UNIT–2 : Software Requirement Specifications & Quality Assurance (PYQ Solution)

AKTU · Software Engineering (BCS601) · Solutions by **Chirag Jain**, Ycotes

SECTION A Answers (≈70 Words Each)

Q1. Discuss the Requirements Engineering Process. Write down the most generalized template used for describing Use Cases. (May 2026)

The Requirements Engineering (RE) process has four main activities: feasibility study (is the system worth building), elicitation and analysis (interviews, questionnaires, observation, prototyping), specification (writing the SRS) and validation (reviews, prototyping, test-case generation), supported by requirements management for changes. A use case template contains: use case name and ID, actor(s), description, preconditions, trigger, basic (main) flow, alternate flows, exceptions, postconditions, business rules and priority.

Q2. Define software quality. Give a comprehensive list of software quality attributes and discuss the procedure to assess the software quality. (May 2026)

Software quality is the degree to which software meets the stated and implied needs of its users and conforms to its specification (fitness for use). Main attributes: functionality, reliability, usability, efficiency, maintainability, portability, security, testability, reusability, interoperability, correctness, integrity and flexibility. To assess quality, each attribute is broken into measurable sub-attributes, metrics are defined and data are collected through reviews, testing and defect tracking. The values are weighted and combined into a quality score, which is compared with the targets.

Q3. What is Software Risk Analysis Management? (May 2025)

Risk analysis and management is the process of identifying, assessing and controlling events that may threaten a software project's schedule, cost or

quality. It has steps: risk identification (technical, project, business risks), risk analysis/projection (probability and impact, risk exposure = probability × loss), risk planning (avoid, minimise or accept) and risk monitoring and control (RMMM plan, regular review). Its purpose is to handle problems before they occur.

Q4. Define Use Case and write most commonly accepted template for the Use Case giving description of its components. (May 2025)

A use case describes a sequence of interactions between an actor (user or external system) and the system to achieve a specific, observable goal. Its common template components are: Name/ID (unique title), Actors (who interacts), Description (goal), Preconditions (state needed before start), Main flow (numbered normal steps), Alternate/exception flows (variations and errors), Postconditions (state after success) and Priority/frequency.

Q5. Mention any two non-functional requirements on software to be developed. (2023)

Non-functional requirements describe how well the system performs rather than what it does. Two examples: (1) Performance — the system must respond to any search query within 2 seconds for 1000 concurrent users. (2) Security — all user passwords must be stored in encrypted form and access must be role-based. Others are reliability, usability, portability and maintainability.

SECTION B Answers (≈250 Words Each)

Q1. Discuss the selection process parameters for a life cycle model. (May 2026)

Choosing a **life cycle model** (waterfall, prototype, iterative, spiral, RAD, agile) depends on the nature of the project. The main parameters are:

- **Requirements clarity and stability** – stable, well-understood requirements suit waterfall; vague or changing ones suit prototyping, iterative or agile models.
- **Project size and complexity** – large, complex projects need incremental or spiral models; small ones can use waterfall.
- **Risk level** – high technical or business risk favours the spiral model because it analyses risk in each loop.
- **Schedule and time-to-market** – tight deadlines favour RAD, incremental delivery or agile.
- **Customer involvement and feedback** – if the customer can review often, prototype or agile models work well.
- **Team size, skill and experience** – a skilled team can handle flexible models; a new team needs a defined, structured model.
- **Technology and tools** – unfamiliar technology needs prototyping; mature technology permits waterfall.
- **Budget and cost control** – fixed-cost projects need predictable models.
- **Documentation and maintenance needs** – regulated domains need heavy documentation.
- **Need for early working software** – incremental models deliver parts early.

Situation	Suitable model
Clear, stable needs	Waterfall
Unclear needs	Prototype
High risk, large	Spiral
Need early release	Incremental / RAD
Frequent change	Agile

Conclusion: no model is best for every project; the choice is made by weighing these parameters against the project's characteristics.

