CBCS SCHEME

BCS501

Fifth Semester B.E./B.Tech. Degree Examination, Dec.2025/Jan.2026 Software Engineering and Project Management

Time: 3 hrs. Max. Marks: 100

Note:

1. Answer any FIVE full questions, choosing ONE full question from each module.

2. M: Marks, L: Bloom's level, C: Course outcomes. Module1 M L C

Q.1 a. Explain the domains of software applications. 08 L2 CO1

b. Outline the unique nature of WebApps. 08 L2 C01

c. Explain various software myths. Discuss. 04 L2 C01

OR

Q.2 a. Explain the activities performed in a software process framework? 06 L2 CO1

b Explain the waterfall model along with its pros and cons. 08 L2 CO1

c. Explain specialized process models. 06 L2 C01

Module-2

03 2. Explain how groundwork parameters are established in requirements engineering 08 L2 CO2

b. What is the importance of quality function deployment in eliciting requirements? How can we validate requirements? 06 L1 CO2

OR

Q.4 a. Explain about scenario based modelling. 10 L2 CO2

b. Illustrate regarding how can we create a Behavioral Model. 10 L2 CO2

Module-3

Q.5 a Explain Agility along with the principles of Agility. 10 L2 CO3

b Explain the Extreme Programming Process. 06 L2 CO3 c Explain about the critics of XP. 04 L2 CO3

OR

Q.6 a. Explain the scrum flow process, 08 م L2 CO3

b. Explain the communication principles guiding framework activity. 08 L2 CO3

c. How can we validate and test principles in coding. 04 L1 CO3

Module -4

Q.7 a. Define Project. Show the contrast of software projects with other types of projects. 06 L2 CO4

b Explain the ISO 12207 software development life cycle with a neat diagram. 10 L2 CO4

OR

Q.8 a. Illustrate the cost benefit evaluation techniques. 10 L2 CO4

b. Illustrate the concept of Risk evaluation. 10 L2 CO4

Module -5

Q.9 a. Explain the details to be drafted for achieving quality in software. 06 L2 CO5

b. Explain the software quality characteristics of ISO 9126. 08 L2 CO5

c. Explain process requirements for the process quality management. 06 L2 CO5

OR

Q10 a. Explain about the decomposition techniques. 10 L2 CO5

b. Explain the COCOMO II model. 10 L2 CO5

SOLUTION

1.

a) Explain the domains of software applications

Software applications can be classified into different **domains** based on their purpose and usage. The major domains are:

1. System Software

System software manages and controls computer hardware and provides a platform for application software.

- Examples: Operating systems (Windows, Linux), device drivers, compilers, editors.
- Characteristics: High performance, close interaction with hardware.

2. Application Software

These programs solve specific user problems or perform business tasks.

- Examples: Payroll systems, banking software, inventory management systems.
- Characteristics: User-oriented, data-intensive.

3. Engineering and Scientific Software

Used for scientific research and engineering analysis.

- Examples: Simulation software, CAD/CAM systems, numerical computation programs.
- Characteristics: Complex algorithms, high precision, heavy computation.

4. Embedded Software

Software embedded within hardware products to control their functionality.

- Examples: Software in washing machines, automobiles, medical devices.
- Characteristics: Real-time constraints, reliability and safety critical.

5. Product-Line Software

Designed to serve a specific market or industry.

- Examples: Hotel management software, billing systems, ERP solutions.
- Characteristics: Reusable components, customizable features.

6. Web Applications (WebApps)

Applications that run on web browsers over the internet.

- Examples: E-commerce websites, social networking sites, online learning platforms.
- Characteristics: Network-centric, dynamic content.

7. Artificial Intelligence Software

Software that exhibits human-like intelligence.

- Examples: Expert systems, machine learning applications, chatbots.
- Characteristics: Knowledge-based, adaptive, uses heuristics.

b) Outline the unique nature of WebApps

Web Applications (WebApps) have several **unique characteristics** that differentiate them from conventional software systems:

1. **Network-Intensive**
   WebApps operate over the internet or intranet and depend heavily on network availability and performance.
2. **Content-Driven**
   They combine text, graphics, audio, video, and animations along with program logic.
3. **Continuous Evolution**
   WebApps are frequently updated to add new features, fix bugs, and improve user experience.
4. **Immediacy**
   Changes made to WebApps are immediately visible to users without the need for manual installation.
5. **Security Sensitive**
   WebApps are exposed to security threats such as hacking, data breaches, and unauthorized access.
6. **Scalability**
   WebApps must support a large number of concurrent users and handle varying workloads.
7. **Cross-Platform Compatibility**
   They should run on different browsers, devices, and operating systems.

