

Software Engineering (315323)

Model Question Paper — Set B · Model Answers

Q.1

a) Types of software with examples.

2 M

1. **System software:** programs written to service other programs, e.g. operating system, compiler, drivers.
2. **Application software:** stand-alone programs that solve a specific business need, e.g. payroll, billing, railway reservation.
3. **Embedded software:** resides in read-only memory of a product and controls it, e.g. washing-machine controller, car ABS.
4. **Web / mobile applications:** network-centric software running in a browser or on a mobile device, e.g. e-commerce site, banking app.

(Any four types with example, ½ M each)

b) Layers of software engineering.

2 M

Tools
Methods
Process
A quality focus

Software engineering — a layered technology

- **Quality focus** (bedrock) — total quality management and continuous process improvement.
- **Process** — the foundation; framework of activities that holds the layers together.
- **Methods** — technical “how-to’s”: communication, requirements analysis, design, coding, testing, support.
- **Tools** — automated/semi-automated support for process and methods (CASE tools).

(Four layers, ½ M each)

c) Principles of deployment practice.

2 M

1. **Customer expectations must be managed.**
2. **A complete delivery package should be assembled and tested.**
3. **A support regime must be established before delivery.**
4. **Appropriate instructional materials must be provided.**

(Any four, ½ M each)

d) Data modelling; data object; attribute.

2 M

Data modelling is the process of identifying the data objects of a system, their attributes and the relationships among them, usually represented with an Entity-Relationship (ER) diagram.

Data object: a representation of composite information that the software must understand, e.g. Student, Course. **Attribute:** a property that describes a data object, e.g. roll_no, name.

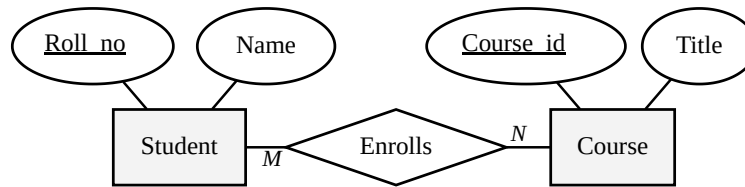


Fig.: Simple data model (ER diagram)

(Definition 1 M, two terms 1 M)

e) 4 P's of the management spectrum.

2 M

1. **People** — stakeholders and the software team (most important).
2. **Product** — the software to be built; its scope and decomposition.
3. **Process** — framework activities and task set used to build the product.
4. **Project** — all the work required to make the product a reality (planning, monitoring, control).

(½ M each)

f) WBS and its advantages.

2 M

Work Breakdown Structure (WBS) is a hierarchical decomposition of the total project work into smaller, manageable work packages (tasks).

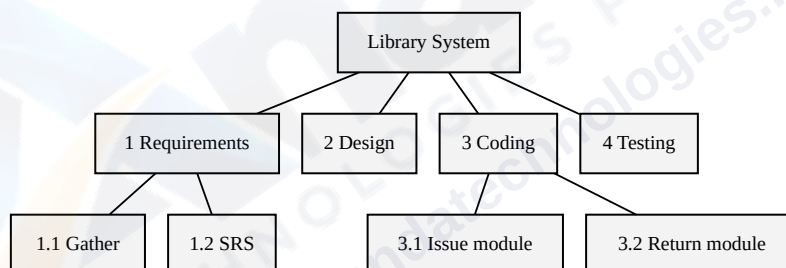


Fig.: Sample WBS

Advantages:

1. Makes estimation of cost, effort and time more accurate.
2. Helps in assigning responsibility of each work package to team members.
3. Gives a basis for scheduling (activity network, Gantt chart) and tracking progress.

(Definition 1 M, two advantages 1 M)

g) SQM vs SQA.

2 M

Software Quality Management (SQM)	Software Quality Assurance (SQA)
Broad, overall activity of managing quality: quality planning, quality assurance, quality control and quality improvement.	A part of SQM; set of planned, systematic activities that give confidence the process and product meet standards.
Defines quality policy, objectives and the quality management system for the organisation.	Implements the policy at project level — SQA plan, reviews, audits.
Responsibility of top management.	Responsibility of the SQA group / team.

(Any two points, 1 M each)

Q.2

a) RAD model.

4 M

RAD (Rapid Application Development) is an incremental process model that emphasises a very short development cycle (60–90 days) by using component-based construction. Several teams work in parallel on different modules.

