1.Explain the fundamental principles of software engineering.

Fundamental Principles of Software Engineering

1. **Modularity: Software is divided into smaller, manageable parts or modules, making it easier to develop, test, and maintain.**
2. **Abstraction: Focuses on essential details while hiding unnecessary complexity, helping developers manage large systems more easily.**
3. **Encapsulation: Combines data and the methods that operate on it into a single unit, protecting internal states and promoting data security.**
4. **Reusability: Encourages writing code that can be reused in different applications or systems, reducing duplication and saving development time.**
5. **Maintainability: Ensures the software can be easily modified, updated, or fixed, which is essential for long-term success and adaptability.**

2.Analyse different software process models, comparing their structures, advantages, and challenges.

Analysis of Different Software Process Models

1. **Waterfall Model**
   - **Structure: Linear and sequential (Requirement → Design → Implementation → Testing → Maintenance).**
   - **Advantages: Simple, easy to manage, well-suited for clearly defined requirements.**
   - **Challenges: Inflexible to changes; late discovery of errors.**
2. **Agile Model**
   - **Structure: Iterative and incremental; delivers working software in small sprints.**
   - **Advantages: Flexible, promotes customer collaboration, quick feedback.**
   - **Challenges: Requires continuous involvement; difficult to scale for large projects.**
3. **Spiral Model**
   - **Structure: Combines iterative development with risk analysis in a spiral loop.**
   - **Advantages: Focuses on risk management; suitable for large, complex projects.**
   - **Challenges: Costly and complex; requires expertise in risk assessment.**
4. **V-Model**
   - **Structure: Extension of Waterfall with a corresponding testing phase for each development stage.**
   - **Advantages: Emphasizes verification and validation; good for small projects.**
   - **Challenges: Not flexible for changes; early testing requires detailed specifications.**

3.Analyse the Software Development Life Cycle (SDLC) phases, explaining how each phase contributes to the successful delivery of a software product.

Analysis of SDLC Phases (5 Marks)

1. **Requirement Gathering and Analysis**
   - **Understands user needs and documents system requirements.**
   - **Ensures clear goals and prevents misunderstandings.**
2. **System Design**
   - **Converts requirements into system architecture and design.**
   - **Guides developers with a clear blueprint, including data flow, UI, and technology stack.**
3. **Implementation (Coding)**
   - **Developers write the actual code based on design specifications.**
   - **Transforms ideas into a working software product.**
4. **Testing**
   - **Identifies and fixes bugs through various testing methods (unit, integration, system).**
   - **Ensures the software meets quality and functional requirements.**
5. **Deployment**
   - **Delivers the software to the end-users or clients.**
   - **Makes the product accessible in a real environment.**
6. **Maintenance**
   - **Involves updates, bug fixes, and enhancements after deployment.**
   - **Keeps the software relevant and functional over time.**

4.Role of Functional and Non-Functional Requirements in Software Development.

1. **Functional Requirements**
   - **Define what the system should do, such as specific features, operations, and behaviors.**
   - **Examples: user authentication, data entry, report generation.**
   - **Role: Guide developers in implementing the core functionalities users expect.**
2. **Non-Functional Requirements**
   - **Define how the system should behave, focusing on quality attributes.**
   - **Examples: performance, security, usability, scalability.**
   - **Role: Ensure the system meets user expectations in terms of reliability and efficiency.**
3. **Importance in Development**
   - **Together, they provide a complete understanding of what the software must deliver.**
   - **Functional requirements drive system capabilities, while non-functional ones shape user satisfaction and long-term viability.**
4. **Impact on Design and Testing**
   - **Both types influence design choices and are essential for thorough testing strategies.**

5.Evaluation of Software Design Methodologies .

