1. Introduction to Software Engineering

What is Software?

Software is a set of instructions, data or programs used to operate computers and execute specific tasks1. It's not just programs but includes:

- Documentation
- Configuration data
- User manuals
- System requirements

Software vs Hardware

Key differences include:

- Development: Software is developed or engineered, not manufactured in the classical sense1
- Durability: Software doesn't "wear out" like hardware1
- Labor Requirements: Software requires different labor patterns compared to hardware manufacturing1

Software Engineering Definition

Software engineering is defined as "the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines"1. The IEEE definition states it as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"1.

Software Engineering as a Layered Technology

Software engineering consists of1:

- Quality Focus: Foundation layer
- Process Model: Systematic approach
- Methods: Technical procedures
- Tools: Automated support

Software Applications Types

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- Web applications
- Business Intelligence software and Big Data
- IoT applications

Software Project Success Factors

According to the Standish Group Chaos Report, project success is measured by[2]:

- On Time: Meeting deadlines
- On Budget: Within financial constraints
- On Target: Meeting requirements
- On Goal: Achieving business objectives
- Value: Delivering business value
- Satisfaction: Stakeholder satisfaction

Related Exam Questions with Answers:
Q: Explain briefly, the four important attributes that all professional software should have
A: The four important attributes are[3]:

1. Acceptability: Software must be acceptable to the type of users for which it is designed
2. Dependability and Security: Software dependability includes reliability, security, and safety
3. Efficiency: Software should not make wasteful use of system resources
4. Maintainability: Software should be written in such a way that it can evolve to meet changing needs

Q: Consider the statement, "Software is not just the programs". Justify this statement with suitable examples
A: Software is not just programs because it includes[4]:

- Documentation: User manuals, system documentation, requirements specifications
- Configuration data: Database schemas, system configuration files
- Data structures: File formats, database designs

- Examples: An operating system includes the kernel (program) plus device drivers, system utilities, configuration files, and user documentation

Q: State the fundamental principles of Software Engineering that are applicable to all types of software systems
A: The fundamental principles include4:

1. Managed software process: Use appropriate project management and quality assurance
2. Software dependability and performance: Build reliable and efficient software
3. Software requirements and specifications: Understand what customers want
4. Software reuse: Reuse existing software components where possible

2. Software Development Life Cycle (SDLC)

SDLC Phases

The traditional SDLC includes1:

1. Requirements Analysis: Understanding the problem
2. System Design: Planning the solution
3. Implementation/Coding: Building the software
4. Testing: Verifying functionality
5. Deployment: Installing the system
6. Maintenance: Ongoing support

Classical Lifecycle Model Limitations

Key limitations that led to Agile methodologies1:

- Rigid sequential approach
- Difficulty accommodating changes
- Late detection of problems
- Heavy documentation requirements

Related Exam Questions with Answers:
Q: State the four main fundamental Software Engineering activities
A: The four main activities are4:

1. Software specification: Defining what the system should do
2. Software development: Designing and programming the system
3. Software validation: Checking that the system does what the customer wants

4. Software evolution: Changing the system in response to changing customer needs

Q: List two limitations in classical lifecycle model which caused the developers to move into Agile Software Methodologies
A: Two key limitations are4:

1. Inflexibility to change: Difficult to accommodate changing requirements once development begins
2. Late feedback: Problems are discovered late in the development cycle, making them expensive to fix

3. Agile Software Development

What is Agile?

Agile is a time-boxed, iterative approach to software delivery that builds software incrementally from the start of the project5. It focuses on:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Three Simple Truths of Agile

1. It is impossible to gather all requirements at the beginning of a project5
2. Whatever requirements could be gathered are guaranteed to change5
3. There will always be more to do than time and money will allow5

Agile Principles

The twelve principles include5:

1. Satisfy the customer through early and continuous delivery
2. Welcome changing requirements
3. Deliver working software frequently
4. Business people and developers must work together daily
5. Build projects around motivated individuals
6. Face-to-face communication is most effective
7. Working software is the primary measure of progress
8. Promote sustainable development

9. Continuous attention to technical excellence
10. Simplicity is essential
11. Best architectures emerge from self-organizing teams
12. Regular reflection and adjustment

Agile vs Traditional Development

Key differences include[6]:

Parameter	**Traditional Methods**	**Agile Methods**
Ease of Modification	Hard	Easy
Development Approach	Predictive	Adaptive
Development Orientation	Process Oriented	Customer Oriented
Project Size	Large	Small or Medium
Planning Scale	Long Term	Short Term
Management Style	Command and Control	Leadership and Collaboration
Documentation	High	Low