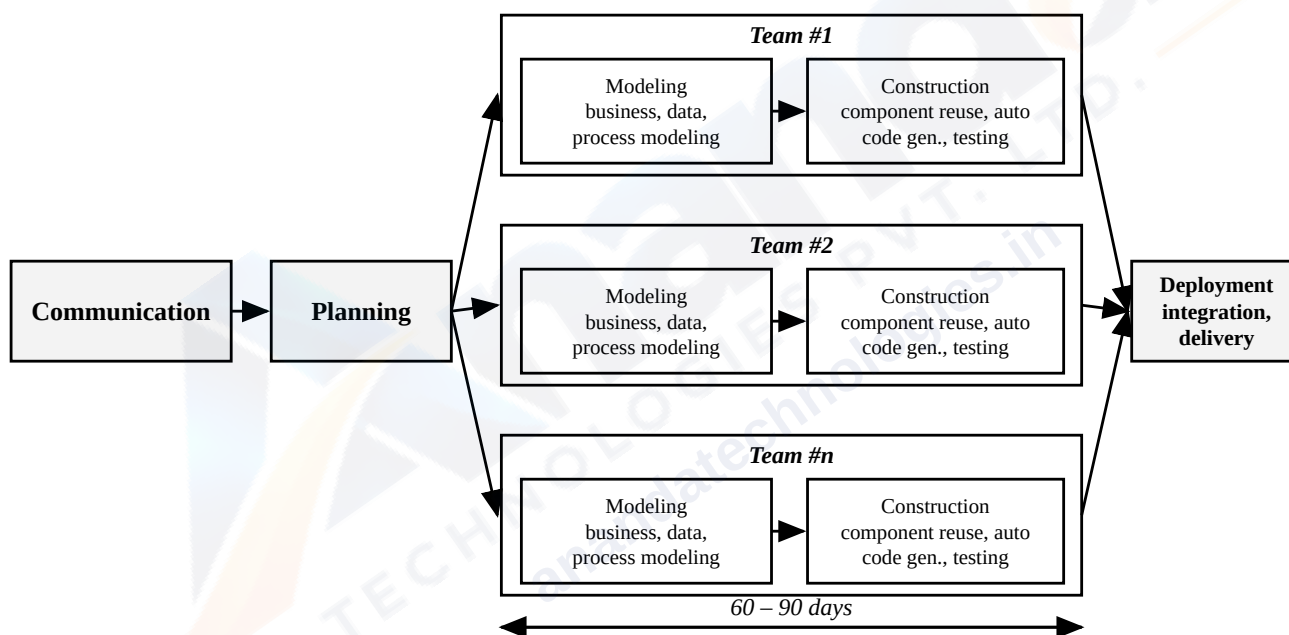


Fig.: RAD (Rapid Application Development) model

- **Communication:** understand the business problem and information characteristics.
- **Planning:** multiple teams work in parallel on different system functions.
- **Modeling:** business modeling, data modeling and process modeling.
- **Construction:** reuse of existing components and automatic code generation; testing.
- **Deployment:** integration of modules and delivery.

When to use RAD: requirements are well understood and project scope is constrained; the system can be modularised and each module completed in 2–3 months; reusable components and automated tools are available; skilled developers and active customer involvement are available.

(Diagram 2 M, explanation 1 M, when to use 1 M)

b) Structure, modularity, concurrency, verification.

4 M

1. **Structure:** A system is built from components and the relationships among them; the structure (hierarchy of modules, call graph) shows how the parts are organised. Example: main menu module calling Billing, Stock and Report modules.
2. **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.
3. **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.
4. **Verification:** Design must be verifiable — it should be possible to check that the design satisfies the requirements (traceability, reviews, test plans). Example: each use case in the SRS is mapped to design modules.

(Each concept 1 M)

c) V-model.

4 M

The **V-model** is a variation of the waterfall model that shows how verification and validation (testing) activities are related to each phase of development. As the team moves down the left side, requirements are refined into detailed design; after code generation the team moves up the right side, performing tests that validate each model created on the way down.

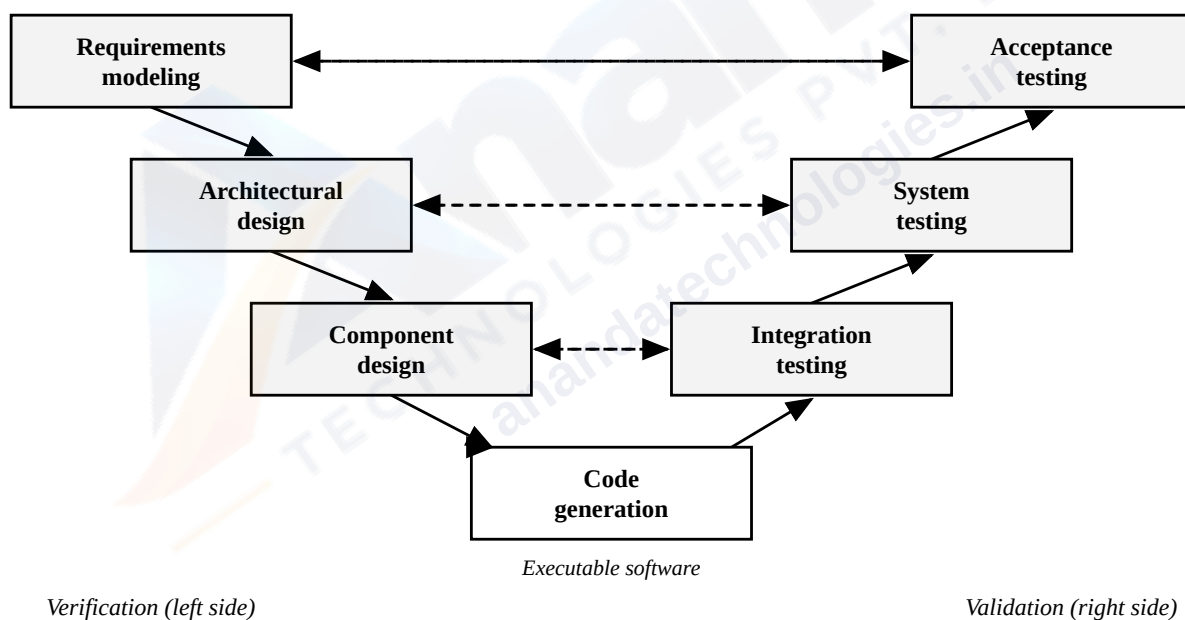


Fig.: V-model of software testing

- **Requirements modeling ↔ Acceptance testing:** customer checks that requirements are met.
- **Architectural design ↔ System testing:** whole system tested against the design.
- **Component design ↔ Integration testing:** interfaces between modules tested.
- **Code generation ↔ Unit testing:** individual modules tested.

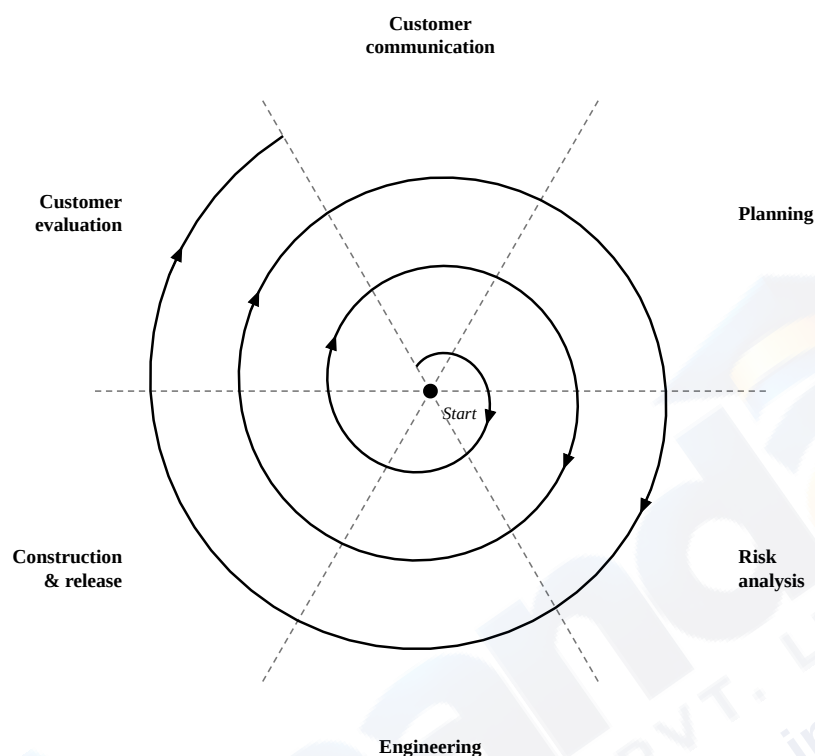
Advantage: test planning starts early, so defects are found early. **Disadvantage:** rigid like waterfall; no early prototype.