Q2. What is risk exposure? What techniques can be used to control each risk? (May 2026)

Risk exposure (RE) measures the size of a risk: **RE = P(UO) × L(UO)**, where P is the probability of an unsatisfactory outcome and L is the loss if it occurs. Example: a 30% chance of a ₹2,00,000 loss gives RE = ₹60,000. Risks are ranked by exposure so that effort is spent on the largest ones first.

Risk control techniques (risk management planning, RMMM):

- **Risk avoidance** – change the plan so the risk cannot occur, e.g. drop a risky feature or use proven technology.
- **Risk reduction/mitigation** – lower probability or impact: training, prototyping, reviews, extra testing, hiring backups.
- **Risk transfer** – shift the risk to another party through insurance, outsourcing or fixed-price contracts.
- **Risk acceptance/contingency** – accept the risk and keep a contingency plan, reserve time and budget.
- **Risk monitoring** – track risk indicators in regular reviews and re-evaluate priorities.

Steps: identify risks → estimate probability and impact → compute exposure → rank → plan a response → monitor.

Risk type	Example control
Schedule slip	Buffer time, incremental delivery
Staff turnover	Documentation, cross-training
Changing requirements	Change control, prototyping
Technical failure	Early prototype, spike

Conclusion: risk exposure ranks the risks and the matching avoidance, reduction, transfer or contingency technique keeps the project under control.

Q3. List various software quality attributes and discuss the procedure for assessment of software quality. Obtain the expression for Error Index (which acts as Quality Indicator for software) through the Statistical Quality Assurance procedure where data is collected and analysed over a defined period of time. (May 2025)

Quality attributes: functionality, reliability, usability, efficiency, maintainability, portability, security, testability, reusability and interoperability.

Assessment procedure:

1. Decide the quality goals and the important attributes.
2. Break each attribute into measurable sub-factors and metrics.
3. Collect data through reviews, inspections, testing and field reports.
4. Compare the measured values with targets and give weights.
5. Combine them into a quality score and take corrective action.

Statistical Quality Assurance (SQA) steps: (1) collect and categorise defect information; (2) trace each defect to its cause; (3) apply the **Pareto principle** to isolate the vital 20% of causes producing 80% of defects; (4) correct those causes.

Error Index (EI): for a defined period, let

- E_i = total errors found in phase i, S_i = serious errors, M_i = moderate errors, T_i = trivial errors,
- PS = size of the product (KLOC, FP, etc.),
- w_s, w_m, w_t = weighting factors (typically 10, 3, 1) for serious, moderate and trivial errors.

Phase index: **PI_i = w_s (S_i/E_i) + w_m (M_i/E_i) + w_t (T_i/E_i)**

Error Index: EI = Σ (i × PI_i) / PS = (PI_1 + 2 PI_2 + 3 PI_3 + … + i PI_i) / PS

The factor i gives more weight to errors found in later phases, because they are costlier to fix. A falling EI over successive releases shows improving quality.

Conclusion: EI converts defect data into one number that works as a quality indicator.

Q4. Suppose you have to develop Library Management Software with common functionalities such as authentication of the user, issue and

return of books by the user, report generation by the librarian, and database maintenance by the library administrator etc. Draw a complete (at least up to 2-levels) Data Flow Diagram (DFD) and State Transition Diagram for this software. (May 2025)

External entities: User (member), Librarian, Administrator. **Data stores:** D1 Users, D2 Books, D3 Issue/Return records.

Level 0 (context diagram): one process **0 Library Management System** with flows: User → login details, book request, return; system → login result, issue slip, due notice; Librarian → report request, system → reports; Administrator → master data updates, system → confirmation.