8. **Short Development Cycles**
   WebApps often follow rapid development and deployment cycles.

c) Explain various software myths. Discuss.

Software myths are false beliefs that lead to poor decision-making and project failure. They are broadly classified into **customer myths, management myths, and practitioner myths**.

1. Customer Myths

- *Myth:* A general statement of objectives is sufficient to start coding.
  Reality: Detailed and clear requirements are essential.

- *Myth:* Changes are easy to accommodate because software is flexible.
  Reality: Changes can be costly if not managed properly.

2. Management Myths

- *Myth:* Adding more programmers will speed up a delayed project.
  Reality: According to Brooks' Law, it may further delay the project.

- *Myth:* New tools and technologies will automatically solve problems.
  Reality: Skilled people and proper processes are more important.

3. Practitioner (Developer) Myths

- *Myth:* Once the program works, the job is done.
  Reality: Maintenance and evolution consume most of the software life cycle.

- *Myth:* Quality can be tested in at the end.
  Reality: Quality must be built in throughout the development process.

Q.2 (a) Explain the activities performed in a software process framework

A **software process framework** defines a set of **framework activities** that are applicable to all software projects, regardless of size or complexity.

The main framework activities are:

1. Communication

- Involves interaction between stakeholders and developers.

- Requirements are gathered, clarified, and documented.

- Techniques: interviews, meetings, questionnaires.

- Output: Software Requirement Specification (SRS).

2. Planning

- Establishes project scope, resources, schedule, and cost.
- Risk analysis and estimation are performed.
- Output: Project plan, schedule, and milestones.

3. Modeling

- Converts requirements into design representations.
- Includes:
    - **Analysis modeling** (data, functional, behavioral models)
    - **Design modeling** (architecture, interface, component design)
- Helps in understanding how the system will be built.

4. Construction

- Actual development of software.
- Includes:
    - Coding
    - Unit testing
    - Integration testing
- Ensures that design is correctly implemented.

5. Deployment

- Software is delivered to the customer.
- Includes installation, training, support, and maintenance.
- Customer feedback is collected for future enhancements.

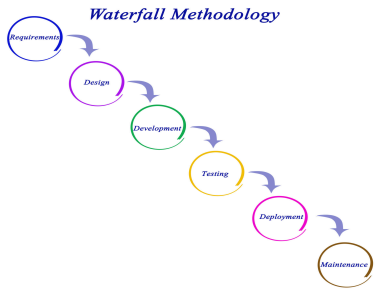
Q.2 (b) Explain the waterfall model along with its pros and cons

Waterfall Model

The **Waterfall model** is a **linear and sequential software development model**, where each phase must be completed before moving to the next phase.

Phases of the Waterfall Model:

1. **Requirements Analysis**
2. **System Design**
3. **Implementation (Coding)**
4. **Testing**
5. **Deployment**
6. **Maintenance**



Advantages of the Waterfall Model:

- **Simple and easy to understand.**
- **Well-defined stages and documentation.**
- **Suitable for small projects with stable requirements.**
- **Easy to manage due to clear milestones.**

Disadvantages of the Waterfall Model:

- **No flexibility for requirement changes.**
- **Testing is done late in the cycle.**

- **High risk and uncertainty.**
- **Not suitable for large or complex projects.**
- **Customer feedback comes very late.**

Q.2 (c) Explain specialized process models

Specialized process models are used when standard process models are not suitable due to specific project constraints or requirements.

1. Component-Based Development Model

- **Software is built by integrating reusable components.**
- **Reduces development time and cost.**
- **Requires a well-maintained component repository.**

2. Formal Methods Model

- **Uses mathematical techniques to specify, develop, and verify software.**
- **Ensures high reliability and correctness.**
- **Used in safety-critical systems (avionics, nuclear systems).**
- **Complex and expensive.**

3. Aspect-Oriented Software Development (AOSD)

- **Focuses on separating cross-cutting concerns such as security and logging.**
- **Improves modularity and maintainability.**
- **Aspects are woven into core functionality.**

4. Concurrent Development Model

- **Activities are performed in parallel rather than sequentially.**
- **Each activity exists in different states (waiting, under development, under review).**

- **Suitable for real-time and large projects.**

Q.3 (a) Explain how groundwork parameters are established in Requirements Engineering (08 marks)

Requirements Engineering (RE) begins by establishing **groundwork parameters**, which define the **problem scope, context, and constraints** of the software to be developed.

