Module 3

Q1. "Use an ER diagram to design a hospital System."

Ans:

1. **Entities & key attributes**: Patient(PatientID, Name, DOB, Gender, Phone), Doctor(DoctorID, Name, Speciality, Phone), Appointment(AppID, Date, Time, Status), Department(DeptID, Name), Treatment(TreatID, Description, Cost), Invoice(InvoiceID, Date, Amount).
2. **Relationships**: Patient —(books)→ Appointment —(with)→ Doctor; Doctor —(works_in)→ Department; Patient —(receives)→ Treatment; Treatment —(billed_in)→ Invoice.
3. **Cardinality**: One Patient can have many Appointments (1:N); One Doctor handles many Appointments (1:N); One Appointment can include many Treatments (1:N or M:N via junction).
4. **Junction tables / weak relationships**: Use Patient_Treatment(PatientID, TreatID, Date, Notes) for M:N between Patient and Treatment; use Doctor_Schedule(DoctorID, Day, StartTime, EndTime) for availability.
5. **Simple ER diagram notation (text)**:
   - Patient (1) —< Appointment >— (N) Doctor
   - Doctor (N) —< Department (1)
   - Patient (N) —< Patient_Treatment >— (N) Treatment
   - Treatment (N) —< Invoice (1)

Q2. "Demonstrate a logical DFD FOR an ATM System."

Ans:

1. **External entities**: User/Customer, Bank Central System.
2. **Processes (level-0 logical)**: 1. Authenticate User, 2. Process Transaction (Withdraw/Deposit/Balance), 3. Update Account, 4. Print Receipt.
3. **Data stores**: D1: CardInfo, D2: AccountInfo, D3: TransactionLogs.
4. **Data flows**: Card + PIN → Authenticate User → AuthResult; Withdraw Request → Process Transaction → Dispense Cash + Update Account; Transaction details → TransactionLogs; Account update request → Bank Central System → Confirmation.
5. **Notes on logic**: Authentication checks CardInfo and AccountInfo; failed auth flows to Print Error and end; success flows to Process Transaction then Update Account and TransactionLogs.

Q3. "Illustrate the structure of a Software Requirement Specification (SRS)."

Ans:

1. **Header**: Title, version, authors, date, approvals.
2. **Introduction**: Purpose, scope, definitions, references, overview.
3. **Overall description**: Product perspective, user classes, constraints, assumptions.
4. **Specific requirements**: Functional requirements, non-functional requirements, external interfaces, data requirements, business rules.
5. **Appendices & traceability**: Glossary, use-case mappings, requirement IDs, change history, acceptance criteria.

Q4. "Apply functional and non-functional requirements to a banking System."
Ans:

1. **Functional req (examples)**: FR1: User login with 2FA; FR2: View account balance; FR3: Transfer funds between accounts; FR4: Generate monthly statements; FR5: Report lost card.
2. **Non-functional req (examples)**: NFR1: System uptime 99.9%, NFR2: Response time < 2s for balance enquiry, NFR3: Data encryption at rest and transit, NFR4: Support 10,000 concurrent users, NFR5: Audit logging for all transactions.
3. **Mapping table (simple)**: | FR ID | Description | Related NFRs |
   |---|---:|---|
   | FR2 | View balance | NFR2, NFR3 |
4. **Priority assignment**: High (security, transactions), Medium (reporting), Low (UI themes).
5. **Acceptance criteria**: For Transfer funds — success on valid accounts, rollback on failure, logs created, confirmation to user within 5s.

Q5. "Formulate a Complete SRS for an online exam portal."
Ans:

1. **Introduction & scope**: Portal for timed online exams, supports student registration, exam authoring, proctoring, grading, results. Version, stakeholders, constraints (web-based, mobile-friendly).
2. **Functional requirements (key)**: User auth (students, admins, invigilators), exam creation, question bank (MCQ/SAQ/LAQ), timed exam engine, auto-grading for objective Qs, manual grading, result publishing, certificate generation.
3. **Non-functional requirements (key)**: Availability 99.5%, response time <3s, secure login + anti-cheat measures, TLS encryption, support 5000 concurrent users.
4. **Data & interfaces**: Entities — User, Exam, Question, Attempt, Result; APIs for LMS integration, CSV import/export, payment gateway.
5. **Acceptance criteria & test cases**: Example — Exam timer must stop at time expiry and auto-submit; grading accuracy ≥99% for objective Qs; logs for each attempt, admin can audit.

Q6. "Design a data dictionary for a Student management System."
Ans:

1. **Entity: Student**: StudentID (PK): int, Name: varchar(100), DOB: date, Email: varchar(100), Phone: varchar(15), Address: text.
2. **Entity: Course**: CourseID (PK): int, CourseName: varchar(100), Credits: int, DeptID: int.
3. **Entity: Enrollment**: EnrollID (PK): int, StudentID (FK): int, CourseID (FK): int, EnrollDate: date, Grade: varchar(2).
4. **Entity: Faculty**: FacultyID: int, Name: varchar(100), DeptID: int, Email: varchar(100).
5. **Notes**: Data types, allowed values (e.g., Grade ∈ {A,B,C,D,F}), constraints (unique email), referential integrity rules (FK cascade/no action).