Level-0 DFD (context diagram)

Level 1 DFD:

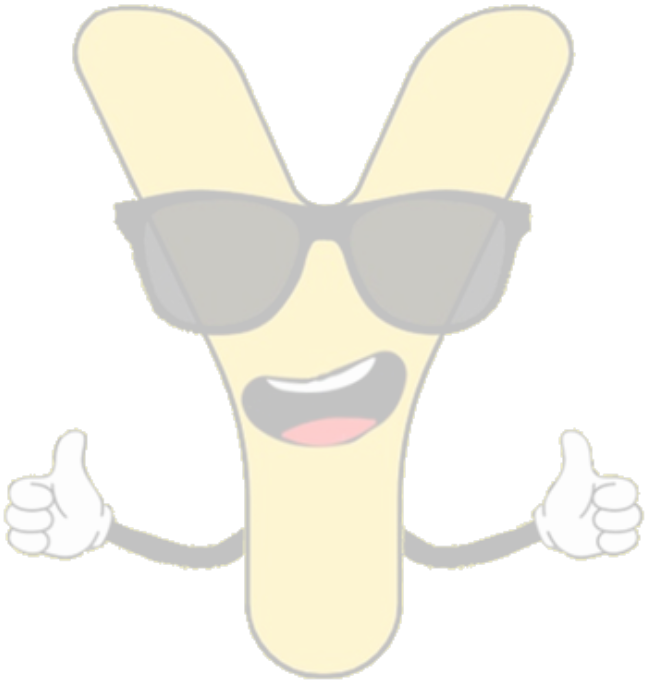
- **1.0 Authenticate user** – reads D1 Users.
- **2.0 Issue/return books** – reads D2 Books, writes D3 Issue records.
- **3.0 Generate reports** – reads D2 and D3, sends reports to the librarian.
- **4.0 Maintain database** – administrator adds/updates/deletes users and books in D1, D2.

Level 2 (process 2.0): 2.1 Search book → 2.2 Check availability and member limit → 2.3 Record issue (D3) → 2.4 Accept return, calculate fine → 2.5 Update book status (D2).

State Transition Diagram (for a book / session):

State	Event	Next state
Logged out	valid login	Logged in
Logged in	invalid login (3 times)	Locked
Logged in	issue request	Book Issued
Book Issued	return	Available
Book Issued	due date passed	Overdue
Overdue	pay fine and return	Available
Logged in	logout	Logged out

Conclusion: the DFD shows how data move between processes and stores, and the state table shows the states of the book and the user session.



SECTION C Answers (≈500 Words Each)

Q1. Draw the Data Flow Diagram (DFD) up to 2nd level for the following software to be developed for the supermarket. The software needs customer to supply his residence address, telephone number, and the driving license number. The customer is then assigned a unique customer number (CN) by the computer. After making any purchases, the customer presents his CN to the check-out staff who credits the value of his purchase against his CN. At the end of each year, the supermarket awards surprise gifts to 10 customers who make the highest total purchase over the year. Also, it awards a 22 carat gold coin to every customer whose purchase exceeds Rs. 10,000. The entries against the CN are reset on the last day of every year after the prize winners' lists are generated. (May 2026)

A **Data Flow Diagram (DFD)** shows how data move between external entities, processes and data stores. Here we use the symbols: external entity (rectangle), process (circle), data store (open rectangle) and data flow (arrow).

External entities: Customer, Check-out Staff, Supermarket Management. **Data stores:** D1 Customer file (CN, address, phone, licence no., total purchase), D2 Purchase/Winners file.

Level 0 – Context diagram: single process **0 Supermarket Customer System**.

- Customer → registration details (address, phone, licence no.); system → CN.
- Check-out staff → CN, purchase value; system → purchase confirmation.
- System → prize winners' lists, gift and coin list → Management.