The groundwork parameters are established through the following steps:

1. Identification of Stakeholders

- Identify all parties affected by the system (users, customers, managers, developers).
- Helps in understanding different viewpoints and expectations.

2. Understanding the Business Context

- Analyze the organization's goals, policies, and workflows.
- Determine how the software will support business objectives.

3. Defining System Scope

- Clearly state what the system will do and what it will not do.
- Prevents scope creep and unrealistic expectations.

4. Eliciting Initial Requirements

- Collect preliminary functional and non-functional requirements.
- Techniques include interviews, questionnaires, workshops, and observation.

5. Identifying Constraints

- Determine limitations such as budget, schedule, technology, legal, and regulatory constraints.
- Influences design and implementation decisions.

6. Understanding the Operating Environment

- Identify hardware, software, network, and platform requirements.

- Includes performance, security, and reliability needs.

7. Feasibility Analysis

- Evaluate technical, economic, and operational feasibility.
- Ensures the project is realistic and achievable.

8. Establishing Acceptance Criteria

- Define conditions under which the system will be accepted by stakeholders.
- Provides a basis for validation and testing.

Q.3 (b) What is the importance of Quality Function Deployment (QFD) in eliciting requirements? (06 marks)

Quality Function Deployment (QFD) is a structured technique used to **translate customer needs into technical requirements**.

Importance of QFD in Requirements Elicitation:

1. **Customer-Focused Approach**

- Captures the *voice of the customer* accurately.
- Ensures customer needs are not misunderstood.

2. **Improves Requirement Clarity**

- Converts vague customer expectations into measurable technical specifications.

3. **Prioritization of Requirements**

- Helps rank requirements based on customer importance.
- Focuses development effort on critical features.

4. **Reduces Miscommunication**

- Acts as a common communication platform for customers and developers.

5. **Early Detection of Conflicts**

- Identifies conflicting requirements at an early stage.

6. **Improves Product Quality**

- Aligns design decisions with customer expectations, leading to higher satisfaction.

Q.3 (c) How can we validate requirements? (06 marks)

Requirements validation ensures that the documented requirements correctly reflect customer needs and are suitable for implementation.

Techniques used for Requirements Validation:

1. Requirements Reviews

- Formal or informal reviews involving stakeholders.
- Check for correctness, completeness, consistency, and clarity.

2. Prototyping

- Develop a working or mock-up model of the system.
- Helps users visualize requirements and provide feedback.

3. Test-Case Generation

- Create test cases from requirements.
- Ensures requirements are testable and verifiable.

4. Model Validation

- Validate analysis models such as use case diagrams and data flow diagrams.
- Ensures logical correctness.

5. Traceability Analysis

- Establish traceability between requirements, design, and test cases.
- Ensures no requirement is missing or extra.

6. Acceptance Criteria Definition

- Confirm that each requirement has clear acceptance conditions.
- Supports final system acceptance.

Q.4 (a) Explain about Scenario-Based Modelling (10 marks)

Scenario-based modelling is a requirements analysis technique that describes **how users interact with the system** under different situations. It focuses on *what the system should do from the user's point of view*.

Key Concepts of Scenario-Based Modelling

1. **Scenario**

- A narrative description of a specific user interaction with the system.
- Describes a sequence of actions and system responses.

2. **Use Cases**

- The most common form of scenario-based modelling.
- Describes a set of interactions between an **actor** and the system to achieve a goal.

3. **Actors**

- External entities that interact with the system (users, hardware, or other systems).

Elements of a Use Case

- **Use Case Name**
- **Actors**
- **Description**
- **Preconditions**
- **Main Flow of Events**
- **Alternative / Exception Flows**
- **Postconditions**

Types of Scenarios

1. **Primary Scenario**

- Describes the normal flow of events (successful execution).