(Diagram 2 M, explanation 2 M)

d) Spiral model — online banking example.

4 M

The spiral model is a risk-driven evolutionary model. Development proceeds in loops; each loop passes through customer communication, planning, risk analysis, engineering, construction & release and customer evaluation.



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

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

Working with example — Online Banking application:

1. **First loop:** communicate with the bank to understand needs (login, balance enquiry, fund transfer); plan; analyse risks such as security and data loss; build a *concept/specification*; bank evaluates it.
2. **Second loop:** requirements are refined and a *prototype* of login and balance enquiry is built; security risk (OTP, encryption) is analysed and resolved; bank staff evaluate the prototype.
3. **Third loop:** fund transfer, statements and bill payment are added; performance risk is tested; release 1 is installed for selected customers.
4. **Further loops:** each loop adds features (loan, cards) and corrects problems until the complete system is released.

After every loop the cost and schedule are adjusted based on customer feedback.

(Diagram 1 M, working 1 M, example 2 M)

Q.3

a) Planning principles.

4 M

1. **Understand the scope of the project:** Scope gives the team a destination; plans cannot be made without knowing it.
2. **Involve stakeholders in the planning activity:** Stakeholders define priorities and constraints; they help to fix delivery dates and order of increments.
3. **Recognize that planning is iterative:** A plan is never fixed; it is adjusted as work proceeds and feedback is received.
4. **Estimate based on what you know:** Effort, cost and duration are estimated from the team's current understanding of the work.

(Also: be realistic; adjust granularity; define how quality will be ensured; accommodate change; track the plan frequently.)

(Any four, 1 M each)

b) Sequence diagram — ATM cash withdrawal.

4 M

A **sequence diagram** is a UML interaction diagram that shows how objects interact with each other in a time sequence. Objects are drawn at the top, time runs downwards along dashed *lifelines*, thin rectangles show *activation*, solid arrows are messages and dashed arrows are return messages.

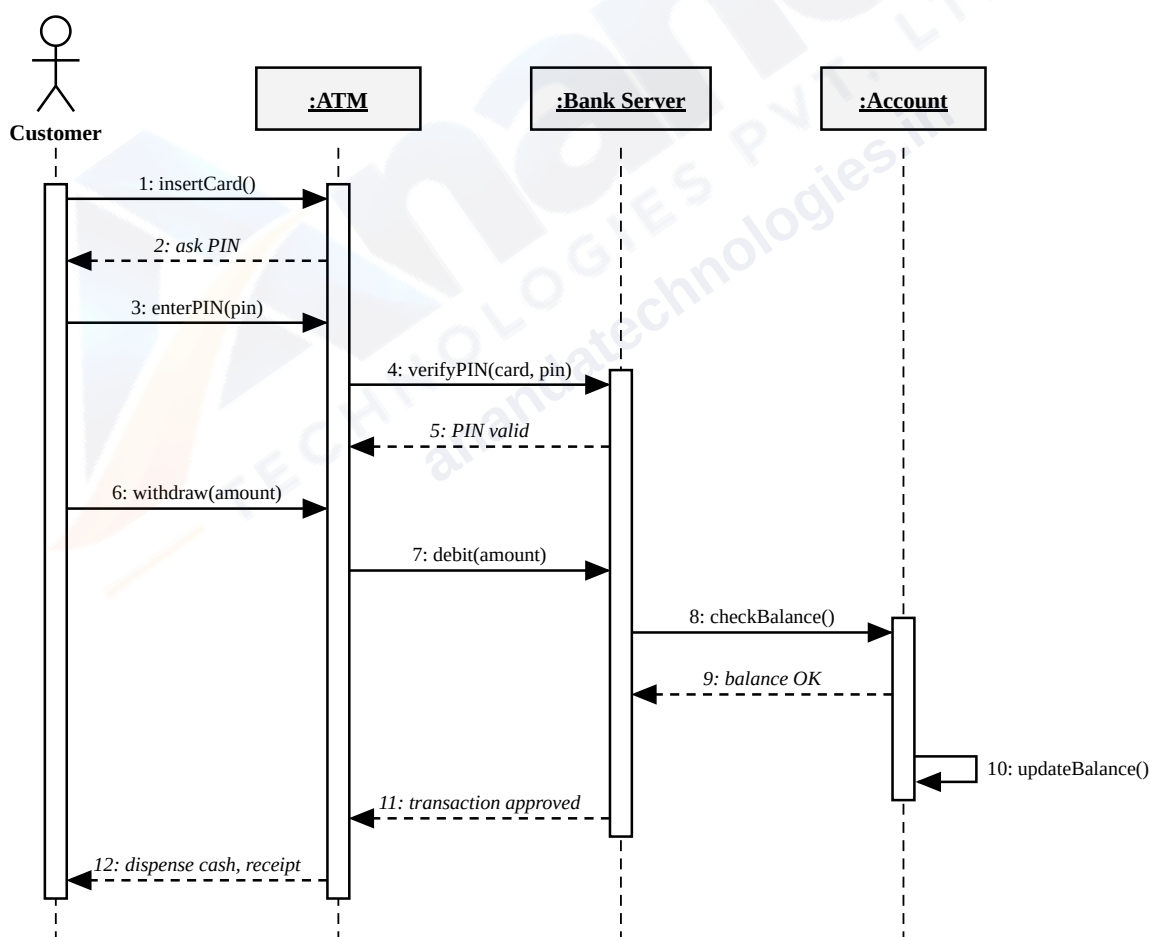


Fig.: Sequence diagram — cash withdrawal from ATM

Explanation: the customer inserts the card and enters the PIN; the ATM sends the PIN to the bank server for verification; on a valid PIN the customer requests withdrawal; the server checks the account balance and updates it; the ATM dispenses cash and prints the receipt.