1. **Structured Design**
   - **Approach: Top-down, focuses on functions and procedures.**
   - **Best For: Small to medium projects with well-defined processes (e.g., payroll systems).**
   - **Pros: Easy to understand and implement.**
   - **Cons: Poor adaptability to change; limited reuse.**
2. **Object-Oriented Design (OOD)**
   - **Approach: Models software using objects (data + behavior).**
   - **Best For: Complex, large-scale systems like e-commerce platforms or mobile apps.**
   - **Pros: Promotes reuse, scalability, and maintainability.**
   - **Cons: Initial design may be complex.**
3. **Component-Based Design**
   - **Approach: Builds software using reusable and interchangeable components.**
   - **Best For: Enterprise applications, distributed systems.**
   - **Pros: Increases modularity and reusability.**
   - **Cons: Component integration can be complex.**
4. **Service-Oriented Architecture (SOA)**
   - **Approach: Designs software as a collection of services.**
   - **Best For: Cloud-based or enterprise systems (e.g., banking applications).**
   - **Pros: Scalable, loosely coupled.**
   - **Cons: Requires robust infrastructure and management.**

7.Analyse the risk management process in software engineering and determine its impact on project success

Risk Management in Software Engineering

1. **Risk Identification**
   - **Detect potential risks (e.g., budget overruns, scope creep, technology failures).**
   - **Helps teams prepare for possible project disruptions.**
2. **Risk Analysis**
   - **Evaluate each risk based on likelihood and impact.**
   - **Prioritizes which risks need immediate attention or mitigation.**
3. **Risk Planning (Mitigation and Contingency)**
   - **Develop strategies to reduce or eliminate risks.**
   - **Examples: backup plans, extra resources, buffer time.**
4. **Risk Monitoring and Control**
   - **Continuously track risks throughout the project.**
   - **Allows early response to emerging issues, minimizing damage.**

Impact on Project Success

- **Effective risk management reduces project failure chances, ensures better decision-making, improves team confidence, and supports on-time, on-budget delivery.**
  Example: Identifying a risk of team skill gaps early allows for timely training, avoiding future delays.

6.Evaluate various software project scheduling techniques and determine which technique is most suitable for different types of software projects.

Evaluation of Software Project Scheduling Techniques

1. **Gantt Chart**
   - **Description: A bar chart showing tasks over time.**
   - **Best For: Small to medium projects with straightforward timelines.**
   - **Pros: Easy to understand and track progress.**
   - **Cons: Hard to manage dependencies in complex projects.**
2. **PERT (Program Evaluation and Review Technique)**
   - **Description: Uses probabilistic time estimates (optimistic, pessimistic, most likely).**
   - **Best For: R&D or uncertain projects with variable task durations.**
   - **Pros: Handles uncertainty; useful for risk analysis.**
   - **Cons: Complex to construct and maintain.**
3. **Critical Path Method (CPM)**
   - **Description: Identifies the longest path of dependent tasks.**
   - **Best For: Large, deadline-driven projects (e.g., software deployment).**
   - **Pros: Highlights tasks critical to timely delivery.**
   - **Cons: Doesn't account for task uncertainty like PERT.**
4. **Agile/Scrum Scheduling**
   - **Description: Uses sprints and backlogs for iterative planning.**
   - **Best For: Projects with evolving requirements (e.g., mobile apps).**
   - **Pros: Highly flexible, promotes frequent feedback.**
   - **Cons: Less effective in fixed-scope, deadline-heavy projects.**

7.Analysis of Black Box and White Box Testing

1. **Black Box Testing**
   - **Objective: Validate the system's functionality without knowing the internal code.**
   - **Methods: Equivalence partitioning, boundary value analysis, functional testing.**
   - **Effectiveness: Detects missing functions, incorrect behavior, and interface issues.**
   - **Best For: System testing, user acceptance testing.**
2. **White Box Testing**
   - **Objective: Verify the internal logic, structure, and code paths of the application.**
   - **Methods: Statement coverage, branch coverage, path testing, loop testing.**
   - **Effectiveness: Detects logical errors, dead code, and security flaws.**
   - **Best For: Unit testing, integration testing.**
3. **Comparison**
   - **Black Box focuses on what the system does, while White Box focuses on how it works.**
   - **Black Box requires no code knowledge; White Box requires programming skills.**
   - **Both are complementary: combining them leads to better test coverage and higher quality.**

8.Significance of Software Quality Assurance (SQA)

1. **Ensures Product Quality**
   - **SQA focuses on maintaining high standards in software development through systematic activities like reviews, audits, and testing.**
   - **It ensures the final product is reliable, efficient, and defect-free.**
2. **Prevents Defects Early**
   - **By applying quality checks throughout the development process, SQA helps identify and fix issues before they become costly problems.**
3. **Improves Customer Satisfaction**
   - **A high-quality product meets user expectations, which builds trust and credibility with customers.**
4. **Supports Compliance and Standards**
   - **SQA ensures adherence to industry standards (e.g., ISO, CMMI), making the software more marketable and legally compliant.**
5. **Reduces Development Costs and Time**
   - **Early detection of errors and continuous quality checks reduce rework, delays, and maintenance costs, leading to more efficient project delivery.**