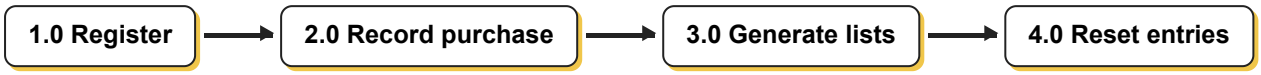


Level-0 DFD (context diagram)

Level 1 – decomposition of process 0

- **1.0 Register customer** – input: address, phone, licence no.; output: CN to customer; writes D1.
- **2.0 Record purchase** – input: CN and purchase amount from check-out staff; updates total purchase in D1; output: receipt/confirmation.

- **3.0 Generate prize lists** – reads D1 at year end; outputs the "top 10" and "above ₹10,000" lists to Management.
- **4.0 Reset yearly entries** – after list generation, resets the purchase totals in D1 on the last day of the year.



Level-1 DFD processes (all use customer file D1)

Level 2 – decomposition of main processes

Process 1.0 Register customer

- 1.1 Accept and validate details (all fields present, licence format).
- 1.2 Generate unique CN (next free number).
- 1.3 Store customer record in D1 and issue CN to the customer.

Process 2.0 Record purchase

- 2.1 Accept CN and amount and validate the CN against D1.
- 2.2 Add the amount to the customer's yearly total in D1.
- 2.3 Print receipt for the customer.

Process 3.0 Generate prize lists

- 3.1 Read yearly total of every customer from D1.
- 3.2 Sort customers by total and select the **top 10** (surprise gifts).
- 3.3 Select all customers with total **> ₹10,000** (gold coin).
- 3.4 Prepare and print the two winners' lists for Management and store in D2.

Process 4.0 Reset yearly entries

- 4.1 Check that lists have been generated (D2).
- 4.2 Set the yearly total of all CNs to zero on the last day of the year.

Data dictionary: CN = unique integer; Customer record = CN + address + phone + licence no. + yearly total.

Conclusion: the DFD up to level 2 shows registration, purchase recording, yearly prize-list generation and reset, with the customer file as the main data store.

Q2. Draw the Use Case diagram for Library Management System with its common functionalities. Discuss the Use Case for 'Login' as per Use

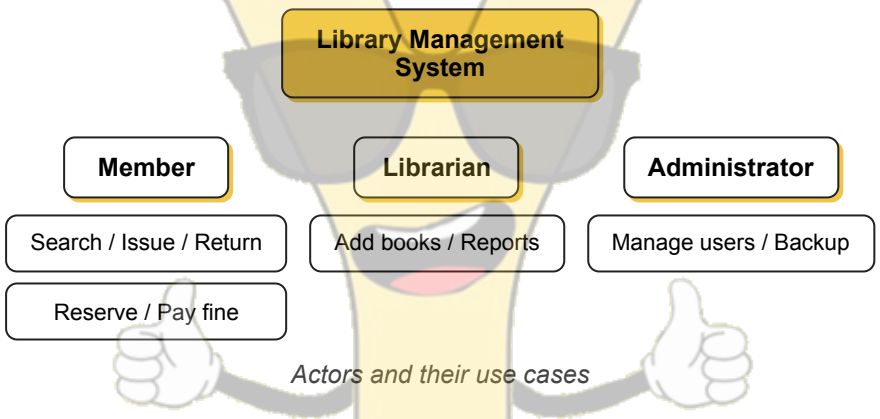
Case template. (May 2026)

A **use case diagram** shows the actors, the use cases (functions) and their relationships in a system. Actors are drawn as stick figures, use cases as ovals inside a system boundary, joined by association lines, with **«include»** and **«extend»** relationships.

Actors: Member (student/faculty), Librarian, Administrator.

Use cases

- **Member:** Login, Search book, Issue book, Return book, Renew book, Reserve book, Pay fine, View account.
- **Librarian:** Login, Add/Update book, Issue and return book, Generate reports, Manage fines, Manage reservations.
- **Administrator:** Login, Manage users, Backup database, Set rules and fine rates.
- **Relationships:** Issue book «include» Check availability and Verify membership; Return book «extend» Calculate fine (when late); all actions «include» Login.