Budget	High	Low
Team Size	Medium	Small

Related Exam Questions with Answers:
Q: Discuss briefly plan-driven and agile software development processes
A:

- Plan-driven processes: All process activities are planned in advance and progress is measured against this plan[3]. Suitable for large systems with stable requirements
- Agile processes: Planning is incremental and easier to change to reflect changing customer requirements[3]. Suitable for smaller systems with rapidly changing requirements

Q: Explain briefly the main aim of Agile software development and its underlying principles
A: The main aim is to reduce overhead in software development while maintaining ability to respond to changing requirements[5]. Key principles include:

- Early and continuous delivery of valuable software
- Welcoming changing requirements
- Frequent delivery of working software
- Close collaboration between business people and developers
- Face-to-face communication
- Working software as primary measure of progress

Q: Briefly discuss the practical problems associated with Agile Software Development
A: Practical problems include[3]:

- Customer involvement: Requires significant time commitment from customers
- Team member commitment: Requires motivated, skilled team members
- Maintaining simplicity: Difficult to keep design simple as system evolves
- Documentation: May lack sufficient documentation for maintenance
- Contractual issues: Difficult to write contracts for agile processes

4. Scrum Methodology

Scrum Overview

Scrum is an agile framework that provides both project management and development process5. It focuses on organizing and development of User Stories through iterative sprints.

Scrum Values

The five core values are5:

- Courage: Do the right thing and work on tough problems
- Focus: Everyone focuses on Sprint work and team goals
- Commitment: Personal commitment to achieving team goals
- Respect: Team members respect each other as capable individuals
- Openness: Be open about work and challenges

Scrum Roles

1. Product Owner5:
    - Voice of the customer
    - Manages Product Backlog
    - Defines development targets
    - Prioritizes requirements
2. Scrum Master5:
    - Facilitates the process
    - Removes obstacles
    - Protects team from distractions
    - Ensures Scrum practices are followed
3. Development Team5:
    - Self-organizing and cross-functional
    - 3-9 members optimal size
    - Collectively responsible for deliverables
    - No individual titles or sub-teams

Scrum Artifacts

- Product Backlog: Prioritized list of features and user stories5
- Sprint Backlog: Tasks selected for current sprint5
- Sprint Burndown Chart: Visual progress tracking5

Scrum Events

1. Sprint Planning: Determine what to complete in coming sprint5
2. Daily Stand-up: 15-minute daily sync meeting5
3. Sprint Demo: Show completed work to stakeholders5
4. Sprint Retrospective: Review what worked and what didn't5

User Stories

Format: "As a <type of user>, I want <some goal> so that <some reason>"5

- Written on index cards or sticky notes
- Focus on discussing features rather than documenting them
- Can be written at varying levels of detail (epics vs. stories)

Related Exam Questions with Answers:
Q: What is meant by daily scrum meeting?
A: Daily scrum is a 15-minute meeting held every day where team members answer three questions4:

1. What did I complete yesterday?
2. What will I work on today?
3. Am I blocked by anything?
   The purpose is to synchronize activities and identify impediments quickly5.

Framework Full Name	**Abbreviation**	**How it Compares to Scrum**	**Team Size**
Scaled Agile Framework	SAFe	Multi-layered structure built on top of Scrum with additional roles and ceremonies	50+ people
Large-Scale Scrum	LeSS	Direct extension of Scrum with minimal additions, keeps core Scrum intact	10-50+ people

Disciplined Agile Delivery	DA/DAD	Uses Scrum as one option among many methodologies in a toolkit approach	Variable
Scrum at Scale	Scrum@Scale	Pure Scrum principles replicated across multiple teams with coordination layer	Variable
Nexus Framework	Nexus	Scrum with added integration team and coordination events for multiple teams	15-45 people (3-9 teams)

5. Extreme Programming (XP)

XP Overview

Extreme Programming was pioneered by Kent Beck and provides a disciplined framework focusing on producing code7. It takes best practices to "extreme levels" including pair programming, Test-Driven Development (TDD), and Acceptance Test-Driven Development (ATDD).