2. **Alternative Scenario**

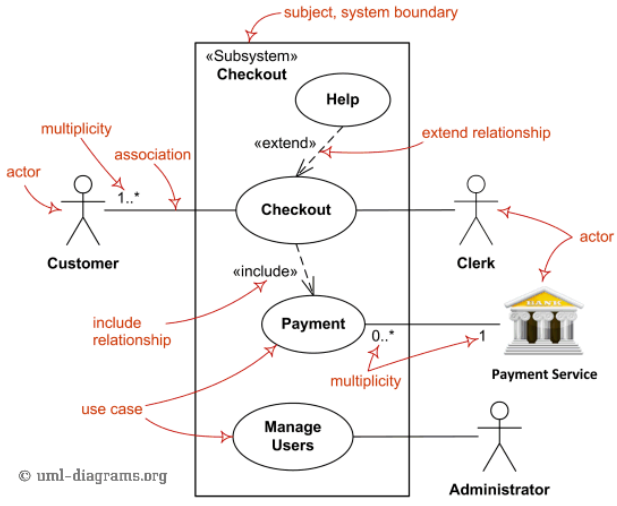
- Describes variations or optional flows.

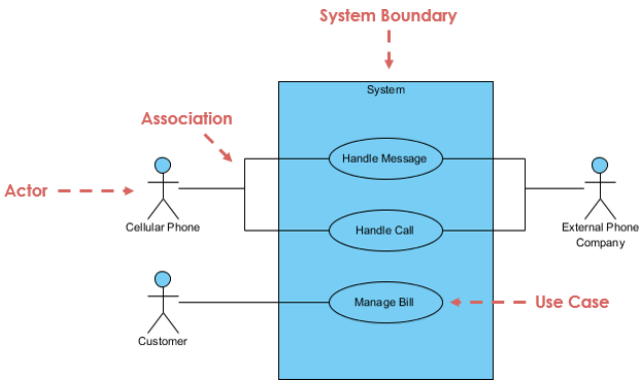
3. **Exception Scenario**

- Handles error conditions or abnormal behavior.

Representation Techniques

- **Use Case Diagrams (UML)**
- **Textual Use Case Descriptions**
- **User Stories**





Advantages of Scenario-Based Modelling

- Easy for stakeholders to understand.
- Improves communication between users and developers.
- Helps in identifying missing requirements.
- Serves as a basis for test case development.

Limitations

- Does not describe internal system behavior.
- Large systems may result in many use cases.
- Needs to be complemented with other models.

Q.4 (b) Illustrate how we can create a Behavioral Model (10 marks)

A **behavioral model** describes **how the system behaves over time** in response to events. It focuses on **dynamic aspects** of the system.

Steps to Create a Behavioral Model

1. Identify Use Cases

- Start with scenario-based models.
- Identify events that trigger system behavior.

2. Identify Events

- Events may be:
    - External (user input)
    - Internal (system-generated)
    - Temporal (time-based)

3. Define States

- A state represents a condition in which the system exists.
- Example: *Idle, Processing, Completed*.

4. Determine State Transitions

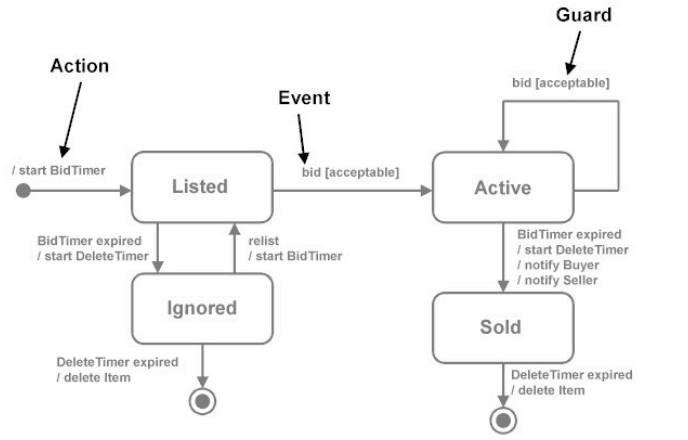
- Transitions define how the system moves from one state to another.
- Each transition is triggered by an event.

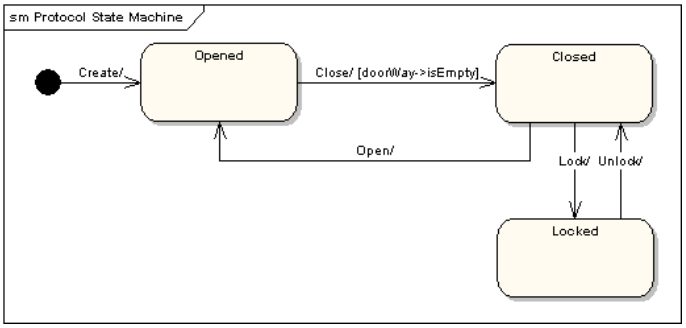
5. Define Actions

- Specify actions performed during transitions or within states.