(Diagram 3 M, explanation 1 M)

c) Static vs dynamic testing.

4 M

Static testing	Dynamic testing
Testing done without executing the code.	Testing done by executing the code with test data.
Done in the verification stage (early).	Done in the validation stage (after code is ready).
Techniques: reviews, walkthroughs, inspections, static analysis tools.	Techniques: black box and white box testing; unit, integration, system, acceptance testing.
Prevents defects; finds errors in documents, design and code.	Finds and fixes defects (failures) in the running software.
Cost of fixing defects is low.	Cost of fixing defects is higher.
Can start before coding (on SRS and design).	Can start only after code is written.

(Any four points, 1 M each)

d) Functional and non-functional requirements.

4 M

Functional requirements describe what the system should do — the services it provides, how it reacts to inputs and behaves in particular situations. Example: “The system shall allow a student to register for an exam.”

Non-functional requirements are constraints on the services or functions — performance, reliability, security, usability, standards. Example: “The system shall respond within 2 seconds.”

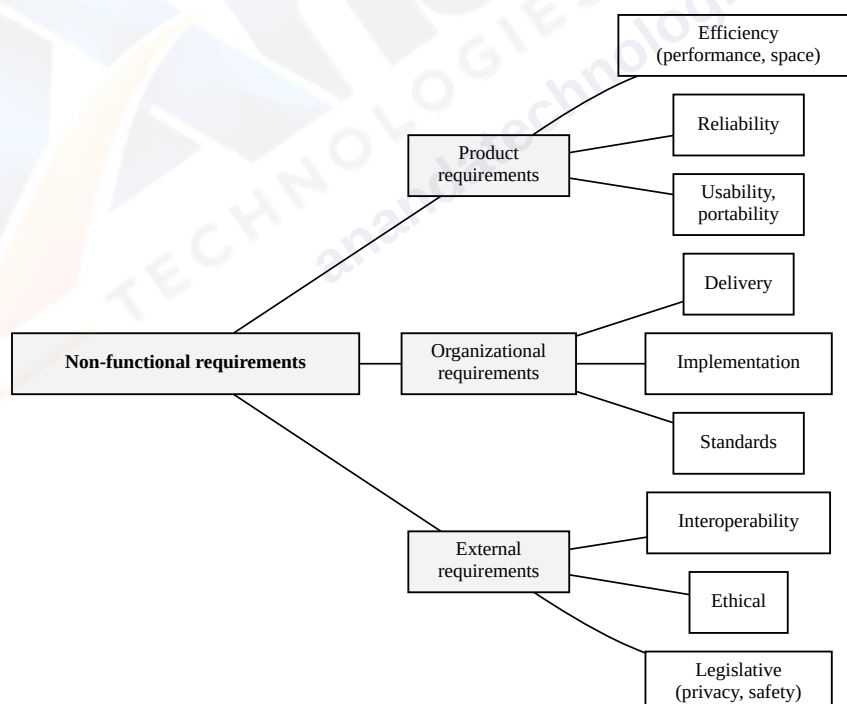


Fig.: Types of non-functional requirements

- **Product requirements:** specify product behaviour — speed, memory, reliability, portability, usability.
- **Organizational requirements:** derived from customer/developer organisation policies — delivery, implementation language, process standards.
- **External requirements:** arise from factors external to the system — interoperability with other systems, legal and ethical requirements.

(Functional 1 M, non-functional 1 M, classification 2 M)

Q.4

a) Extreme Programming (XP).

4 M

Extreme Programming (XP) (Kent Beck) is the most widely used agile process. It uses an object-oriented approach and has four framework activities:

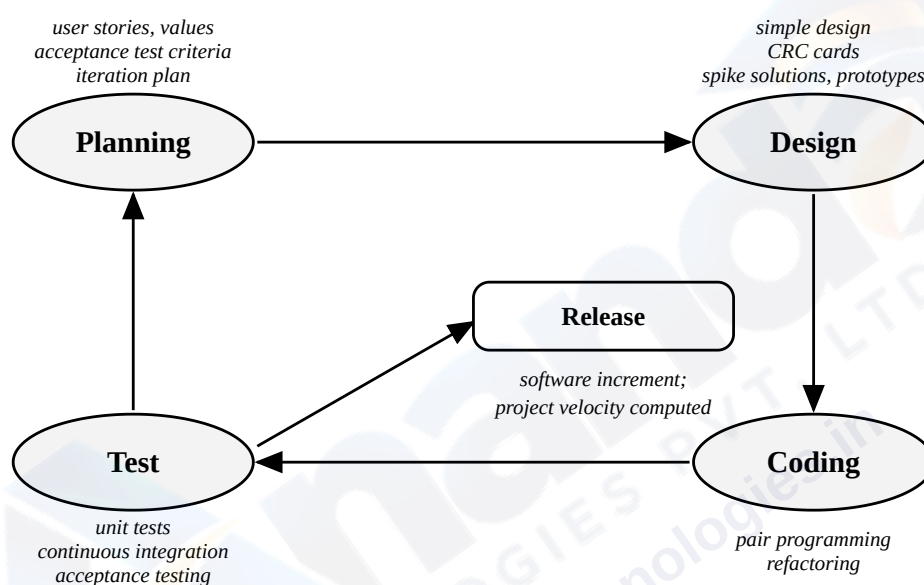


Fig.: Extreme Programming (XP) process

1. **Planning:** starts with listening; customers write *user stories* on index cards with a priority; the team estimates cost; stories are grouped into the next release; after the first release *project velocity* is computed.
2. **Design:** follows the KIS (keep it simple) principle; CRC (class-responsibility-collaborator) cards are used; for difficult problems a *spike solution* (prototype) is built; refactoring is encouraged.
3. **Coding:** unit tests are written before coding; *pair programming* — two people work at one computer; code is integrated continuously.
4. **Testing:** unit tests are automated and run daily; *acceptance tests* are specified by the customer and focus on overall features.

(Diagram 2 M, activities 2 M)

b) Use case diagram — Online Examination System.