XP Practices

- Planning: Creation of user stories with customer priorities7
- Design: Keep It Simple (KIS) principle, CRC cards, spike solutions, refactoring7
- Coding: Unit tests before coding, pair programming, continuous integration7
- Testing: Automated unit tests, daily integration testing, customer acceptance tests7

XP Planning/Feedback Loops

- Pair Programming: Seconds7
- Unit Test: Minutes7
- Pair Negotiation: Hours7

- Stand Up Meeting: One Day[7]
- Acceptance Test: Days[7]
- Iteration Plan: Weeks[7]
- Release Plan: Months[7]

Related Exam Questions with Answers:
Q: Extreme programming expresses user requirements as stories, with each story written on a card. Discuss the advantages and disadvantages of this approach to requirements description
A:
Advantages[3]:

- Stories are easy to understand by customers and developers
- Encourages face-to-face communication
- Flexible and easy to change
- Focus on user needs rather than system functions

Disadvantages[3]:

- Difficult to judge if all requirements are covered
- May lack detail for complex requirements
- Hard to maintain consistency across stories
- Challenging for contractual purposes

6. Other Agile Methodologies

Feature Driven Development (FDD)

Five sequential processes[7]:

1. Develop the overall model
2. Build the feature list
3. Plan by feature
4. Design by feature
5. Build by feature

Dynamic Systems Development Method (DSDM)

Project phases include[7]:

- Pre-Project
- Feasibility
- Foundations

- Exploration
- Engineering
- Incremental Deployment
- Post-Project

Related Exam Questions with Answers:
Q: Compare different agile methodologies and their suitability for different project types
A: Different methodologies suit different contexts6:

- Scrum: Best for projects requiring strong project management and regular stakeholder feedback
- XP: Suitable for projects requiring high code quality and frequent releases
- FDD: Good for larger projects with clear feature requirements
- DSDM: Appropriate for projects with fixed time and budget constraints

7. Feasibility Study

Purpose of Feasibility Study

A feasibility study is conducted before committing to a project to determine whether to8:

- Go ahead
- Do not go ahead
- Think again

Types of Feasibility

1. Economic Feasibility: Cost/benefit analysis8
2. Technical Feasibility: Technical approach and requirements8
3. Operational Feasibility: Maintenance and support capabilities8
4. Organizational Feasibility: Management and technical expertise8

Feasibility Study Components

- Scope: Boundaries of the system8
- Benefits: Quantifiable advantages8
- Planning & Resources: Staffing, equipment, timeline8
- Alternatives & Risks: Options and mitigation strategies8

Related Exam Questions with Answers:

Q: Under what circumstances might a company justifiably charge a much higher price for a software system than the software cost estimate plus a reasonable profit margin?
A: Circumstances include3:

- Unique expertise: Company has specialized knowledge not available elsewhere
- High risk projects: Projects with significant technical or business risks
- Tight deadlines: Rush projects requiring additional resources
- Regulatory compliance: Systems requiring specialized compliance knowledge
- Mission-critical systems: Where failure would have severe consequences

Q: What are the most important differences when software development is planned between agile and plan-based development?
A: Key differences include3:

- Planning horizon: Agile uses short-term iterative planning vs. long-term comprehensive planning
- Requirements handling: Agile welcomes changing requirements vs. plan-based tries to freeze requirements
- Documentation: Agile emphasizes working software vs. comprehensive documentation
- Customer involvement: Agile requires continuous customer collaboration vs. periodic reviews

Framework	**Core Concepts**	**Key Benefits**	**Main Challenges**	**When to Use**	**Key Terms**

Scrum	Fixed-length sprints (1-4 weeks), defined roles (Product Owner, Scrum Master, Development Team), ceremonies, and artifacts[1][2][3]	- Enhanced flexibility and adaptability - Faster delivery of value through iterative cycles - Enhanced product quality through continuous testing - Higher team engagement and self-organization[1][3]	- Requires strict discipline from team - Can be overwhelming for new teams - Potential resistance to change in traditional organizations[2][3]	Projects with changing requirements, teams new to agile, need for structured approach with clear roles[1][2]	Sprint, Scrum Master, Product Owner, Daily Standup, Sprint Review, Sprint Retrospective, Product Backlog[3]
Kanban	Visual workflow management, continuous delivery, Work-in-Progress (WIP) limits, pull-based system[4][5]	- Workflow visualization and transparency - Flexibility and adaptability to existing processes - Continuous improvement culture	- Lack of formal structure can lead to ambiguity - Risk of overwhelming WIP without proper limits - May not provide enough	Continuous flow work, maintenance projects, teams needing flexibility, organizations wanting minimal process change[4][5]	Kanban Board, WIP Limits, Lead Time, Cycle Time, Throughput, Pull System[4][5]