Actors and their use cases

Use case: Login (template)

Component	Description
Use case ID / name	UC-01 / Login
Actors	Member, Librarian, Administrator
Description	Allows a registered user to access the system by authentication
Preconditions	User is registered; system is running
Trigger	User opens the application and selects Login
Main flow	1. User enters user ID and password. 2. System validates them against the user database. 3. System identifies the role. 4. System opens the home page for that role.
Alternate flow	2a. User selects "Forgot password": system sends a reset link.
Exception flow	2b. Wrong ID/password: system shows an error and allows retry. 2c. Three failed attempts: account is locked.
Postconditions	Success: user session started. Failure: no session, attempt logged
Business rules	Passwords are stored encrypted; session expires after 15 min of inactivity
Priority / frequency	High / many times a day

Conclusion: the use case diagram shows what each actor can do, and the template describes the Login use case step by step with its alternate and exception flows.

Q3. Give a list of various software quality attributes along with their brief description. Discuss the procedure for assessment of quality using these attributes. Justify the statement "Software Quality Assurance is an umbrella activity". (May 2026)

Software quality is the degree to which software meets the stated and implied requirements and is fit for use.

Quality attributes

Attribute	Meaning
Functionality	Provides the required functions correctly
Reliability	Performs without failure for a given time
Usability	Easy to learn and operate
Efficiency	Good use of time and memory
Maintainability	Easy to correct, modify and enhance
Portability	Works on different platforms
Security	Protects data from unauthorised access
Testability	Easy to test and verify
Reusability	Parts usable in other systems
Interoperability	Works with other systems
Correctness	Satisfies specification
Integrity	Control over unauthorised access

Procedure of quality assessment

1. Identify the **quality goals** and the relevant attributes from stakeholder needs.
2. Refine each attribute into **sub-attributes and measurable metrics** (e.g. reliability → MTBF, defect density).
3. Define **target values** and weights.
4. **Collect data** through reviews, inspections, testing, user surveys and defect reports.
5. **Compute and compare** the measured values with targets, and rate each attribute.
6. **Analyse** results (e.g. Pareto analysis, error index) and take corrective action.
7. **Repeat** for every release.



Quality assessment procedure

SQA is an umbrella activity

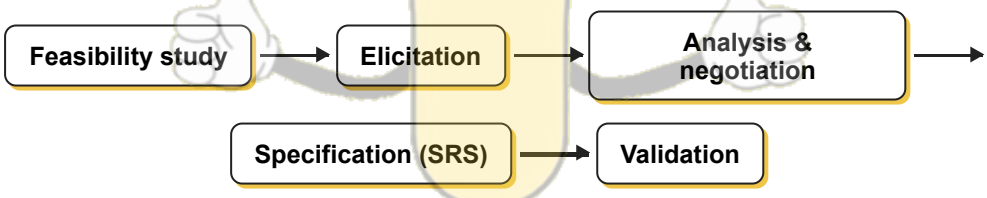
- An **umbrella activity** is applied throughout the process, not in one phase.

- SQA includes **formal technical reviews** of requirements, design and code, **testing strategy**, **standards enforcement**, **change control** and **measurement** in every phase.
- It follows the project from requirement gathering to maintenance, covering all the framework activities (communication, planning, modelling, construction, deployment).
- It checks both the process and the product, and it is independent of any single life cycle model.
- Hence, like configuration management and risk management, SQA covers the whole project like an umbrella.

Conclusion: quality is assessed by measuring the attributes against targets, and SQA is an umbrella activity because it is applied in every stage of development.

Q4. Discuss the Requirements Engineering Process. Write down the most generalized template used for describing Use Cases. Give description of its various components taking a suitable example. (May 2025, 2023)

Requirements Engineering (RE) is the systematic process of finding out, documenting and managing what the users need from a system. It produces the **Software Requirements Specification (SRS)**.