9.Impact of Agile Methodology in Modern Software Development

1. **Flexibility and Adaptability**
   Agile enables teams to respond quickly to changing requirements, allowing frequent updates and continuous improvement.
2. **Customer Collaboration**
   Agile involves customers throughout development, ensuring the product aligns closely with user needs and expectations.
3. **Faster Delivery**
   By using iterative sprints, Agile delivers working software early and regularly, improving time-to-market.
4. **Improved Team Communication**
   Daily stand-ups and close collaboration enhance transparency and coordination among team members.
5. **Higher Product Quality**
   Continuous testing and integration in Agile reduce defects and improve software quality.

10.Role of Quality Assurance in Reducing Software Failures

1. **Early Detection of Defects**
   Quality assurance (QA) involves systematic testing and reviews early in development, identifying bugs before they escalate.
2. **Process Standardization**
   QA enforces consistent development processes and best practices, reducing errors caused by ad-hoc or poor practices.
3. **Verification and Validation**
   QA ensures the software meets specified requirements and performs as expected, preventing functional failures.
4. **Continuous Monitoring**
   QA includes ongoing monitoring and testing, catching regressions or new defects introduced during updates.
5. **Improved Reliability and User Confidence**
   By minimizing failures, QA enhances software stability and builds user trust in the product.

11.Comparison of Software Reliability Models (5 Marks)

1. **Jelinski-Moranda Model**
   - **Assumes a fixed number of initial faults.**
   - **Uses failure data to estimate reliability improvement as faults are fixed.**
   - **Best for: Simple systems with known fault counts.**
   - **Limitations: Assumes all faults have equal chance of failure.**
2. **Goel-Okumoto (Exponential) Model**
   - **Models the fault detection rate as a decreasing function over time.**
   - **Suitable for systems where faults are removed continuously during testing.**
   - **Best for: Software with ongoing testing and fault removal.**
   - **Limitations: Assumes a constant failure rate decay.**
3. **Musa Model**
   - **Considers execution time and operational profile to estimate reliability.**
   - **Useful for operational testing environments.**
   - **Best for: Real-time or mission-critical systems.**
   - **Limitations: Requires detailed usage data.**

Suggested Best Model

The Goel-Okumoto model is widely used because it realistically models fault detection over time and adapts well to many software testing scenarios.

13. Differentiate between prescriptive and specialized process models.

Feature	Prescriptive Models	Specialized Models
Scope	General-purpose	Domain/project-specific
Structure	Well-defined steps/phases	Customized or focused
Examples	Waterfall, Incremental, Spiral	CBD, Formal Methods, RAD, Agile
Flexibility	Less flexible	More adaptable to specific needs
Usage	Broad software development	Specialized environments or goals

12.Characteristics of Good Software .

1. **Reliability**
   The software should consistently perform its intended functions without failure over time, ensuring dependable operation.
2. **Usability**
   It must be easy to learn, use, and understand by the target users, providing a satisfactory user experience.
3. **Maintainability**
   Good software should be easy to modify, update, and fix, allowing adaptation to changing requirements with minimal effort.
4. **Efficiency**
   It should make optimal use of system resources such as memory, processing power, and network bandwidth to perform tasks quickly.
5. **Portability**
   The software should be easily transferable to different environments or platforms without requiring major changes.

14.Analysis of Software Engineering Ethics Principles

Software engineering ethics is guided by professional codes such as those from ACM/IEEE, which emphasize key principles like public interest, quality, honesty, and professionalism.

1. Influence on Decision-Making:Ethical principles ensure that engineers make responsible decisions, especially when facing trade-offs between cost, speed, and safety. For example, choosing to delay a product to fix security flaws rather than rushing its release reflects ethical judgment.

2. Impact on Product Quality:

Ethical practice promotes rigorous testing, accurate documentation, and reliable design, all of which contribute to higher software quality. Developers are encouraged to avoid shortcuts that compromise maintainability or performance.

3. Building User Trust:

By respecting user privacy, securing data, and being transparent about software behavior, ethical software engineers help build user trust. Trust increases product adoption and long-term reputation.

