It is also used for time–cost trade-off (crashing).

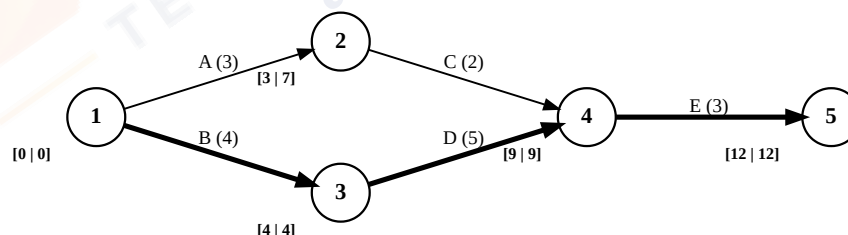


Fig.: Example CPM network — critical path B–D–E, duration 12 days

ii) PERT (Program Evaluation and Review Technique): developed by the US Navy (Polaris project, 1958). It is a *probabilistic* technique used when activity times are uncertain. Three time estimates are used: optimistic t_o , most likely t_m , pessimistic t_p ; expected time $t_e = (t_o + 4t_m + t_p)/6$ and variance $\sigma^2 = ((t_p - t_o)/6)^2$. The critical path is found using t_e , and the probability of completing the project by a given date can be calculated.

CPM	PERT
Deterministic — one time estimate per activity.	Probabilistic — three time estimates per activity.
Activity-oriented.	Event-oriented.
Used for repetitive, well-known projects (construction).	Used for new, research and development projects with uncertain times.
Focuses on time–cost trade-off (crashing).	Focuses on time and probability of meeting deadlines.
No variance / probability computed.	Variance and standard deviation of project duration are computed.

(CPM 2 M, PERT 2 M, differences 2 M)

c) RMMM plan; risk table.

6 M

RMMM (Risk Mitigation, Monitoring and Management) plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan. Each risk may also be documented individually on a *Risk Information Sheet (RIS)*.

- **Risk mitigation (avoidance):** proactive steps taken to avoid the risk or reduce its probability.
- **Risk monitoring:** tracking the project to check whether predicted risks occur, whether mitigation steps are applied, and collecting information for future analysis.
- **Risk management and contingency planning:** actions taken when mitigation has failed and the risk has become a real problem.

Risk table — College ERP project (impact: 1 catastrophic, 2 critical, 3 marginal, 4 negligible; RE = probability × cost):

ID	Risk	Category	Probability	Impact	Cost if occurs (₹)	Risk exposure RE (₹)
R2	Key developer leaves mid-project	Staff	50 %	Critical (2)	120,000	60,000
R4	Delivery deadline is too tight	Business	50 %	Critical (2)	100,000	50,000
R1	Customer changes requirements often	Product size / customer	60 %	Marginal (3)	80,000	48,000
R3	Server/hosting not ready in time	Technology / environment	30 %	Marginal (3)	40,000	12,000

The table is sorted by risk exposure (highest first). The project manager draws a **cut-off line**; risks above it (R2, R4, R1) are managed actively with an RMMM entry, others are re-evaluated periodically.

RMMM for R2 (key developer leaves): *Mitigation* — cross-train members, pair programming, keep documentation current; *Monitoring* — watch morale and workload; *Management* — assign the backup developer and re-plan the schedule.

(RMMM explanation 2 M, risk table 3 M, prioritisation 1 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