4 M

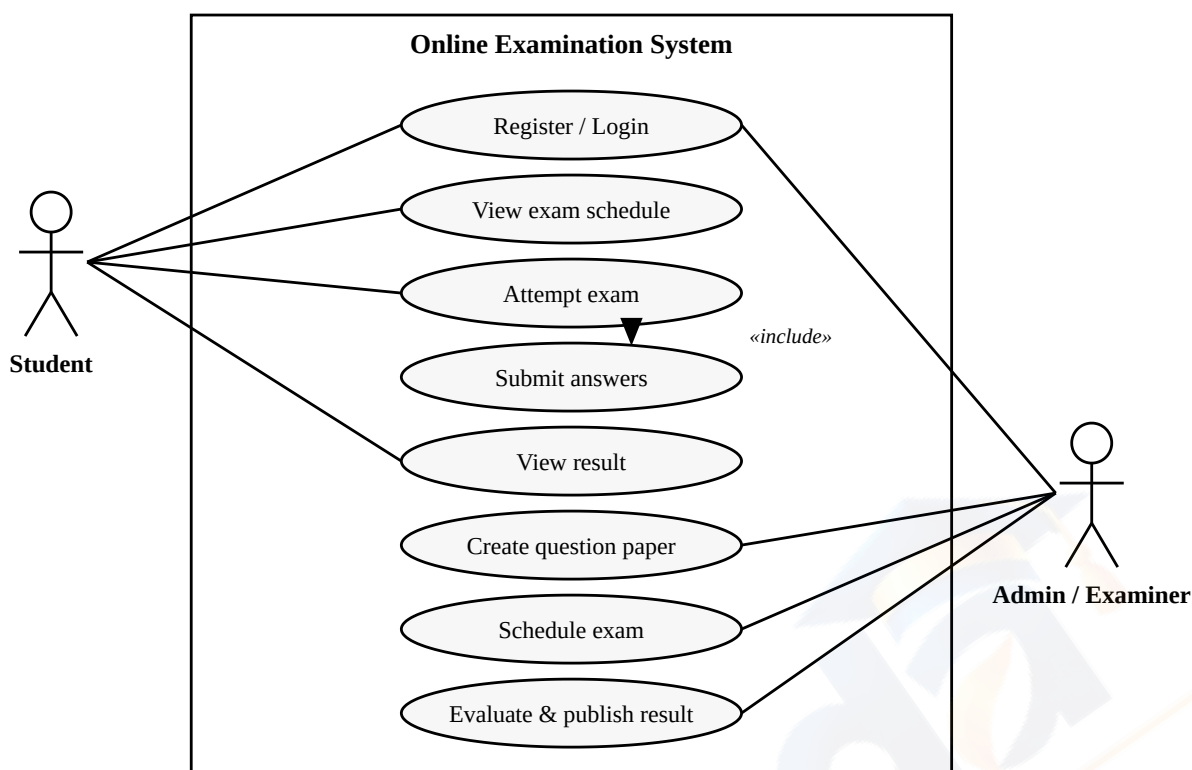


Fig.: Use case diagram — Online Examination System

Actors: Student and Admin/Examiner. **Use cases:** Register/Login, View exam schedule, Attempt exam (includes Submit answers), View result, Create question paper, Schedule exam, Evaluate & publish result.

(Actors 1 M, use cases 2 M, boundary & links 1 M)

c) Earned value analysis — numerical.

4 M

Given: BCWS (PV) = ₹ 1,20,000, BCWP (EV) = ₹ 1,00,000, ACWP (AC) = ₹ 1,10,000, BAC = ₹ 5,00,000

Term	Formula	Calculation	Result	Interpretation
SV	BCWP – BCWS	1,00,000 – 1,20,000	₹ -20,000	Negative — behind schedule
CV	BCWP – ACWP	1,00,000 – 1,10,000	₹ -10,000	Negative — over budget
SPI	BCWP / BCWS	1,00,000 / 1,20,000	0.83	< 1 — only 83 % of planned work done
CPI	BCWP / ACWP	1,00,000 / 1,10,000	0.91	< 1 — ₹ 0.91 of work per ₹ 1 spent

Estimate at completion EAC = BAC / CPI = 5,00,000 / 0.9091 = ₹ 550,000 (likely overrun of ₹ 50,000).

Status: the project is behind schedule and over budget; corrective action (re-planning, adding resources, reducing scope) is needed.

(Each term with interpretation 1 M)

d) CMMI levels and process areas.

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:

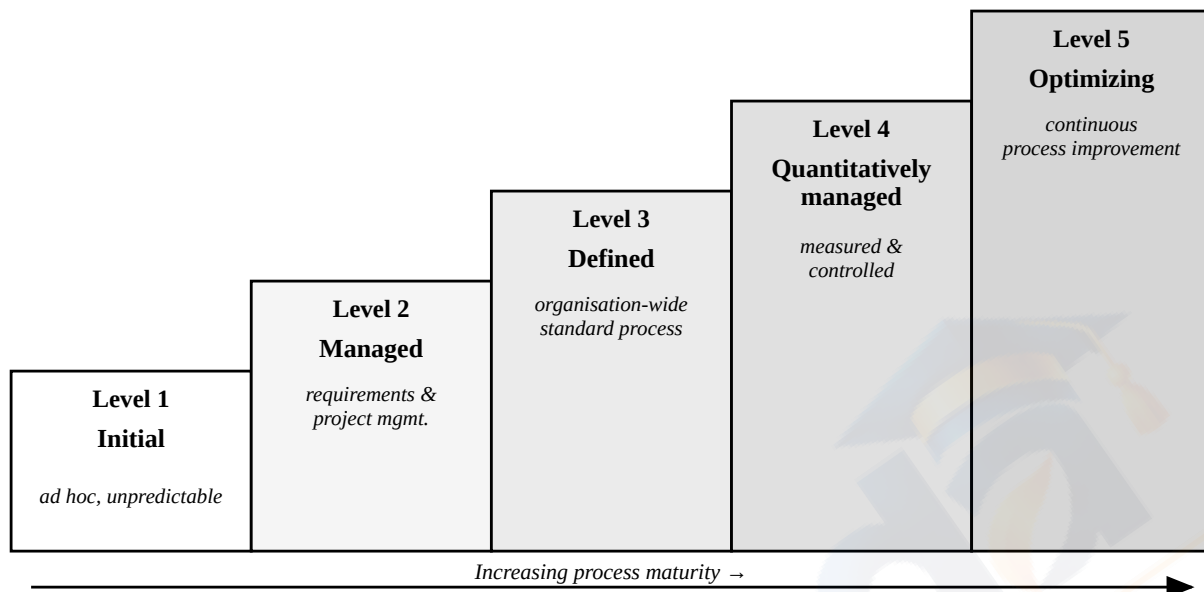


Fig.: CMMI maturity levels (staged representation)