15.Modular Architecture Design for an Online Library Management System

1. Overview of the System:

The system manages books, users, borrowing/returning, and admin operations for a library. A modular architecture divides the software into independent, interchangeable modules with specific responsibilities

2. Key Modules:

User Management Module – Handles registration, login, and user roles.
Book Catalog Module – Manages book data (search, view, categorize).
Borrow/Return Module – Tracks borrowing, due dates, and returns.
Notification Module – Sends reminders and alerts to users.
Admin Module – For inventory management, reports, and user control.
Database Access Module – Central interface for data storage/retrieval.

3. How Design Enhances:

Maintainability:Each module is loosely coupled and self-contained. Bugs or changes in one module (e.g., book catalog) don't affect others, making debugging and updates easier.

Flexibility:New features (like an e-book reader or payment system) can be added as separate modules without rewriting the entire codebase.

Performance:Modules can be optimized independently. For example, caching can be added to the Book Catalog module to reduce database load without impacting others.

16.Importance of Data Flow Diagrams (DFDs) in Software Design

1. Visual Representation of System Functionality:DFDs provide a clear and graphical way to understand how data moves through a system. They show processes, data stores, and external entities, helping developers and stakeholders grasp the system's logic at a glance.

2. Enhances Communication:DFDs act as a bridge between technical and non-technical stakeholders. They simplify complex processes and promote better understanding, which is crucial for gathering accurate requirements.

3. Aids in Requirement Analysis:By mapping out data sources, destinations, and flow paths, DFDs help identify missing or redundant processes, ensuring the system requirements are complete and well-defined.

4. Supports Modular Design:Each process in a DFD can be further broken down into sub-processes, promoting a modular approach to system design. This makes the system easier to develop, maintain, and test.

5. Helps Detect Design Flaws Early:DFDs can reveal inconsistencies, unnecessary data movement, or inefficiencies in system design. Detecting such issues early reduces cost and rework during later development stages.

17.Significance of Unified Modeling Language (UML) in Software Engineering

1.Standardization:UML is an industry-standard modeling language that provides a consistent way to design and document software systems across teams and projects.

2.Improves Understanding:It visually represents system components (like classes, objects, and processes), making it easier to understand complex systems.

3.Supports Requirement Analysis:Use Case and Activity diagrams help capture functional requirements early, aiding in accurate system planning and development.

4.Enhances Communication:UML bridges communication gaps between developers, designers, and non-technical stakeholders by using easy-to-understand visual models.

5.Facilitates Maintenance and Scalability:By offering clear design documentation, UML makes it easier to maintain, update, and scale the system in the future.

18.Explain function-oriented and object-oriented software design.

1. Function-Oriented Design:

- **Focuses on functions and procedures that operate on data.**
- **The system is divided into modules, each performing a specific task.**
- **Data is global, and functions access and modify this data.**
- **Example: Structured Design, where the design starts with a Data Flow Diagram (DFD).**
- **Limitation: Difficult to manage large systems and maintain code as functionality grows.**

2. Object-Oriented Design:

- **Focuses on objects that combine both data and behavior (methods).**
- **The system is built using classes, which are blueprints for objects.**
- **Promotes encapsulation, inheritance, and polymorphism.**
- **Example: Designing a "Student" class with properties (name, ID) and methods (enroll, updateProfile).**
- **Advantage: Easier to maintain, reuse, and scale, making it ideal for complex, real-world applications.**

19.Cost estimation techniques are essential in planning and budgeting software projects. Common techniques include:

- **Expert Judgment – Relies on experienced professionals for quick estimates but can be biased.**
- **Analogous Estimating – Uses past similar projects to estimate costs; it's fast but less accurate.**
- **Parametric Estimating – Calculates costs using statistical models (e.g., cost per LOC); needs historical data.**
- **Bottom-Up Estimating – Estimates costs by summing detailed component estimates; highly accurate but time-consuming.**
- **Three-Point Estimating – Uses optimistic, pessimistic, and most likely values to account for uncertainty.**

Each method varies in accuracy, effort, and applicability, and often a combination is used for reliable results.