Requirements engineering process (with management of changes)

Stages

1. **Feasibility study** – checks technical, economic and operational viability and gives a go/no-go report.
2. **Requirements elicitation** – collecting needs by interviews, questionnaires, observation, brainstorming, use cases and prototyping.
3. **Requirements analysis and negotiation** – detect conflicts, incompleteness and ambiguity, and agree priorities (using DFD, ER and use case models).
4. **Requirements specification** – write the SRS clearly, with functional, non-functional and interface requirements.

5. **Requirements validation** – reviews, walkthroughs, prototyping and test-case design to ensure the requirements are correct, complete and consistent.
6. **Requirements management** – track changes and traceability throughout the project.

Use case template (generalised)

Component	Description
Name / ID	Short verb phrase and a unique ID
Actors	Users or systems that interact
Description	Goal in one or two lines
Preconditions	What must be true before it starts
Trigger	Event that starts the use case
Main (basic) flow	Numbered normal sequence of steps
Alternate flows	Valid variations
Exception flows	Error situations
Postconditions	State after success/failure
Business rules / NFRs	Constraints
Priority, frequency	Importance and how often

Example: "Withdraw cash" (ATM)

- **Actor:** Customer. **Precondition:** card is valid and the ATM has cash.
- **Main flow:** 1. Customer inserts card. 2. Enters PIN. 3. Chooses withdraw and enters amount. 4. System verifies balance. 5. Dispenses cash and prints receipt.
- **Alternate flow:** 3a. Customer chooses a fixed-amount button.
- **Exception flow:** 2a. Wrong PIN three times: card retained. 4a. Insufficient balance: error shown.
- **Postcondition:** account debited by the amount.

Conclusion: the RE process converts user needs into a validated SRS, and the use case template records the interaction between actors and system in a structured way.

Q5. Discuss all nine (9) types of UML diagrams under static and dynamic category which are used for specifying various concepts notations in Object Oriented Design/development of software. (May 2025)

UML (Unified Modeling Language) is a standard graphical language for specifying, visualising and documenting object-oriented software. Its nine classic diagrams fall into **static (structural)** and **dynamic (behavioural)** categories.

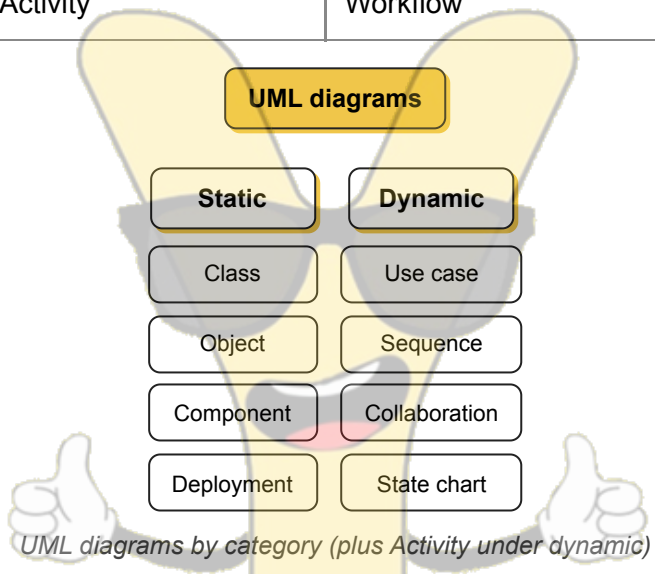
Static diagrams (structure of the system)

1. **Class diagram** – classes with attributes and methods, and relationships (association, aggregation, composition, inheritance); the main design diagram.
2. **Object diagram** – a snapshot of objects and their links at an instant.
3. **Component diagram** – software components (modules, libraries) and their interfaces and dependencies.
4. **Deployment diagram** – physical nodes (servers, devices) and the artifacts deployed on them.