6. Create State-Transition Diagram

- Also called **State Machine Diagram**.
- Shows states, events, and transitions clearly.





Other Behavioral Modelling Tools

1. **Sequence Diagrams**

- Show interaction among objects over time.

2. **Activity Diagrams**

- Represent workflows and control flow.

3. **Statecharts**

- Extended state diagrams with hierarchy and concurrency.

Importance of Behavioral Modelling

- Helps understand dynamic system behavior.
- Useful for real-time and event-driven systems.
- Forms the basis for detailed design and testing.

Q.5 (a) Explain Agility along with the Principles of Agility (10 marks)

Agility

Agility in software engineering refers to the ability of a development process to **respond quickly and effectively to change**. Agile development emphasizes **customer collaboration, incremental delivery, simplicity, and adaptability** over rigid plans and heavy documentation.

Agile methods are best suited for projects where **requirements change frequently** and rapid delivery is essential.

Principles of Agility

1. **Customer Satisfaction**

- Deliver valuable software early and continuously to satisfy customers.

2. **Welcome Change**

- Changing requirements are accepted even late in development.

3. **Incremental Delivery**

- Working software is delivered in small, frequent increments.

4. **People over Process**

- Skilled individuals and teamwork are more important than rigid processes and tools.

5. **Face-to-Face Communication**

- Direct communication is the most effective method of information exchange.

6. **Working Software as a Measure of Progress**

- Progress is measured by functional software, not documentation.

7. **Sustainable Development**

- Maintain a constant pace that can be sustained indefinitely.

8. **Technical Excellence**

- Continuous attention to good design and quality improves agility.

9. **Simplicity**

- Do only what is necessary; avoid unnecessary work.

10. **Self-Organizing Teams**

- Teams organize their own work for better efficiency and creativity.

Q.5 (b) Explain the Extreme Programming (XP) Process (06 marks)

Extreme Programming (XP) is an **Agile process model** that focuses on **customer satisfaction, simplicity, and high-quality code** through frequent feedback and continuous improvement.

Phases of the XP Process

1. **Planning**

- User stories are collected from customers.

- Release planning and iteration planning are performed.

2. **Design**

- Simple design is preferred.

- Use of CRC (Class–Responsibility–Collaboration) cards.

3. **Coding**

- Pair programming is followed.
- Collective code ownership is practiced.
- Coding standards are strictly applied.

4. **Testing**

- Test cases are written before coding (Test-Driven Development).
- Continuous unit and acceptance testing

5. **Listening (Feedback)**

- Continuous customer feedback is taken to refine requirements.

Key XP Practices

- Pair programming
- Refactoring
- Continuous integration
- Small releases

Q.5 (c) Explain the Critics of Extreme Programming (XP) (04 marks)

Despite its advantages, XP has certain **limitations and criticisms**:

1. **Not Suitable for Large Projects**

- XP works best with small teams and small to medium projects.

2. **Requires High Customer Involvement**

- Continuous customer availability is difficult in real-world projects.

3. **Minimal Documentation**

- Lack of documentation may cause maintenance issues.

4. **Relies on Highly Skilled Developers**

- Success depends heavily on experienced and disciplined team members.

Q6.**a) Explain the Scrum Flow Process (08 marks)**

Scrum is an **Agile framework** used for managing and developing complex software products. It follows an **iterative and incremental** approach.

Scrum Flow Process

1. **Product Backlog**

- A prioritized list of features, enhancements, and bug fixes.
- Maintained by the **Product Owner**.

2. **Sprint Planning**

- Team selects items from the product backlog.
- Sprint goal and sprint backlog are defined.

3. **Sprint Backlog**

- Set of tasks to be completed during a sprint (usually 2–4 weeks).

4. **Sprint (Iteration)**

- Development activities including design, coding, and testing.
- Team works collaboratively to produce a working increment.

5. **Daily Scrum Meeting**

- 15-minute stand-up meeting.
- Team members answer:
    - What did I do yesterday?
    - What will I do today?
    - Any impediments?

6. **Increment**

- A potentially shippable product developed at the end of the sprint.