Q7. "Create use-cases for a hotel booking application."
Ans:

1. **Use-case: Search Rooms**
   - Actor: Guest; Pre: Home page open; Basic flow: enter dates → view available rooms → filter → view details; Post: list of available rooms shown.
2. **Use-case: Book Room**
   - Actor: Guest; Pre: Selected room; Flow: enter guest details → payment → confirm booking → send email; Post: Booking record created, confirmation sent.
3. **Use-case: Cancel Booking**
   - Actor: Guest/Admin; Pre: Valid booking; Flow: request cancel → check policy → process refund → update status; Post: Booking cancelled.
4. **Use-case: Manage Room (Admin)**
   - Actor: Admin; Flow: add/edit room types, prices, availability → save; Post: Room info updated.
5. **Use-case: Check-in/Check-out**
   - Actor: Receptionist; Flow: verify booking → ID check → allocate room → update status; Post: Guest checked in/out, billing updated.

Q8. "Create a clear method to separate user requirements from System requirements in a healthcare management System."
Ans:

1. **Definition step**: Write User Requirements (UR) in plain user language (who, what, why) and System Requirements (SR) as technical, testable specs.
2. **Template separation**: Use two sections in documents: UR (use-cases, user goals) and SR (functional specs, data models, APIs).
3. **Traceability matrix**: Map each UR to one or more SR IDs to show how the system meets user needs.
4. **Review with stakeholders**: Validate UR with doctors/nurses; translate accepted UR into SR by developers/architects.
5. **Sign-off & change control**: Users sign UR; any SR changes require mapping back and re-approval from user reps.

Q9. "Construct a logical DFD that models the core processes of an online ticket booking System."
Ans:

1. **External entities**: Customer, Payment Gateway, Event Provider.
2. **Core processes (level-0)**: 1. Search Events, 2. Select Seats, 3. Process Payment, 4. Issue Tickets.
3. **Data stores**: D1: EventCatalog, D2: BookingRecords, D3: PaymentLogs, D4: SeatInventory.
4. **Data flows**: Search query → Search Events → available list from EventCatalog; Seat selection → Select Seats → update SeatInventory; Payment details → Process Payment → PaymentGateway → confirmation → BookingRecords.
5. **Logic notes**: Seat locking during payment; timeout returns seats to inventory; ticket issued on payment success and email sent.

Q10. "Formulate a systematic approach for using data dictionary entries to validate the Consistency of requirement Specification."

Ans:

1. **Centralize dictionary**: Maintain single source of truth with entity names, attributes, types, defaults, constraints.
2. **Cross-check requirements**: For each requirement, ensure referenced entities/attributes match dictionary names and types exactly.
3. **Automated validation**: Use scripts to scan SRS for data terms and flag mismatches or undefined attributes.
4. **Version control & traceability**: Tag dictionary entries with version and link to requirement IDs that use them.
5. **Review cycles**: Periodic reviews with domain experts to confirm meanings, allowed values, and consistency across modules.

Q11. "Develop a feasible solution to represent data relationships using ER diagram for a library management System."

Ans:

1. **Entities & attrs**: Book(BookID, Title, ISBN, Publisher), Member(MemberID, Name, Email), Loan(LoanID, IssueDate, DueDate, ReturnDate), Author(AuthorID, Name), Category(CategoryID, Name).
2. **Relationships**: Book —(written_by)→ Author (M:N via Book_Author), Book —(belongs_to)→ Category (M:1), Member —(borrows)→ Loan —(for)→ Book.
3. **Cardinality**: Member can have many loans (1:N), Book can be loaned many times but only once per active loan (1:N), Book–Author M:N.
4. **Junction tables**: Book_Author(BookID, AuthorID), Book_Copy(CopyID, BookID, Shelf, Status) to manage multiple copies.
5. **Constraints & notes**: Enforce DueDate > IssueDate, Status ∈ {Available, Loaned, Reserved}, FK rules for referential integrity.

Q12. "Design a Structured interviewing technique that maximizes Stakeholder engagement for effective requirement elicitation."

Ans:

1. **Preparation**: Create agendas, send context and key questions beforehand, and define objectives for each interview.
2. **Use open and guided questions**: Start open for goals and problems, then narrow to specifics and examples.
3. **Active listening & validation**: Paraphrase answers back, confirm understanding, and record decisions.
4. **Visual aids**: Use mock-ups, process maps, and quick ER sketches to prompt feedback and keep focus.
5. **Follow-up & traceability**: Summarize minutes, circulate for sign-off, and map answers to requirement IDs for traceability.

Q13. "Propose a structured format for documenting functional requirements that enables traceability and impact analysis of changes."

Ans:

1. **Requirement template**: ReqID, Title, Description, Priority, Author, Date, Source (stakeholder), Acceptance Criteria, Related SR/UR IDs, Status.
2. **Traceability fields**: DependsOn (list of other ReqIDs), Impacts (modules or tests impacted), TestCaseIDs.
3. **Versioning & change history**: Track Version, ChangeDate, ChangedBy, Reason for Change.
4. **Link to design & code**: Fields for DesignDocRef and CodeModuleRef to enable impact analysis.
5. **Tooling suggestion**: Store requirements in a traceability tool or spreadsheet with filters for priority, owner, change status.