Dynamic diagrams (behaviour of the system)

5. **Use case diagram** – actors and system functions from the user's view.
6. **Sequence diagram** – messages between objects arranged in time order (lifelines and activation bars).
7. **Collaboration (communication) diagram** – objects and numbered messages, focusing on object links.
8. **State chart (state machine) diagram** – the states of one object and the events causing transitions.
9. **Activity diagram** – the flow of activities, decisions, forks and joins, like an advanced flowchart; supports swimlanes.

Category	Diagram	Shows
Static	Class	Classes and relations
Static	Object	Instances at a time
Static	Component	Modular structure
Static	Deployment	Hardware mapping
Dynamic	Use case	Functions, actors
Dynamic	Sequence	Time-ordered messages
Dynamic	Collaboration	Object links and messages
Dynamic	State chart	Object life-cycle
Dynamic	Activity	Workflow



UML diagrams by category (plus Activity under dynamic)

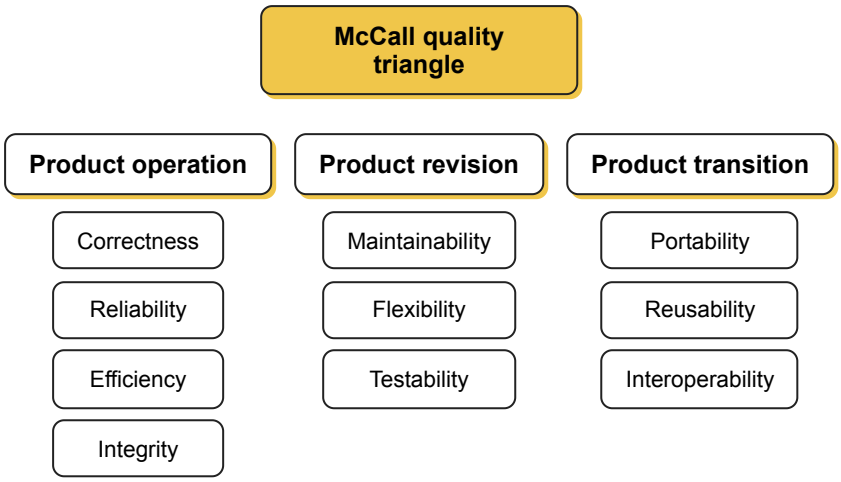
Use in development: use case diagrams capture requirements, class and object diagrams support design, sequence, collaboration, state and activity diagrams model behaviour, and component and deployment diagrams support implementation and installation.

Conclusion: static diagrams show what the system is made of, dynamic diagrams show how it behaves, and together they give a complete object-oriented model.

Q6. Discuss the various Mc Call's quality factors with quality triangle. (2023)

McCall's quality model (1977) describes software quality through **11 quality factors** grouped into **three categories** that form the **quality triangle**: product

operation, product revision and product transition. Each factor is measured through **quality criteria** and metrics.



McCall's quality factors

Product operation (how it runs)

- **Correctness** – does what the specification requires.
- **Reliability** – performs its function with required precision without failing.
- **Efficiency** – uses minimum computing resources (time, memory).
- **Integrity** – protects against unauthorised access or data change.
- **Usability** – effort needed to learn, operate and interpret output.

Product revision (how easily it is changed)

- **Maintainability** – effort to locate and fix errors.
- **Flexibility** – effort to modify an operating program.
- **Testability** – effort to test and ensure it performs its intended function.

Product transition (how well it adapts to a new environment)

- **Portability** – effort to move to another platform.
- **Reusability** – extent to which parts can be used in another application.
- **Interoperability** – effort to couple it with another system.

Quality criteria such as traceability, completeness, consistency, accuracy, modularity, simplicity, self-descriptiveness and communicativeness are checked for each factor; scores (0-10) are combined with weights to rate quality.

Conclusion: McCall's triangle organises the 11 factors under operation, revision and transition, showing quality from the user's and the maintainer's point of view.