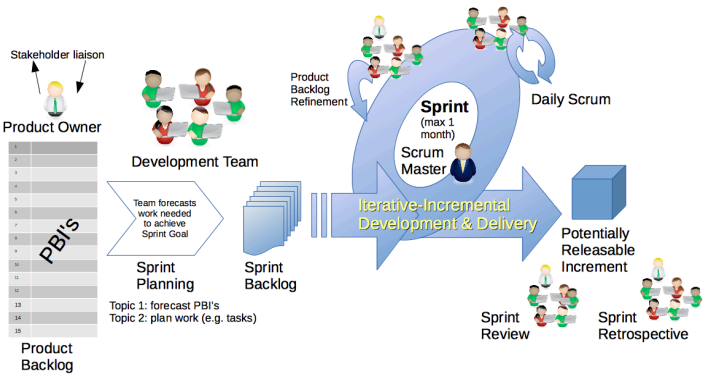
7. **Sprint Review**

- Increment is demonstrated to stakeholders.

- Feedback is collected.

8. **Sprint Retrospective**

- Team reflects on the sprint.
- Identifies improvements for the next sprint.



b) Explain the Communication Principles Guiding Framework Activity (08 marks)

Communication is the first framework activity in software process models. Effective communication ensures correct understanding of requirements.

Communication Principles

1. **Prepare Before Communicating**

- Understand the problem and objectives before discussions.

2. **Involve All Stakeholders**

- Customers, users, managers, and developers must participate.

3. **Listen Actively**

- Focus on understanding stakeholder needs rather than assumptions.

4. **Facilitate Open Communication**

- Encourage questions, clarifications, and feedback.

5. **Use Simple and Clear Language**

- Avoid technical jargon when interacting with customers.

6. **Draw and Visualize**

- Use diagrams, sketches, and models for better understanding.

7. **Document Decisions**

- Record key requirements, assumptions, and decisions.

8. **Negotiate and Collaborate**

- Resolve conflicts through discussion and compromise.

c) How can we validate and test principles in coding? (04 marks)

Validation and testing principles ensure that code meets requirements and quality standards.

Principles for Validation and Testing in Coding

1. **Early Testing**

- Test cases are prepared early to detect errors sooner.

2. **Test All Levels**

- Perform unit, integration, system, and acceptance testing.

3. **Traceability**

- Ensure every requirement has corresponding test cases.

4. **Independent Testing**

- Testing should be done by developers and independent testers.

Q.7 (a) Define Project. Show the contrast of software projects with other types of projects (06 marks)

Definition of a Project

A **project** is a **temporary endeavor** undertaken to create a **unique product, service, or result**, with a defined **start, end, scope, cost, and schedule**.

Contrast between Software Projects and Other Projects

Aspect	Software Projects	Other Projects (Construction/Manufacturing)
Nature	Intangible (code, documents)	Tangible (buildings, machines)
Requirement Stability	Frequently changing	Mostly stable
Production	Development-oriented	Manufacturing-oriented
Visibility of Progress	Difficult to visualize	Easily visible
Maintenance	High and continuous	Comparatively low
Failure Reasons	Poor requirements, complexity	Material, labor, logistics

Q.7 (b) Explain the ISO/IEC 12207 Software Development Life Cycle with a neat diagram (10 marks)

ISO/IEC 12207 is an **international standard** that defines a **common framework for software life cycle processes**. It ensures **consistency, quality, and control** in software development.

ISO/IEC 12207 Life Cycle Processes

ISO 12207 groups processes into **three main categories**:

1. Primary Life Cycle Processes

- **Acquisition** – Obtaining software products.
- **Supply** – Supplying software to the customer.
- **Development** – Requirements, design, coding, testing.
- **Operation** – Running the software.
- **Maintenance** – Modifications and upgrades.

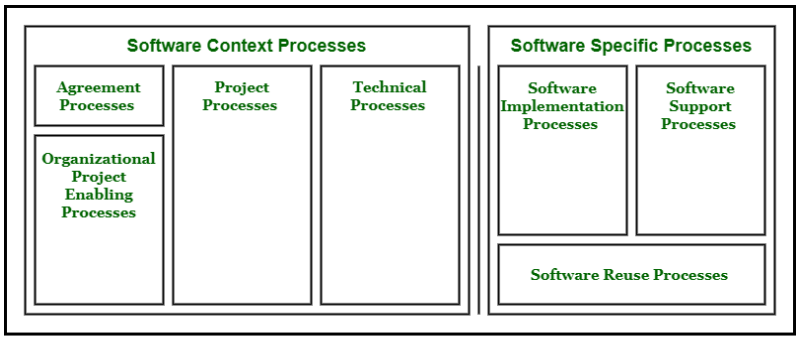
2. Supporting Life Cycle Processes

- Documentation
- Configuration management
- Quality assurance
- Verification
- Validation
- Joint review
- Audit
- Problem resolution

3. Organizational Life Cycle Processes

- Management
- Infrastructure
- Improvement
- Training

Neat Diagram (ISO 12207 Overview)



SDLC Activities of ISO 12207

Advantages of ISO 12207

- Provides a standardized software life cycle framework.