Q14. "Create a multi-user requirement gathering approach using viewpoint-oriented analysis."
Ans:

1. **Identify viewpoints**: Define distinct stakeholder views — EndUser, Admin, Operator, Security, Regulator.
2. **Collect per-viewpoint requirements**: Hold separate sessions to elicit goals, constraints, and scenarios for each viewpoint.
3. **Model viewpoints**: Create viewpoint-specific models (use-cases, data needs, UI sketches) and document conflicts.
4. **Integrate & resolve conflicts**: Use a consolidation workshop to merge viewpoints, prioritize, and resolve contradictions.
5. **Maintain mapping**: Keep a matrix linking requirements to originating viewpoint for future reference and accountability.

Q15. "Develop a framework to assess the technical feasibility of a Software project."
Ans:

1. **Criteria checklist**: Evaluate Technology fit, Skill availability, Performance limits, Integration complexity, Security & compliance.
2. **Proof of Concept (PoC)**: Build small PoC to test core risky components (APIs, scaling, third-party).
3. **Resource & timeline estimate**: Estimate hardware, software, team skills, and realistic schedule with buffers.
4. **Risk analysis & mitigation**: Identify top technical risks, likelihood & impact, and mitigation plans.
5. **Decision gate**: Use a feasibility scorecard (Pass/Conditional/Fail) and recommend go/no-go with defined conditions.

Q16. "Build a data dictionary template suitable for documenting all entities and attributes in an inventory System."
Ans:

1. **Template fields**: EntityName, AttributeName, DataType, Length, AllowedValues, DefaultValue, PK/FK, Nullable, Description, Constraints, Source.
2. **Example entry**: Entity: Item, Attribute: ItemCode, DataType: varchar, Length: 20, PK: Yes, Nullable: No, Description: Unique item identifier.

3. **Relationships section**: Entity: StockLocation, Relation: Item(ItemCode) → StockLocation(LocationID), Cardinality.
4. **Maintenance rules**: Owner, last updated date, and change log fields for each entry.
5. **Usage guidance**: Standard naming conventions, units for numeric fields, and data validation rules (e.g., Quantity >= 0).

Q17. "Design a change management process that protects SRS Integrity while adapting to evolving needs."
Ans:

1. **Change request intake**: All changes submitted as CR with ReqID, reason, impact description, and requester.
2. **Impact analysis**: Team assesses technical, schedule, cost, and test impacts and updates traceability matrix.
3. **Approval board**: Change Control Board (stakeholders + tech lead) approves, rejects, or defers CRs.
4. **Versioned updates**: Approved changes update SRS with new version, change log, and affected requirement links.
5. **Communication & regression tests**: Notify stakeholders, update related artifacts, and run regression test suite before deployment.

Q18. "Build a process for validating requirements to avoid conflicts and ensure Completeness in a financial application."
Ans:

1. **Requirement review workshops**: Cross-functional reviews (finance, compliance, dev, QA) to find gaps and conflicts.
2. **Consistency checks**: Use rule checks (terminology, data types, limits) and data dictionary cross-references.
3. **Traceability and mapping**: Map requirements to business rules, use-cases, and test cases to ensure coverage.
4. **Conflict resolution**: Log conflicts, prioritize by business impact, and resolve with stakeholder arbitration.
5. **Final validation**: Acceptance testing with real scenarios, sign-off from finance and security before moving to design.

Q19. "Develop a step-by-step method to evaluate the feasibility of a Software project focusing on Operational and Schedule aspects."
Ans:

1. **Operational feasibility**: Check if users can operate the system, training needs, support model, and process changes required.
2. **Resource assessment**: Verify availability of staff, infrastructure, and operational budgets needed for support and maintenance.
3. **Schedule estimation**: Break project into phases, use historical velocity or estimation techniques (story points) to get realistic timelines.

4. **Critical path & milestones**: Identify dependencies, critical path, and buffer time for key deliverables; prepare projected timeline.
5. **Go/no-go decision**: Score feasibility using readiness, resource fit, and schedule risk; recommend proceed, delay, or cancel.

Q20. "Propose a method to validate Software requirements using stakeholder walkthroughs and checklist-based reviews."
Ans:

1. **Prepare artifacts**: Distribute SRS, use-cases, and prototypes before the walkthrough with clear objectives.
2. **Walkthrough session**: Facilitator leads stakeholders through each requirement, asking for confirmations and examples.
3. **Checklist-based review**: Use checklists covering clarity, completeness, correctness, testability, traceability, and ambiguity.
4. **Record feedback & actions**: Capture comments as Defects/CRs, assign owners, and set deadlines for fixes.
5. **Closure & sign-off**: Re-run focused walkthrough on updated items and obtain formal sign-off from stakeholder representatives.

Q21. "Create a Complete quality management system covering quality Control, assurance, and improvement for Software projects."
Ans:

1. **Quality Assurance (QA)**: Define standards, processes, coding guidelines, peer reviews, and CI/CD pipeline with automated checks.
2. **Quality Control (QC)**: Test plans, unit/integration/system tests, acceptance tests, and defect tracking with SLAs for fixes.
3. **Continuous improvement**: Post-release reviews, root cause analysis for defects, and process updates based on retrospectives.
4. **Metrics & reporting**: Track defect density, test coverage, mean time to repair, release stability; report to stakeholders regularly.
5. **Governance & training**: Quality board to approve changes, regular training for team on best practices, and audits for compliance.