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

Key process areas (PAs) at each level:

Maturity level	Process areas
2 — Managed	Requirements management, project planning, project monitoring and control, supplier agreement management, measurement and analysis, process and product quality assurance, configuration management
3 — Defined	Requirements development, technical solution, product integration, verification, validation, organizational process focus and definition, organizational training, integrated project management, risk management, decision analysis and resolution
4 — Quantitatively managed	Organizational process performance, quantitative project management
5 — Optimizing	Organizational performance management (innovation and deployment), causal analysis and resolution

(Diagram/levels 2 M, process areas 2 M)

e) Basic COCOMO with example.

4 M

COCOMO (Constructive Cost Model) was proposed by Barry Boehm (1981). It is an empirical (heuristic) model that estimates effort, development time and staff from the size of the software in KLOC. It has three levels — **Basic**, **Intermediate** (adds 15 cost drivers) and **Detailed** (cost drivers per phase).

Project modes:

- **Organic:** small, simple project (about 2–50 KLOC), small experienced team, familiar environment, e.g. payroll, inventory.
- **Semi-detached:** medium size (about 50–300 KLOC), mixed team experience, e.g. compiler, DBMS.
- **Embedded:** large/complex (over 300 KLOC) with tight hardware, software and operational constraints, e.g. air-traffic control, ATM network.

Basic COCOMO equations: Effort $E = a \times (\text{KLOC})^b$ person-months · Development time $D = c \times (E)^d$ months · Staff = E / D persons

Mode	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	1.2	2.5	0.32

Example: An organic project of 32 KLOC:

Given: Size = 32 KLOC

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

$$E = 2.4 \times (32)^{1.05} = 2.4 \times 38.05 = \mathbf{91.33 \text{ person-months}}$$

$$D = 2.5 \times (91.33)^{0.38} = \mathbf{13.90 \text{ months}}$$

$$\text{Average staff} = E / D = 91.33 / 13.90 \approx \mathbf{6.6 \text{ persons}}$$

Mode	Effort E (PM)	Dev. time D (months)	Staff (persons)
Organic	91.33	13.90	6.6

Note: COCOMO II (1995) extends the model for modern development with three sub-models: application composition, early design and post-architecture, using object points / function points / LOC.

(Explanation 2 M, example 2 M)

Q.5

a) Prototyping model.

6 M

The **prototyping model** is an evolutionary model used when the customer defines general objectives but not detailed requirements, or when the developer is unsure about an algorithm or interface. A quick working model (prototype) is built and refined using customer feedback.

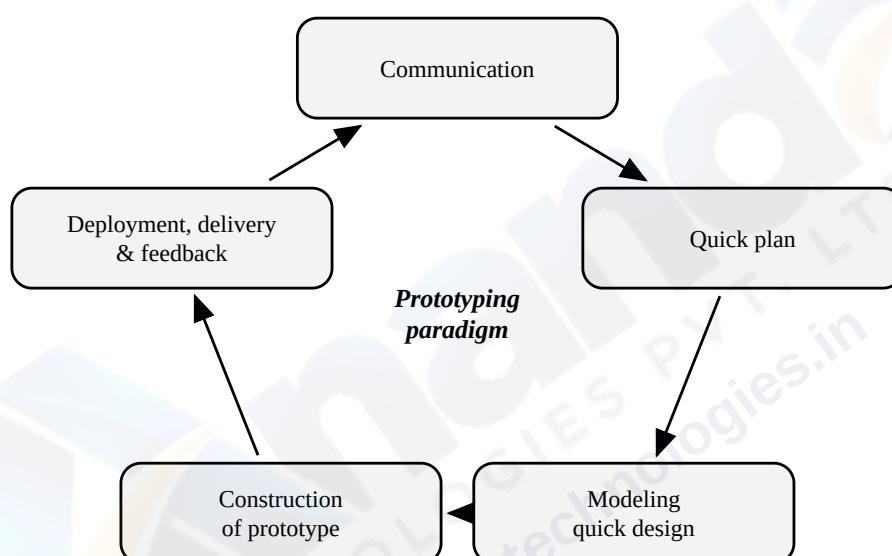


Fig.: Prototyping model (iterates until customer is satisfied)

1. **Communication:** developer and customer meet to define overall objectives and known requirements.
2. **Quick plan:** a prototyping iteration is planned quickly.
3. **Modeling (quick design):** focuses on visible aspects — user interface and output formats.
4. **Construction of prototype:** the quick design leads to building a prototype.
5. **Deployment, delivery and feedback:** the prototype is evaluated by the customer; feedback refines requirements. The cycle repeats until the customer is satisfied.

Advantages:

- Customer sees a working model early; requirements become clear.
- Errors and missing functions are detected early.

- Better user involvement and satisfaction.
- Useful when requirements are unclear or changing.

Disadvantages:

- Customer may consider the prototype as the final product and demand quick delivery.
- Developer may make implementation compromises (poor language, inefficient algorithm) to build it quickly.
- Repeated iterations may increase cost and time.
- Poor documentation.

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

b) Class diagram — Bank Management System.

6 M

A **class diagram** shows the static structure of a system — classes, their attributes and operations, and relationships. Each class is a rectangle with three compartments: name, attributes, operations (– private, + public).