- Improves software quality and reliability.
- Applicable to large and complex projects.
- Enhances communication among stakeholders.

Q.7 (c) What are Outsourced Projects? (04 marks)

Outsourced projects are software projects where an organization **delegates development work to an external vendor or third party** instead of doing it in-house.

Characteristics of Outsourced Projects

1. **External Development Team**

- Work is carried out by another company or organization.

2. **Cost Effectiveness**

- Reduces development and operational costs.

3. **Access to Expertise**

- Enables use of specialized skills and technologies.

4. **Contract-Based Execution**

- Governed by legal agreements defining scope, cost, and timelines.

Examples

- Offshore software development
- IT services outsourced to vendors

Q.8 (a) Illustrate the Cost–Benefit Evaluation Techniques (10 marks)

Cost–Benefit Evaluation is used to determine whether a software project is **economically feasible** by comparing expected costs with anticipated benefits.

Types of Costs

- **Development costs**: hardware, software, manpower, training
- **Operational costs**: maintenance, upgrades, support
- **Indirect costs**: overheads, downtime

Types of Benefits

- **Tangible benefits**: increased revenue, reduced labor cost
- **Intangible benefits**: customer satisfaction, improved decision making

---

Cost–Benefit Evaluation Techniques

1. Net Present Value (NPV)

- Calculates the present value of future benefits minus costs.
- Considers the time value of money.
- Positive NPV → project is feasible.

2. Return on Investment (ROI)

- Measures profitability of the project.
- ROI = (Net Benefits / Total Cost) × 100
- Higher ROI indicates better investment.

3. Payback Period

- Time required to recover the initial investment.
- Shorter payback period is preferred.

4. Benefit–Cost Ratio (BCR)

- Ratio of total benefits to total costs.

- BCR > 1 indicates a viable project.

5. Internal Rate of Return (IRR)

- Discount rate at which NPV becomes zero.
- Project is accepted if IRR exceeds minimum required rate.

Importance

- Supports management decision making.
- Helps compare multiple project alternatives.
- Reduces financial risk.

Q.8 (b) Illustrate the Concept of Risk Evaluation (10 marks)

Risk evaluation is the process of **identifying, analyzing, and prioritizing risks** that may negatively impact a software project.

Steps in Risk Evaluation

1. Risk Identification

- Identify potential risks such as:
    - Technical risks
    - Schedule risks
    - Cost risks
    - Personnel risks
    - External risks

2. Risk Analysis

- Estimate:
    - **Probability** of risk occurrence

- **Impact** on project objectives

3. Risk Prioritization

- Rank risks based on severity.
- Focus on high-probability, high-impact risks.

4. Risk Exposure (RE)

- RE = Probability × Loss
- Used to quantify risk impact.

5. Risk Mitigation Planning

- Actions taken to reduce probability or impact.
- Examples: training, prototyping, resource backup.

6. Risk Monitoring

- Track risks throughout the project lifecycle.
- Update risk status regularly.

Q.9 (a) Explain the details to be drafted for achieving quality in software (06 marks)

To achieve **high quality software**, certain **quality-related details must be clearly drafted and documented** during software development.

Details to be Drafted for Software Quality

1. **Quality Objectives**

- Define measurable quality goals such as reliability, performance, and usability.

2. **Quality Standards**

- Specify applicable standards (ISO, IEEE, coding standards).

3. **Quality Assurance Plan**

- Defines QA activities, roles, responsibilities, and schedules.

4. **Verification and Validation Criteria**

- Clearly mention how requirements, design, and code will be verified and validated.

5. **Testing Strategy**

- Define levels of testing, test environment, and acceptance criteria.

6. **Configuration and Change Management Plan**

- Controls changes to software artifacts to maintain quality.