Q22. "Plan a method for updating legacy System using reuse and modern testing practices."
Ans:

1. **Assessment & modularization**: Analyze legacy code, identify reusable modules, and define boundaries for refactor or wrap.
2. **Strangler pattern**: Gradually replace parts by routing functionality to new services while keeping legacy running.
3. **Automated tests**: Create regression test suite (unit, integration, end-to-end) around legacy behavior before changes.
4. **CI/CD & sandboxing**: Use CI pipeline to run tests for every change, deploy to staging, and run acceptance tests with sample data.

5. **Rollback & monitoring**: Implement safe rollback plans, monitor metrics after each deployment, and iterate based on telemetry.

Module 4

Q1. Demonstrate unique test cases for login modules.
Ans:

Test Case ID	Test Scenario	Input Data	Expected Output	Type
TC01	Valid login credentials	Username: user1, Password: 12345	Login successful, redirect to dashboard	Positive
TC02	Invalid password	Username: user1, Password: wrongpass	Error message “Invalid Password”	Negative
TC03	Empty fields	Username: "", Password: ""	Display “Username and Password required”	Negative
TC04	SQL Injection attempt	Username: ' OR 1=1 --, Password: anything	Prevent login, show error “Invalid input”	Security
TC05	Case sensitivity check	Username: User1, Password: 12345	Show “Invalid Username” if system is case-sensitive	Functional

Q2. Apply Blackbox testing for an Ecommerce Cart.
Ans:

1. **Add Item Test**: Add product to cart → Expected: Item appears with correct price and quantity.
2. **Remove Item Test**: Remove an item → Expected: Item disappears and total updates correctly.
3. **Quantity Update Test**: Change quantity to 3 → Expected: Total = 3 × item price.
4. **Empty Cart Checkout**: Proceed with empty cart → Expected: Show “Cart is empty” message.
5. **Discount Application Test**: Apply valid coupon → Expected: Discount applied and total reduced accordingly.

Q3. Use equivalence partitioning to test date input.
Ans:

Input Range	**Example Input**	**Expected Result**
Valid dates	15/08/2024	Accepted
Invalid date format	2024-15-08	Rejected (format error)
Invalid date values	31/02/2024	Rejected (invalid date)
Empty input	""	Error “Date required”
Future date (if not allowed)	15/08/2030	Rejected

Explanation:

- Divide date inputs into valid and invalid partitions.
- Test one value from each partition to reduce test cases while maintaining good coverage.

Q4. Illustrate how validation testing is performed.
Ans:

1. **Definition**: Validation testing ensures that the software meets user needs and expectations.
2. **Process**:
   - Execute system tests to check real-world functionality.
   - Conduct user acceptance testing (UAT) with actual end-users.
   - Compare outputs against business requirements.
3. **Methods**: Functional testing, acceptance testing, integration testing.
4. **Tools used**: Selenium, JMeter, Postman, or manual UAT.
5. **Goal**: Confirm that “we built the right product” for user satisfaction.

Q5. Analyze the difference between Verification and Validation.
Ans:

Aspect	**Verification**	**Validation**
Definition	Checks if product is built correctly as per design/specs	Checks if product meets user needs
Phase	Done during development	Done after development (testing)
Focus	Process-oriented	Product-oriented
Methods	Reviews, inspections, walkthroughs	Functional, system, and acceptance testing
Example	Checking if login screen follows design specs	Checking if user can successfully log in

Q6. Apply regression testing after bug fixes.
Ans:

1. **Definition**: Regression testing ensures that recent code changes haven't broken existing functionality.
2. **Steps**:
   ○ Identify affected modules after a bug fix.
   ○ Re-run all relevant test cases that were previously passed.
   ○ Compare new results with previous expected outputs.
   ○ Automate repetitive tests using tools like Selenium or TestNG.
   ○ Document outcomes and verify system stability before release.
3. **Goal**: Confirm that bug fixes did not introduce new errors.

Q7. Distinguish between Alpha and Beta Testing.
Ans:

Feature	Alpha Testing	Beta Testing
Conducted by	Internal testing team	Actual end-users
Environment	Developer’s lab or test environment	Real user environment
Stage	Before Beta release	After Alpha testing
Purpose	Detect internal bugs and issues	Get real-world feedback and usability info
Example	Company testers check app features	Selected customers test pre-release version

Q8. Examine how quality standards affect Testing.
Ans:

1. **Consistent process**: Standards like ISO 9001 or IEEE 829 ensure all testing follows a defined, repeatable process.
2. **Better documentation**: Enforces structured test plans, cases, and reports that improve traceability.
3. **Improved reliability**: Ensures consistent test results and reduces defects through defined procedures.
4. **Higher customer trust**: Certified quality processes show commitment to delivering error-free software.
5. **Continuous improvement**: Encourages regular review and refinement of testing methods for long-term quality growth.