		- Reduced work in progress bottlenecks[4][5]	structure for complex projects[4][5]		
Feature-Driven Development (FDD)	Feature-focused development, 5-step process, short iterations (2 weeks), domain object modeling[6][7][8]	- Client-focused feature delivery - Regular and on-time delivery to customers - Strong progress tracking and reporting - Suitable for large-scale, long-term projects[6][7][8]	- Requires lots of effort and persistence - Heavy reliance on documentation - May be complex for smaller projects - Need for experienced chief programmers[6][8]	Large-scale software projects, teams preferring documentation over meetings, projects with well-defined feature requirements[6][7][8]	Chief Programmer, Class Owner, Domain Expert, Feature List, Domain Object Modeling[6][8]

Extreme Programming (XP)	Pair programming, test-driven development, continuous integration, short iterations (1-2 weeks), extreme practices[9][10]	- Robust software through continuous testing - Fast development cycles with high quality - Low cost of change through rapid feedback - Error avoidance through pair programming[9][10]	- Requires high discipline and commitment - Customer must participate permanently - Relatively high costs due to pair programming - Intensive practices may overwhelm teams[9][10]	Projects with volatile requirements, teams valuing engineering excellence, close customer collaboration available[9][10]	Pair Programming, Test-Driven Development (TDD), Continuous Integration, User Stories, Refactoring, Collective Ownership[9][10]
DSDM (Dynamic Systems Development Method)	8 core principles, timeboxed iterations, business case focus, "enough design up front" approach[11][12]	- Strong business alignment and strategic focus - Controlled project delivery with governance	- Requires understanding of business context - May be too structured for some agile teams	Project contexts with tight scope, need for governance, business-critical applications, traditional organizations transitioning to agile[11][12]	Timeboxing, Business Case, Enough Design Up Front (EDUF), MoSCoW Prioritization, Facilitated Workshops[11][12]

		- Clear project foundations established early - Suitable for project contexts with defined scope[11,12]	- Need for early business case establishment - Complex framework with multiple principles[11,12]		
Test-Driven Development (TDD)	Write tests before code, Red-Green-Refactor cycle, continuous testing approach[13,14]	- Enhanced software quality through systematic testing - Reduced post-deployment bugs - Early detection of issues and defects - Better code design and architecture[13,14]	- Requires significant mindset shift - Initial learning curve for developers - May slow initial development pace - Need for comprehensive testing tool knowledge[13,14]	Quality-critical applications, teams focused on reducing bugs, projects requiring high reliability[13,14]	Red-Green-Refactor, Unit Tests, Test Cases, Continuous Testing, Test Coverage

8. Requirements Engineering

Definition

Requirements engineering is "the subset of systems engineering concerned with discovering, developing, tracing, analyzing, qualifying, communicating and managing requirements that define the system at successive levels of abstraction"8.

Requirements Engineering Process

1. Requirements Elicitation: Discovering requirements by interacting with stakeholders8
2. Requirements Analysis: Detailed analysis of gathered requirements8
3. Requirements Specification: Converting requirements into standard form8
4. Requirements Validation: Checking requirements define what customer wants8
5. Requirements Management: Managing changes and traceability8

Types of Requirements

- Functional Requirements: What the system should do8
- Non-functional Requirements: Quality attributes and constraints8
    - Product requirements (performance, reliability, usability)
    - Organizational requirements (standards, implementation)
    - External requirements (regulatory, ethical)
- Domain Requirements: Specific to application domain8

Requirements Traceability

Types of traceability analysis8:

- Impact Analysis: "What if this was to change?"
- Derivation Analysis: "Why is this here?"
- Coverage Analysis: "Have I covered everything?"

Related Exam Questions with Answers:
Q: Explain what is an SRS
A: SRS (Software Requirements Specification) is a document that completely describes what the software system will do without describing how it will do it4. It includes:

- Functional requirements
- Non-functional requirements
- Interface requirements
- Design constraints
- Quality attributes

Q: Non-functional requirements are mainly classified as product, organizational and external requirements. Briefly explain these three types of requirements with examples
A: The three types are4:

1. Product requirements: Specify behavior of the product (e.g., performance, reliability, usability)
2. Organizational requirements: Derived from policies and procedures (e.g., process standards, implementation requirements)
3. External requirements: Arise from factors external to the system (e.g., regulatory requirements, ethical requirements)

Q: State two problems in requirement elicitation process
A: Two common problems are4:

1. Stakeholders don't know what they want: Users may have difficulty expressing their needs clearly
2. Conflicting requirements: Different stakeholders may have contradictory requirements

9. Software Testing

Testing Terminology

- Reliability: Measure of success with which observed behavior confirms to specification9
- Failure: Departure from required behavior9
- Fault (Bug): Defect in the system9
- Error: Human action that produces incorrect result9

Dealing with Errors

Approaches include9:

- Verification: Checking against specifications
- Modular Redundancy: Backup systems
- Patching: Fixing identified problems
- Testing: Planned creation of failures

Levels of Software Testing

1. Unit Testing: Individual subsystems by developers9

2. Integration Testing: Groups of subsystems and interfaces9
3. System Testing: Entire system against requirements9
4. Acceptance Testing: System evaluation by client9

Testing Approaches

- Static Analysis: Hand execution, walkthroughs, code inspection9
- Dynamic Analysis: Black-box and white-box testing9

Black-box Testing

Focus on input/output behavior9:

- Equivalence Partitioning: Divide inputs into equivalence classes
- Boundary Value Analysis: Test at boundaries
- Decision Table: Systematic test case generation

White-box Testing

Focus on code coverage9:

- Statement Testing: Test single statements9
- Decision Testing: Test all possible decisions9
- Loop Testing: Test loop execution scenarios9
- Path Testing: Ensure all paths are executed9

Acceptance Test Driven Development (ATDD)

ATDD is an agile system development approach that validates 'Building the Right Code' through executable specifications meaningful to business, developers, and testers9.
Related Exam Questions with Answers:
Q: "Testing can detect the presence of errors and their absence". Do you agree with this statement? Justify your answer
A: I disagree with this statement3. Testing can only detect the presence of errors, not their absence. As Dijkstra said, "Testing can show the presence of bugs but never their absence"9. Complete testing is impossible because:

- Infinite number of possible inputs
- Infinite number of possible execution paths
- Testing is always done with finite resources and time

Q: Compare and contrast black-box testing vs white box testing
A:
Black-box testing9:

- Focus on input/output behavior
- No knowledge of internal structure
- Tests functionality against specifications
- Uses equivalence partitioning, boundary value analysis

White-box testing9:

- Focus on internal code structure
- Requires knowledge of code
- Tests code coverage and logic paths
- Uses statement, decision, and path testing

Q: Explain briefly, Acceptance Test Driven Development (ATDD)
A: ATDD is an agile development approach where9:

- Tests are written before code implementation
- Focuses on building the right code for customer needs
- Uses executable specifications meaningful to business, developers, and testers
- Validates requirements through automated acceptance tests
- Ensures customer requirements are met through continuous testing

Q: State two goals of program testing
A: Two main goals are4:

1. Demonstrate software meets requirements: Show the software works as intended
2. Discover defects: Find errors, faults, and failures in the software

Q: Differentiate Verification and Validation, Software Inspection and Software Testing, Release Testing and System Testing
A:
Verification vs Validation4:

- Verification: "Are we building the product right?" - checking against specifications
- Validation: "Are we building the right product?" - checking against user needs

Software Inspection vs Software Testing4:

- Inspection: Static analysis of code/documents without execution
- Testing: Dynamic analysis by executing the software

Release Testing vs System Testing4:

- System Testing: Testing complete integrated system against requirements
- Release Testing: Testing a release of system for deployment to customers

10. DevOps and CI/CD

DevOps Overview

DevOps is a set of practices that bridges the gap between software development (Dev) and IT operations (Ops)10. It aims to shorten the software development lifecycle and provide continuous delivery with high software quality.

Key Differences: Agile vs CI/CD vs DevOps

- Agile: Focuses on development process and methodology10
- CI/CD: Focuses on automation of build, test, and deployment10
- DevOps: Focuses on culture and collaboration between Dev and Ops teams10

CI/CD Components

- Continuous Integration (CI): Frequent code merging with automated testing10
- Continuous Delivery (CD): Automated preparation for production release10
- Continuous Deployment (CD): Automated release to production10

DevOps Benefits

- Breaks down walls between development and operations10
- Unifies teams for better, faster outcomes10
- Focuses on both agility and stability10
- Reduces time to market10

Related Exam Questions with Answers:
Q: Explain the relationship between Agile, CI/CD, and DevOps
A: The three are complementary but different10:

- Agile provides the development methodology and process framework
- CI/CD provides the technical practices and automation tools
- DevOps provides the cultural foundation and collaboration model
  Together they enable faster, more reliable software delivery