Q.9 (b) Explain the Software Quality Characteristics of ISO 9126 (08 marks)

ISO/IEC 9126 is an international standard that defines a **software product quality model**. It specifies **six main quality characteristics**.

Software Quality Characteristics (ISO 9126)

1. **Functionality**

- Ability of software to provide required functions.
- Sub-characteristics: suitability, accuracy, interoperability, security.

2. **Reliability**

- Ability to maintain performance under stated conditions.
- Sub-characteristics: maturity, fault tolerance, recoverability.

3. **Usability**

- Ease with which users can learn and use the software.
- Sub-characteristics: understandability, learnability, operability.

4. **Efficiency**

- Relationship between performance and resource usage.
- Sub-characteristics: time behavior, resource utilization.

5. **Maintainability**

- Ease of modifying software.
- Sub-characteristics: analyzability, changeability, stability, testability.

6. **Portability**

- Ability to transfer software to different environments.
- Sub-characteristics: adaptability, installability, replaceability.

Q.9 (c) Explain Process Requirements for Process Quality Management (06 marks)

Process Quality Management ensures that software development processes are **well defined, controlled, and continuously improved**.

Process Requirements for Quality Management

1. **Defined and Documented Processes**

- Software processes must be clearly defined and documented.

2. **Process Standardization**

- Use of standard models and procedures across projects.

3. **Process Measurement and Metrics**

- Collect metrics to monitor process performance.

4. **Process Compliance**

- Ensure processes are followed as per defined standards.

5. **Continuous Process Improvement**

- Regular reviews and corrective actions to improve processes.

6. **Training and Skill Development**

- Train personnel to correctly follow defined processes.

Q.10 (a) Explain about Decomposition Techniques (10 marks)

Decomposition is a fundamental technique in software engineering used to **divide a complex problem into smaller, manageable parts**. It helps in better understanding, design, estimation, and implementation of software systems.

Objectives of Decomposition

- Reduce complexity
- Improve understanding and clarity
- Enable parallel development
- Improve maintainability and testing

Types of Decomposition Techniques

1. Functional Decomposition

- System is divided based on **functions or operations** it performs.
- Uses a **top-down approach**.
- Represented using **Data Flow Diagrams (DFDs)**.
- Example: Payroll system → Calculate salary, generate payslip, maintain records.

2. Process Decomposition

- Focuses on breaking down processes into subprocesses.
- Emphasizes *how* tasks are carried out.
- Useful during analysis phase.

3. Data Decomposition

- System is decomposed based on **data objects and data structures**.
- Emphasizes relationships among data.
- Represented using **Entity-Relationship (ER) diagrams**.

4. Object-Oriented Decomposition

- System is decomposed into **objects and classes**.

- Each object contains data and associated operations.
- Supports reuse and maintainability.

5. Problem Decomposition

- Divides the overall problem into smaller sub-problems.
- Each sub-problem is solved independently and integrated later.

Q.10 (b) Explain the COCOMO II Model (10 marks)

COCOMO II (Constructive Cost Model II) is an **algorithmic software cost estimation model** proposed by **Barry Boehm**. It estimates **effort, cost, and schedule** for modern software development processes.

Need for COCOMO II

- Supports modern development practices
- Handles reuse, prototyping, and object-oriented development
- Improves estimation accuracy

Sub-Models of COCOMO II

1. Application Composition Model

- Used in early stages.
- Based on **Object Points**.
- Suitable for GUI-based applications.

2. Early Design Model

- Used when architecture is roughly defined.
- Uses **Function Points or KSLOC**.
- Considers a small set of cost drivers.

3. Post-Architecture Model

- Used after system architecture is finalized.
- Uses detailed cost drivers and scale factors.
- Provides the most accurate estimates.

Effort Estimation Formula

Effort (PM)=A×(Size)E×∏EMi\text{Effort (PM)} = A \times (\text{Size})^E \times \prod EM_iEffort (PM)=A×(Size)E×∏EMi

Where:

- **A** = Constant
- **Size** = KSLOC
- **E** = Scale factor exponent
- **EM** = Effort multipliers

Scale Factors (5)

- Precedentedness (PREC)
- Development Flexibility (FLEX)
- Architecture / Risk Resolution (RESL)
- Team Cohesion (TEAM)
- Process Maturity (PMAT)

Advantages of COCOMO II

- Accurate and flexible
- Supports reuse and modern practices
- Widely accepted and standardized

Limitations

- Requires historical data
- Estimation accuracy depends on correct input values