Module 5

Q1. Analyze the relationship between cohesion and coupling.
Ans:

1. **Cohesion measures how closely related the functions within a single module are. High cohesion means a module performs one specific task.**
2. **Coupling measures the degree of interdependence between modules. Low coupling means modules can work independently.**
3. **Relationship: As cohesion increases, coupling generally decreases, leading to better modularity.**
4. **Ideal system: High cohesion + Low coupling = easy to understand, test, and maintain.**
5. **Example: A module that only handles user login (high cohesion) and interacts with other modules through limited interfaces (low coupling) is well designed.**

Q2. Compare abstraction and information hiding as complexity reduction techniques.
Ans:

Aspect	Abstraction	Information Hiding
Definition	Shows essential features while ignoring details	Hides internal working from outside access
Goal	Simplify complex systems	Protect data integrity and prevent misuse
Implementation	Achieved through classes, functions, or APIs	Done using private variables and restricted access
Focus	What a system does	How the system does it
Example	“Print document” function hides printing steps	Hiding internal variables in a class using private keyword

Q3. Differentiate between architectural patterns and design patterns based on their scope.
Ans:

Aspect	Architectural Patterns	Design Patterns
Scope	High-level system structure	Component-level implementation
Focus	Defines overall system organization	Solves recurring code-level problems
Examples	Layered, Client-Server, MVC	Singleton, Factory, Observer
Used by	System architects	Software developers
Purpose	Guide framework and interaction of components	Improve code reusability and flexibility

Q4. Examine how good modularity contributes to achieving functional dependencies in a software system.
Ans:

1. **Definition: Modularity divides a system into smaller, manageable, and independent modules.**
2. **Functional dependency: Each module should perform a single, specific function (high cohesion).**
3. **Advantages: Makes debugging and testing easier, as each module can be tested independently.**
4. **Maintenance: Improves maintainability and reduces the risk of errors spreading.**
5. **Result: Promotes reusability and clear functional relationships between components.**

Q5. Classify seven types of cohesion based on their strength.
Ans:

Cohesion Type (Weak → Strong)	**Description**
1. Coincidental	**Randomly grouped elements; no relation.**
2. Logical	**Elements perform similar operations but not related logically.**
3. Temporal	**Elements executed at the same time (e.g., initialization).**
4. Procedural	**Elements form part of a sequence of operations.**
5. Communicational	**Elements operate on the same data set.**
6. Sequential	**Output of one element is input to the next.**
7. Functional	**All elements work together for a single well-defined task.**

Q6. Analyze the impact of type coupling on software maintainability.
Ans:

1. **Definition: Coupling refers to the degree of dependency between modules.**

2. **High coupling: Makes maintenance harder since changes in one module affect others.**
3. **Low coupling: Promotes independence and easier updates or replacements.**
4. **Types of coupling: Data, control, content, common, and stamp coupling.**
5. **Impact: Lower coupling improves flexibility, modular testing, and long-term software quality.**

Q7. Investigate the role of refinement in top-down design methodologies.
Ans:

1. **Definition: Refinement means breaking a complex problem into smaller, more detailed sub-problems.**
2. **Role in top-down design: Starts from the main function and gradually adds implementation details.**
3. **Benefits: Simplifies development, improves readability, and helps identify logical errors early.**
4. **Example: “Process order” → “Validate order” → “Check payment” → “Confirm delivery.”**
5. **Result: Structured, step-by-step design leading to maintainable and testable systems.**

Q8. Compare command line interfaces with graphical interfaces in terms of reusability.
Ans:

Aspect	**Command Line Interface (CLI)**	**Graphical User Interface (GUI)**
Reusability	**Easily scriptable and reusable in automation**	**Less reusable due to manual interactions**
User skill required	**Needs technical knowledge**	**Easy for non-technical users**
Customization	**Highly customizable with parameters**	**Limited to provided controls**
Speed	**Faster for experts**	**Slower due to visual navigation**
Example	**git commit -m "update"**	**Clicking commit button in Git GUI**

Q9. Categorize modern GUI elements based on their primary interaction functions.
Ans:

Interaction Function	**GUI Elements**
Input	**Text fields, checkboxes, radio buttons, file upload**
Navigation	**Menus, tabs, breadcrumbs, navigation bars**
Action	**Buttons, sliders, toggles, icons**
Output/Feedback	**Alerts, tooltips, progress bars, message boxes**
Display	**Tables, cards, images, charts, dashboards**

Q10. Examine the design considerations for effective error message implementation.
Ans:

1. **Clarity: Use simple, understandable language without technical jargon.**
2. **Actionable advice: Suggest how the user can fix the error.**
3. **Politeness: Keep tone neutral and helpful, not blaming the user.**
4. **Consistency: Use the same format and color scheme for all error messages.**
5. **Visibility: Display near the error location and highlight clearly for quick identification.**

Q11. Distinguish between procedural and object-oriented programming paradigms.
Ans:

Aspect	Procedural Programming	Object-Oriented Programming (OOP)
Focus	Function-based	Object and class-based
Data handling	Data and functions are separate	Data and functions are combined (encapsulation)
Reusability	Code reused through functions	Code reused through inheritance and polymorphism
Example languages	C, Pascal	Java, Python, C++
Best suited for	Simple, sequential logic	Complex systems with multiple entities

Q12. Analyze the criteria for selecting appropriate programming languages for specific projects.
Ans:

1. **Project requirements: Choose based on performance, platform, and problem type.**
2. **Team expertise: Select languages familiar to the development team.**
3. **Community support: Languages with strong libraries and communities save development time.**
4. **Performance needs: Use C++ or Rust for speed; Python or JavaScript for flexibility.**
5. **Integration capability: Check compatibility with other tools, databases, and APIs.**