20. What are the advantages and disadvantages of automated software testing?

Advantages of Automated Software Testing:

- **Faster Execution – Tests can be run quickly and repeatedly, especially useful for regression testing.**
- **Reusability – Test scripts can be reused across different versions of the software.**
- **Accuracy – Reduces human error, ensuring consistent results.**

Disadvantages of Automated Software Testing:

- **High Initial Cost – Tools and script development can be expensive initially.**
- **Maintenance Overhead – Scripts must be updated when the application changes.**
- **Requires Skilled Resources – Needs knowledge of tools and scripting languages.**

21. Analyse the impact of reverse engineering in modernizing legacy software.

□ Understanding Legacy Systems: Reverse engineering helps uncover the structure, logic, and functionality of outdated software where documentation is missing or outdated.

□ Improves Maintainability: By analyzing the system's design, developers can refactor code to make it more modular, readable, and maintainable.

□ Facilitates Migration: It aids in migrating old systems to modern platforms (e.g., from COBOL to Java) by revealing dependencies and workflows.

□ Enhances Security and Compliance: Helps identify hidden vulnerabilities or outdated components that may pose security risks.

□ Supports Reuse and Integration: Enables reuse of valuable legacy components in new systems or integration with modern applications.

22. . What are the main challenges in implementing software reuse in large-scale applications?

1.Component Compatibility: Reused components may not easily integrate due to differences in architecture, platforms, or programming languages.

2.Quality Assurance: Ensuring the reused components meet performance, security, and reliability standards can be difficult.

3.Lack of Documentation: Incomplete or outdated documentation can hinder understanding and reuse of existing code.

4.Version Control and Maintenance: Managing updates, bug fixes, and compatibility across multiple reused components adds complexity.

5. Organizational Resistance: Teams may resist reuse due to a "not-invented-here" mindset or lack of proper reuse policies and incentives.

23.Compare various software re-engineering techniques

1. Code Restructuring

Description: Improves the structure of source code without changing its functionality.

Pros: Enhances readability and maintainability.

Cons: No functional improvement or feature addition.

2. Data Reengineering

Description: Focuses on improving data structures, formats, or databases.

Pros: Boosts performance and data consistency.

Cons: Can be complex if data is highly integrated.

3. Reverse Engineering

Description: Analyzes software to understand its design and documentation.

Pros: Useful when documentation is missing.

Cons: Time-consuming and may not fully capture system behavior.

4. Forward Engineering

Description: Uses reverse-engineered information to redesign or rebuild the system using modern technologies.

Pros: Results in a fully modernized system.

Cons: Resource-intensive.

5. Migration

Description: Moves software from old platforms/languages to new ones.

Pros: Enhances compatibility and performance.

Cons: Risk of data or functionality loss.

24. Software Maintenance Plan for a Cloud-Based Application:

1.Preventive Maintenance: Regularly update libraries, frameworks, and cloud configurations to prevent potential issues and improve performance.

2.Corrective Maintenance: Continuously monitor logs and user feedback to identify and fix bugs or errors in real-time.

3.Adaptive Maintenance: Modify the application to stay compatible with evolving cloud environments, APIs, OS updates, and third-party services.

4.Perfective Maintenance: Optimize features, UI, and performance based on user analytics and emerging requirements.

5.Backup & Security Management: Implement automated backups, monitor vulnerabilities, and apply security patches to protect data and ensure availability.

25. Develop a testing framework for a web-based ecommerce application.

1. Types of Testing: Include unit, integration, functional, performance, security, and user acceptance testing.

2.Automation Tools: Use Selenium for UI testing, JUnit/TestNG for backend tests, and JMeter for performance testing.

3.Test Environment: Set up separate development, staging, and production-like environments for reliable testing.

4.Continuous Integration: Integrate automated tests with CI tools like Jenkins or GitHub Actions for automatic test execution.

5. Reporting: Generate test reports for tracking results and maintain tests regularly as the application evolves.

26. Software Reuse Strategy for Mobile App Development

1.Component Library Creation: Develop and maintain a centralized repository of reusable UI components, modules, and APIs.

2.Standardized Development Guidelines: Establish coding standards and design patterns to ensure components are reusable and compatible.

3.Modular Architecture: Use modular design (e.g., microservices or modular SDKs) to promote easy reuse and integration.

4.Documentation and Training: Provide thorough documentation and train developers on how to effectively reuse components.

5.Version Control and Feedback Loop: Implement versioning for reusable assets and encourage developer feedback to improve and update reusable components.