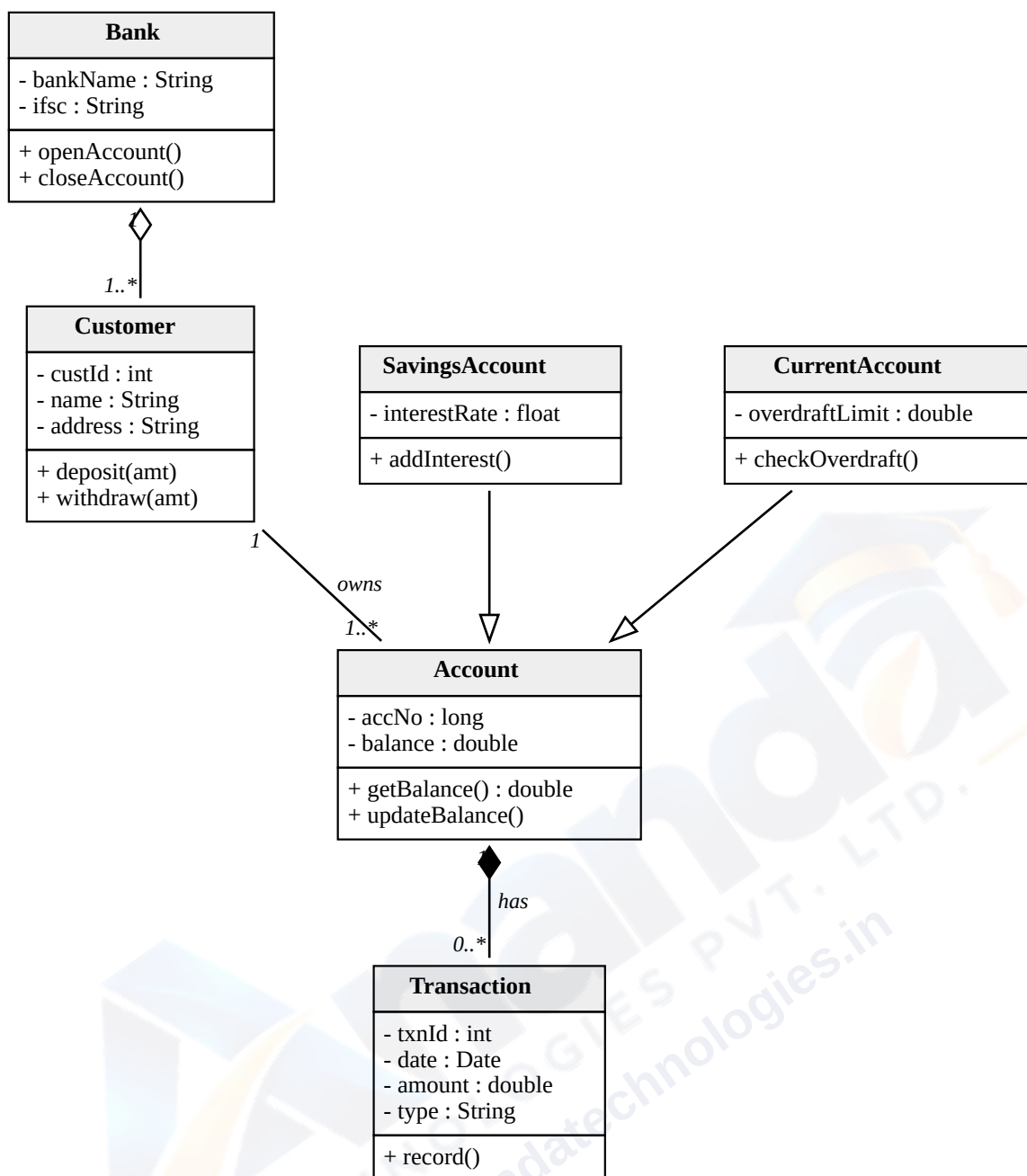


Fig.: Class diagram — Bank Management System

- **Generalization** (hollow triangle): SavingsAccount and CurrentAccount are kinds of Account (inheritance).
- **Aggregation** (hollow diamond): a Bank has many Customers; customers can exist independently.
- **Composition** (filled diamond): an Account has Transactions; transactions cannot exist without the account.
- **Association with multiplicity**: one Customer owns 1..* Accounts.

(Classes 3 M, relationships & multiplicity 3 M)

c) PERT — expected times, critical path, duration, SD.

6 M

Three time estimates in PERT:

- **Optimistic time (t_o)**: minimum time if everything goes better than expected.

- **Most likely time (t_m):** time required under normal conditions.
- **Pessimistic time (t_p):** maximum time if everything goes wrong (excluding disasters).

Expected time $t_e = (t_o + 4t_m + t_p) / 6$ · Variance $\sigma^2 = ((t_p - t_o) / 6)^2$

Activity	i-j	t_o	t_m	t_p	t_e	σ^2
A	1-2	2	3	4	3.00	0.11
B	1-3	4	6	8	6.00	0.44
C	2-4	3	5	13	6.00	2.78
D	3-4	2	3	4	3.00	0.11
E	3-5	6	8	10	8.00	0.44
F	4-6	3	4	5	4.00	0.11
G	5-6	2	3	10	4.00	1.78

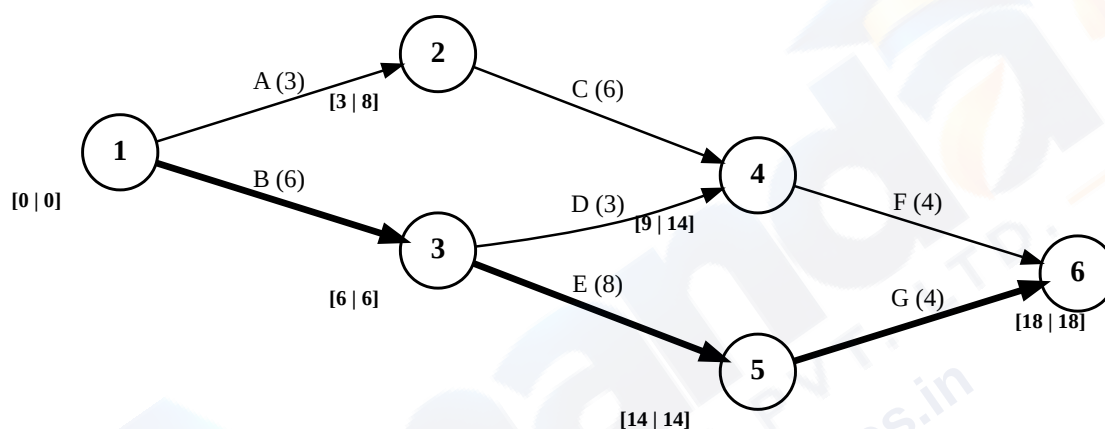


Fig.: PERT network with expected times (critical path thick)

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	3	6	9	14	18
Latest time L	0	8	6	14	14	18

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
A	1-2	3	0	3	5	8	5
B *	1-3	6	0	6	0	6	0
C	2-4	6	3	9	8	14	5
D	3-4	3	6	9	11	14	5
E *	3-5	8	6	14	6	14	0
F	4-6	4	9	13	14	18	5
G *	5-6	4	14	18	14	18	0

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

Critical path: 1 → 3 → 5 → 6 (B–E–G)

Project duration = 18

Variance of project = sum of variances on critical path = 0.44 + 0.44 + 1.78 = 2.67

Standard deviation $\sigma = \sqrt{2.67} = 1.63$ weeks. Expected project duration = **18 weeks.**

(Three estimates 1 M, t_e table 1 M, network 2 M, critical path & SD 2 M)

Q.6

a) Requirement engineering tasks.