Q13. Break down the key components that constitute good programming practices.
Ans:

1. **Readable code: Use meaningful names, comments, and consistent indentation.**
2. **Modular design: Divide code into small, reusable functions or classes.**
3. **Error handling: Anticipate and manage runtime errors gracefully.**
4. **Version control: Use Git or similar tools for collaboration and tracking changes.**
5. **Testing & documentation: Write unit tests and maintain updated documentation.**

Q14. Compare different coding standards in terms of their benefits for team development.
Ans:

Aspect	Consistent Coding Standard (e.g., PEP8, Google Style)	No Standard
Readability	Improves readability across team	Code becomes inconsistent
Collaboration	Easier for multiple developers to work together	Hard to merge changes
Error reduction	Encourages best practices and avoids bugs	Increases chances of mistakes
Maintenance	Easier long-term maintenance	Difficult for new members to understand
Example	PEP8 for Python ensures uniform style and clear structure	Unstructured code lacks readability

Q15. Investigate the relationship between user-interface design principles and GUI control selection.
Ans:

1. **Consistency: Use similar controls for similar actions (e.g., "Save" button always in same place).**
2. **User familiarity: Choose controls users recognize easily, like dropdowns for multiple options.**
3. **Feedback: Controls should provide instant feedback — buttons highlight or change color when clicked.**
4. **Accessibility: Use large buttons, readable fonts, and keyboard navigation for inclusiveness.**
5. **Aesthetic balance: Maintain clean layouts, proper spacing, and visual hierarchy to guide user focus effectively.**

Q1. Explain how ISO 9000 standards contribute to software quality improvements.
Ans:

1. **Standardized process: ISO 9000 provides guidelines for consistent software development and documentation.**

2. **Focus on quality management: Emphasizes process control, quality assurance, and continuous improvement.**
3. **Customer satisfaction: Ensures software meets user expectations through quality planning and regular audits.**
4. **Error reduction: Encourages early detection and correction of defects to minimize rework.**
5. **Continuous improvement: Promotes regular reviews and updates in processes for better long-term quality outcomes.**

Q2. Describe the five levels of the Capability Maturity Model (CMM) and their importance in software process improvement.
Ans:

Level	Name	Description & Importance
1	Initial	Unstructured, unpredictable processes; success depends on individuals.
2	Repeatable	Basic project management established; projects can be repeated with similar success.
3	Defined	Standardized and documented processes for the entire organization.
4	Managed	Quantitative metrics used to measure performance and quality.
5	Optimizing	Continuous process improvement through feedback and innovation.

Importance: Helps organizations assess maturity, reduce risk, improve productivity, and achieve consistent software quality.

Q3. Explain the need for proper software project management to avoid project failure.
Ans:

1. **Clear planning: Defines goals, timelines, and deliverables to prevent confusion.**
2. **Resource allocation: Ensures proper use of time, budget, and workforce.**
3. **Risk control: Identifies and mitigates potential problems before they affect outcomes.**
4. **Team coordination: Promotes collaboration and communication among members.**
5. **Quality assurance: Maintains software standards and prevents defects that cause delays or rework.**

Q4. Explain how the size of software is estimated or calculated during project planning.
Ans:

1. **Lines of Code (LOC): Count the total number of executable lines written in the source code.**
2. **Function Point Analysis (FPA): Measures functionality provided to the user (inputs, outputs, files, etc.).**
3. **Use Case Points: Estimates based on the number and complexity of use cases.**
4. **Expert judgment: Uses experience of project managers and developers.**
5. **Purpose: Helps estimate cost, time, manpower, and project feasibility.**

Q5. Summarize the steps involved in 'risk mitigation, monitoring, and management' plan.
Ans:

1. **Risk identification: List all possible risks that could affect project outcomes.**
2. **Risk analysis: Evaluate each risk by probability and impact level.**
3. **Risk mitigation: Develop strategies to reduce or eliminate high-risk items.**
4. **Monitoring: Continuously track risk indicators and status throughout the project.**
5. **Management & reporting: Take corrective action when risks arise and update stakeholders regularly.**

Q6. Explain how metric analysis helps track the progress and performance of a software project.
Ans:

1. **Measurement of performance: Uses metrics like defect density, velocity, and productivity to measure progress.**
2. **Schedule tracking: Compares planned versus actual progress using milestone metrics.**
3. **Quality monitoring: Tracks errors, test coverage, and code reviews to maintain standards.**
4. **Team efficiency: Identifies areas where productivity can be improved.**
5. **Decision-making: Provides quantitative data for management to make informed project adjustments.**

Q7. Describe the components of a network scheduling diagram and its use in project planning.
Ans:

1. **Activities: Represent project tasks or operations.**
2. **Nodes: Represent events or milestones in the project.**
3. **Arrows: Show dependencies and task sequences.**
4. **Critical Path: The longest path determining project duration.**
5. **Use: Helps visualize workflow, identify task dependencies, and manage timelines effectively using tools like PERT or CPM.**

Q8. Difference between Function Point Analysis (FPA) and Lines of Code (LOC) estimation method.
Ans:

Aspect	Function Point Analysis (FPA)	Lines of Code (LOC)
Measurement type	Measures software functionality	Measures physical code size
Language dependency	Independent of programming language	Dependent on language
Focus	User requirements and features	Developer effort
Use case	Useful in early design phase	Useful after code completion
Accuracy	More accurate for large systems	Less accurate due to coding style variations

Q9. Explain the difference between verification and validation in software testing with a suitable example.
Ans:

Aspect	Verification	Validation
Definition	Ensures product is built correctly as per design	Ensures correct product is built as per user needs
Focus	Process and documentation	Functionality and output
Stage	During development	After development
Example	Checking design documents against specifications	Testing login feature to ensure user can log in successfully
Outcome	Detects logical/design errors	Ensures real-world usability

Q10. Describe the significance of quality assurance in the software development life cycle.
Ans:

1. **Error prevention: Detects issues early during the development phase.**
2. **Process control: Monitors adherence to standards and procedures.**
3. **Customer confidence: Ensures product reliability and usability.**
4. **Cost reduction: Early issue detection minimizes rework costs.**
5. **Continuous improvement: Encourages feedback and refinement of development processes.**

Q11. Design a Gantt Chart for a software project.
Ans:

Task	Start Date	End Date	Duration (Days)	Dependencies
Requirement Analysis	01-Jan	05-Jan	5	-
Design Phase	06-Jan	12-Jan	7	Requirement Analysis
Development	13-Jan	25-Jan	13	Design Phase
Testing	26-Jan	31-Jan	6	Development
Deployment	01-Feb	03-Feb	3	Testing

Use: Shows task timeline, dependencies, and progress visually to manage schedule and resources effectively.

Q12. Formulate a risk management plan for an e-learning platform.
Ans:

1. Risk Identification:
   - Server downtime
   - Data breaches or unauthorized access
   - Low student engagement
   - Payment gateway failure
2. Risk Analysis:
   - Server downtime – High probability, High impact
   - Data breach – Medium probability, Critical impact
3. Mitigation Strategy:
   - Use backup servers and cloud redundancy
   - Implement SSL encryption and strong authentication
   - Add gamification to increase engagement
   - Integrate multiple secure payment gateways
4. Monitoring:
   - Daily server health checks and real-time alerts
   - Regular audits and penetration tests
5. Contingency Plan:
   - Activate backup server during downtime
   - Notify users of delays with alternative access routes

Q13. Describe the purpose of regression testing and when it is applied in project testing.
Ans:

1. **Definition: Regression testing ensures that new code changes or bug fixes do not break existing features.**
2. **When applied:**
   - **After bug fixes or code enhancements**
   - **During maintenance or version upgrades**
3. **Purpose:**
   - **To confirm overall system stability after modifications.**
   - **To prevent reintroduction of old defects.**
4. **Method: Run previously passed test cases on the updated build.**
5. **Result: Guarantees software consistency, reliability, and smooth functionality after every update.**

Q1. Design a comprehensive testing framework for software quality assurance using verification and validation principles.
Ans:

1. **Verification Phase (ensures software is built correctly):**
   - ***Reviews & Inspections*: Check requirements, design documents, and test cases.**
   - ***Static Testing*: Perform code walkthroughs and peer reviews to detect design or logic flaws early.**
   - ***Checklists*: Ensure compliance with coding and documentation standards.**
2. **Validation Phase (ensures correct product is built):**
   - ***Dynamic Testing*: Execute functional, integration, and system tests.**
   - ***User Acceptance Testing (UAT)*: Validate software against user expectations and real-world scenarios.**
3. **Testing Levels:**
   - **Unit Testing → Integration Testing → System Testing → Acceptance Testing.**
4. **Quality Metrics: Track defect density, test coverage, and requirement traceability.**
5. **Automation & Reporting: Integrate automated testing (Selenium, JUnit) and continuous feedback via dashboards to ensure ongoing quality control.**

Q2. Develop a complete project plan for a restaurant POS (Point of Sale) system (scope, cost, schedule, risk plan).
Ans:

1. **Project Scope:**
   - **Develop a POS system for restaurant billing, inventory, and order tracking.**
   - **Modules: Order Management, Billing, Inventory, Reporting, Staff Management, Payment Gateway.**
2. **Cost Estimate:**
   - **Hardware: ₹80,000 (POS terminals, printers, scanners)**
   - **Software Development: ₹2,50,000**
   - **Testing & Deployment: ₹70,000**
   - **Maintenance & Support (6 months): ₹30,000**

- Total Estimated Cost: ₹4,30,000

3. **Project Schedule:**

Phase	Duration	Start	End
Requirement Gathering	5 days	01-Mar	05-Mar
System Design	7 days	06-Mar	12-Mar
Development	20 days	13-Mar	02-Apr
Testing	10 days	03-Apr	12-Apr
Deployment & Training	5 days	13-Apr	18-Apr

1.
2. **Risk Plan:**
   - *Risk 1*: Hardware compatibility → Mitigation: Pre-testing with POS devices.
   - *Risk 2*: Data loss → Mitigation: Automated daily backups.
   - *Risk 3*: Staff adaptation issues → Mitigation: Provide training sessions.
   - *Risk 4*: Internet failure → Mitigation: Offline mode for POS transactions.

Q3. Create a cost estimation model using Function Point Analysis (FPA) technique.