Q: Describe the benefits of implementing DevOps practices
A: Key benefits include10:

- Faster time to market through automation
- Improved collaboration between teams
- Higher quality software through continuous testing
- Better reliability and stability in production
- Reduced manual errors through automation
- Faster recovery from failures

11. Software Reuse and Maintenance

Software Reuse

Advantages4:

- Reduced development time and cost
- Increased reliability through proven components
- Lower risk

Disadvantages4:

- Increased maintenance costs
- Lack of control over functionality
- Version management complexity

Software Change Requests

Common reasons include4:

- Changing business requirements
- Technology evolution
- Bug fixes and performance improvements
- Regulatory compliance changes

Related Exam Questions with Answers:
Q: State one advantage and one disadvantage of using reusable software
A:
Advantage: Reduced development time and cost as existing components don't need to be redeveloped4
Disadvantage: Increased maintenance costs due to dependency on external components4
Q: State two reasons that raise software change requests
A: Two common reasons are4:

1. Changing business requirements: Business needs evolve over time

2. Bug fixes: Defects discovered during operation need correction

Q: State two types of costs that are incurred when re-using software
A: Two types of costs are4:

1. Search and assessment costs: Time spent finding and evaluating reusable components
2. Adaptation and integration costs: Effort to modify and integrate components into the system

12. System Architecture and Design

Architectural Views

Four useful perspectives for designing and documenting system architecture4:

1. Logical View: Functional requirements
2. Process View: Non-functional requirements
3. Development View: Software management
4. Physical View: System topology

Ensuring System Characteristics

Through architectural design4:

- Security: Access control, encryption, authentication mechanisms
- Performance: Load balancing, caching, efficient algorithms
- Availability: Redundancy, failover mechanisms, monitoring

Object-Oriented Design

Considerations for suitability4:

- Not suitable for all software development
- Best for systems with clear object models
- May not be optimal for procedural or functional systems

Related Exam Questions with Answers:
Q: Explain how system characteristics (Security, Performance, Availability) can be ensured with architectural design
A:
Security4: Implement layered security architecture with authentication, authorization, encryption, and secure communication protocols

Performance4: Use load balancing, caching strategies, efficient algorithms, and optimized data structures
Availability4: Design redundant systems, implement failover mechanisms, and use monitoring for proactive issue detection
Q: What are the four views or perspectives useful when designing and documenting a system's architecture?
A: The four views are4:

1. Logical View: Shows the key abstractions and their relationships
2. Process View: Shows how the system is composed of communicating processes
3. Development View: Shows how software is decomposed for development
4. Physical View: Shows the system hardware and how software is distributed

Q: Consider the statement, "Object-Oriented Design Process is suitable for any software development." Do you agree? Justify your answer
A: I disagree with this statement4. Object-oriented design is not suitable for all software development because:

- Some systems are better modeled with functional or procedural approaches
- Real-time systems may require different design paradigms
- Mathematical or scientific applications may not benefit from OO design
- The problem domain should determine the most appropriate design approach

13. Quality Assurance and Metrics

Quality Definition

Quality is "fitness for purpose" or conformance to requirements – providing something that satisfies the customer while ensuring all stakeholder needs are considered8.

Usability Metrics

Two metrics for measuring ease of use4:

- Time to complete tasks
- Error rate during task completion
- User satisfaction scores
- Learning curve steepness

Related Exam Questions with Answers:
Q: State two metrics that can be used to measure ease of use of a system
A: Two metrics are4:

1. Task completion time: How long it takes users to complete specific tasks
2. Error rate: Number of errors users make while using the system

14. Project Management and Cost Estimation

Cost Estimation Risks

Four ways to reduce risk in cost estimates[3]:

1. Use multiple estimation techniques
2. Include contingency buffers
3. Regular re-estimation during project
4. Use historical data from similar projects

Software Project Pricing

Circumstances for charging higher prices[3]:

- Unique or specialized requirements
- High-risk projects
- Tight deadlines
- Proprietary technology requirements
- Limited vendor availability

Related Exam Questions with Answers:
Q: Cost estimates are inherently risky, irrespective of the estimation technique used. Suggest four ways in which the risk in a cost estimate can be reduced
A: Four ways to reduce risk are[3]:

1. Use multiple estimation techniques: Compare results from different methods
2. Include contingency buffers: Add extra time and resources for unexpected issues
3. Regular re-estimation: Update estimates as more information becomes available
4. Use historical data: Base estimates on data from similar completed projects