6 M

Requirement engineering is the broad spectrum of tasks and techniques that lead to an understanding of requirements. It has seven distinct tasks:

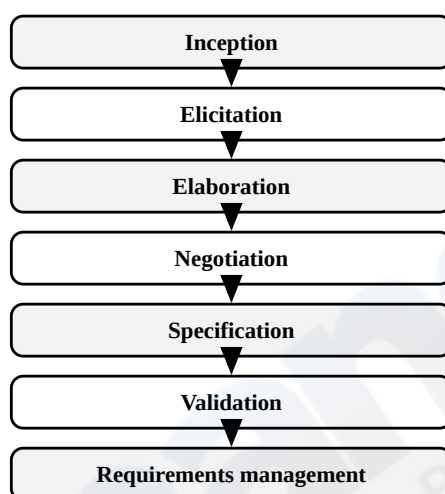


Fig.: Requirement engineering tasks

1. **Inception:** establishes basic understanding of the problem, the people who want a solution, the nature of the solution and the effectiveness of communication.
2. **Elicitation:** drawing out requirements from stakeholders — what the objectives are, what is to be accomplished; problems of scope, understanding and volatility are handled.
3. **Elaboration:** information from inception and elicitation is expanded and refined into a requirements (analysis) model — user scenarios, classes, behaviour.
4. **Negotiation:** conflicting requirements of different stakeholders are reconciled by ranking/prioritising them so that each party is satisfied (win-win).
5. **Specification:** requirements are written down — SRS, models, use cases, prototypes.
6. **Validation:** work products are reviewed (technical review) to check that requirements are unambiguous, consistent, complete and correct.
7. **Requirements management:** activities to identify, control and track requirements and changes throughout the project (traceability tables).

(Diagram 1 M, tasks 5 M)

b) Function point computation.

6 M

Step 1 — Unadjusted function points (average weights):

Information domain value	Count	Weight (average)	Count × weight
External inputs (EI)	32	4	128
External outputs (EO)	60	5	300
External inquiries (EQ)	24	4	96
Internal logical files (ILF)	8	10	80
External interface files (EIF)	2	7	14
Count total (UFP)			618

Step 2 — Value adjustment:

$$\Sigma F_i = 4 + 3 + 2 + 4 + 3 + 5 + 4 + 3 + 3 + 4 + 2 + 3 + 1 + 4 = 45$$

$$\text{CAF (value adjustment)} = 0.65 + 0.01 \times 45 = \mathbf{1.10}$$

$$\text{FP} = \text{UFP} \times \text{CAF} = 618 \times 1.10 = \mathbf{679.80}$$

Function points = 679.80

FP can be used to estimate effort and cost, e.g. if productivity is 10 FP per person-month, effort $\approx$ 68.0 person-months.

(UFP 3 M, CAF 2 M, FP 1 M)

c) Six Sigma — DMAIC, DMADV; online food ordering.

6 M

Six Sigma is a widely used strategy for statistical quality assurance, popularised by Motorola in the 1980s. It uses data and statistical analysis to measure and improve a company's operational performance by identifying and eliminating defects. Six Sigma quality means not more than **3.4 defects per million occurrences**.

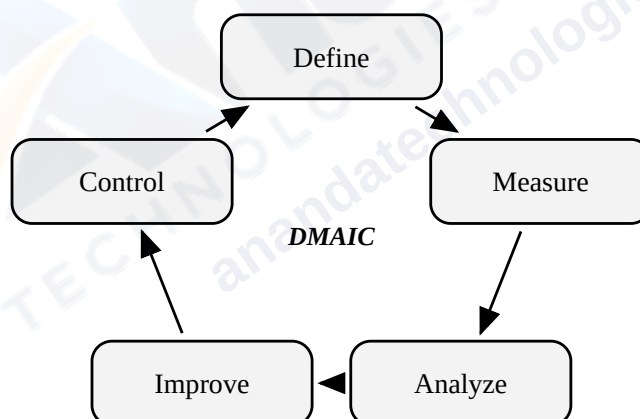


Fig.: DMAIC cycle (Six Sigma — improving an existing process)

DMAIC (for improving an existing process):

1. **Define** customer requirements, deliverables and project goals through well-defined methods of customer communication.
2. **Measure** the existing process and its output to determine current quality performance (collect defect metrics).
3. **Analyze** defect metrics and determine the vital few causes.

4. **Improve** the process by eliminating the root causes of defects.
5. **Control** the process to ensure that future work does not reintroduce the causes of defects.

DMADV (for developing a *new* process/product):

1. **Define** customer requirements and project goals.
2. **Measure** customer needs and specifications (critical-to-quality characteristics).
3. **Analyze** process options to meet the customer requirements.
4. **Design** the process (details) to meet the customer requirements.
5. **Verify** the design performance and its ability to meet customer requirements.

Applying Six Sigma (DMAIC) to an Online Food Ordering System:

Phase	Application
Define	Customers complain of wrong orders and late delivery; goal: reduce failed/late orders to near Six Sigma level.
Measure	Collect data: orders per day, failed payments, wrong items, average delivery time, app crash reports.
Analyze	Find root causes — payment gateway time-outs, menu not synced with restaurant stock, poor route assignment.
Improve	Add payment retry, real-time stock update, automatic nearest-rider assignment; test the changes.
Control	Monitor defect rate dashboards, set alerts, review metrics weekly so defects do not return.

(DMAIC 2 M, DMADV 2 M, application 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