Ans:

1. **Identify Components:**
   - External Inputs (EI): 10
   - External Outputs (EO): 8
   - External Inquiries (EQ): 5
   - Internal Logical Files (ILF): 6
   - External Interface Files (EIF): 3
2. **Assign Weights (Average complexity):**
   - EI: 10×4 = 40
   - EO: 8×5 = 40
   - EQ: 5×4 = 20
   - ILF: 6×10 = 60
   - EIF: 3×7 = 21
   - Total Unadjusted Function Points (UFP) = 181
3. **Value Adjustment Factor (VAF):**
   - Based on 14 general system characteristics (e.g., performance, complexity).
   - Suppose VAF = 1.10
4. **Adjusted Function Points (AFP) = UFP × VAF = 181 × 1.10 = 199.1 ≈ 199 FP**
5. **Cost Estimation:**

- If cost per FP = ₹1000 → Total Cost = 199 × 1000 = ₹1,99,000

Q4. Construct a comprehensive risk management plan.
Ans:

1. **Risk Identification:** List potential risks — technical, operational, financial, schedule-related.
2. **Risk Analysis:** Assess each risk by *Probability (P)* and *Impact (I)* on a scale of 1–5.
3. **Risk Matrix:**

Risk	Probability	Impact	Risk Score (P×I)	Priority
Server Failure	4	5	20	High
Schedule Delay	3	4	12	Medium
Data Breach	2	5	10	High

1.
2. **Mitigation Strategies:**
   - Backup servers for redundancy.
   - Implement strict code version control.
   - Conduct regular security audits.
3. **Monitoring & Review:**
   - Weekly status reviews, real-time dashboards, and automated alerts for high-risk triggers.

Q5. Design a framework using RMMM (Risk Mitigation, Monitoring, and Management) methodology for software projects.
Ans:

1. **Risk Mitigation:**
   - Preventive actions to reduce the likelihood of risks (e.g., training, code review, backup).
2. **Risk Monitoring:**
   - Track potential risks during project execution through regular meetings and reports.
3. **Risk Management:**
   - Plan for handling risk events when they occur (contingency plans).
4. **Framework Steps:**
   - Identify → Assess → Mitigate → Monitor → Manage → Review.
5. **Example:**
   - Risk: Server crash → Mitigation: Backup server → Monitoring: Log uptime → Management: Switch to backup automatically.

Q6. Propose a complete test case methodology for achieving maximum defect detection.
Ans:

1. **Requirement Analysis: Understand functional and non-functional requirements.**
2. **Test Planning: Define scope, objectives, and testing strategy (manual + automated).**
3. **Test Case Design:**
   - **Use Blackbox techniques: Boundary Value Analysis, Equivalence Partitioning.**
   - **Use Whitebox techniques: Code coverage, control flow.**
4. **Execution & Logging: Run tests systematically and record all defects with severity and status.**
5. **Review & Regression Testing: Retest after bug fixes to ensure stability.**
6. **Defect Metrics: Track defect density, test coverage, and defect leakage rate to evaluate testing effectiveness.**

Q7. Synthesize a project scheduling approach using network scheduling techniques for software development.
Ans:

1. **List project activities with start/end dependencies.**
2. **Create activity relationships using nodes and arrows (PERT/CPM diagram).**
3. **Estimate durations for each task and identify earliest and latest start times.**
4. **Determine critical path — the longest path that defines total project duration.**
5. **Apply slack time to optimize non-critical activities.**
6. **Example Activities:**

Activity	Description	Duration (Days)	Depends On
A	Requirement Gathering	4	-
B	Design	6	A
C	Coding	10	B
D	Testing	5	C
E	Deployment	3	D

1.
   - **Critical Path: A → B → C → D → E = 28 days.**

Q8. Build an organizational process maturity implementation plan for software quality improvement.
Ans:

1. **Assessment: Evaluate current processes using CMM or ISO maturity criteria.**
2. **Goal Setting: Define quality objectives such as defect reduction and faster delivery.**
3. **Process Definition: Create standardized development, testing, and review procedures.**
4. **Training & Awareness: Conduct workshops for developers and QA teams.**
5. **Measurement & Improvement: Use metrics (defect rate, cycle time) to track performance and refine processes continuously.**

Q9. Compose an Earned Value Analysis (EVA) system plan and real-time project performance measurement strategy for evaluating effectiveness of people management in software project teams.

Ans:

1. **EVA Components:**
   - **Planned Value (PV): Budgeted cost for scheduled tasks.**
   - **Earned Value (EV): Value of work actually completed.**
   - **Actual Cost (AC): Real cost spent for completed work.**
2. **Performance Indicators:**
   - **Cost Performance Index (CPI) = EV / AC → >1 = under budget.**
   - **Schedule Performance Index (SPI) = EV / PV → >1 = ahead of schedule.**
3. **People Management Metrics:**
   - **Productivity per developer (EV/Person-Hour).**
   - **Defect rate per team member.**
   - **Team utilization and overtime ratio.**
4. **Real-time Monitoring:**
   - **Integrate EVA dashboards (like Jira or MS Project) to display team progress.**
   - **Conduct weekly review meetings to evaluate team efficiency and project health.**
5. **Decision Support:**
   - **CPI < 0.9 or SPI < 0.9 → triggers corrective action like resource reallocation or schedule revision.**