Software Engineering Revision

1.) Distinguish between software engineering and system engineering.

[2 marks]

- System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering.
- Software engineering is part of this more general process and is concerned with all aspect of the development and evolution of the computer system or other complex system.
- Software engineers are responsible for designing, developing, and maintaining software applications.
- System engineers are responsible for designing and maintaining complex systems and ensuring they work seamlessly.

b. Software engineers should adopt a systematic and organized approach to their work and use

appropriate **tools and techniques** depending on the problem to be solved, the **development constraints** and the **resources** available. Explain the meaning of underlined words and phrases.

[3 marks]

Tools and Techniques:

Tools: These are specific software applications or hardware devices that aid in different stages of development. For example, code editors like Visual Studio Code or Sublime Text assist programmers in writing and editing code.

Techniques: These are methodologies, practices, and processes followed throughout the development lifecycle. Agile methodologies like Scrum or Kanban focus on iterative development with short sprints and continuous feedback. Design patterns offer reusable solutions for common software problems. Testing techniques like unit testing, integration testing, and system testing ensure thorough software functionality validation.

development constraints

Constraints are limitations or restrictions that impact the development process. They can be internal (budget, deadlines, team expertise) or external (hardware limitations, regulations, user expectations).

Resources

Resources are assets available for software development, including human resources (developers, testers, project managers), financial resources (budget), technological resources (hardware, software licenses), and information resources (requirements documents, technical documentation). Optimizing resource utilization is crucial for efficient software development.

c. Briefly outline the generic activities common in all software processes.

[2 marks]

The generic activities common in all software processes are:

1. **Specification**: This activity involves defining the main functionalities of the software as per the customer requirements and expectations.
2. **Development**: This activity involves the production of the software system.
3. **Validation and verification**: This activity involves checking that the software is what the customer wants.
4. **Evolution**: This activity involves changing the software in response to changing demands.
5. **Planning:** Establish engineering work plan, describes technical risk, lists resources requirements, work produced and defines work schedule.

d. Software process model is simplified representation of a software process, presented from a specific perspective. Name and briefly explain these process perspectives. [3 marks]

Waterfall Model:

- Perspective: Sequential, linear order of phases (requirements, design, implementation, testing, deployment, maintenance).
- Focus: Planning and adherence to strict stages, clear deliverables between phases.
- Pros: Simple to understand, good for well-defined projects.
- Cons: Inflexible, difficult to adapt to changing requirements, high risk of errors later in the process.

2. Iterative Model:

- Perspective: Cyclical development with repeating cycles of requirements, design, implementation, and testing.
- Focus: Early and continuous feedback, delivering working software increments in each iteration.
- Pros: Flexible, adapts well to changing requirements, reduces risk of major errors.
- Cons: Requires strong planning and communication, may not be suitable for projects with strict deadlines.

3. Incremental Model:

- Perspective: Delivers working software in stages with increasing functionality with each increment.
- Focus: Building and delivering core features first, then adding functionalities step-by-step.
- Pros: Manages complexity by delivering smaller working versions, reduces risk of late project failures.
- Cons: Requires good requirements planning and prioritization, may involve rework if early decisions impact later increments.

4. Prototyping Model:

- Perspective: Develops a preliminary version of the software to gather user feedback and refine requirements.
- Focus: Early user involvement, validating concepts and functionalities before full development.
- Pros: Reduces risk of building the wrong product, improves user satisfaction.
- Cons: Can be time-consuming, may not reflect the final product functionality.

5. Spiral Model:

- Perspective: Combines iterative and risk-driven approach, with cycles of planning, risk assessment, development, and evaluation.
- Focus: Managing risks and adapting to changing needs throughout the development lifecycle.
- Pros: Flexible, suitable for complex projects with high risks, continuous risk mitigation.
- Cons: Requires experienced project management, can be complex to implement.

e. Explain any TWO attributes of a good software product.

[2 marks]

Usability: This refers to how easy and intuitive the software is to learn and use for its target audience.

Reliability: This refers to the consistency and dependability of the software in performing its intended functions.

Maintainability:

Correctness:

f. Explain any TWO key challenges facing software engineering industry today.

[2 marks]

Talent Shortage and Skills Gap:

- Demand: The ever-growing demand for software applications across various industries outpaces the supply of qualified software engineers.

Security and Privacy Concerns:

- Growing threats: As software becomes more complex and interconnected, it becomes increasingly vulnerable to cyberattacks, data breaches, and privacy violations. The rise of ransomware, malicious bots, and sophisticated hacking techniques poses a significant challenge to software security.

Rapidly evolving technologies: The constant emergence of new technologies and frameworks requires developers to continuously learn and adapt.

Managing project complexity: Large software projects often involve complex architectures, diverse teams, and intricate dependencies, making them challenging to manage effectively.

Meeting user expectations: Delivering software that meets the ever-increasing demands and expectations of users can be difficult, and user testing and feedback loops become crucial.

g. Explain any TWO roles of Code of Ethics to software engineers

12 Marks]

Guiding Ethical Decision-Making:

- Provides a framework: The Code establishes a set of principles, such as public interest, competence, honesty, and fairness, that serve as a compass for navigating complex ethical dilemmas.
- Raises awareness: It highlights potential ethical challenges that software engineers might encounter in their work, such as privacy concerns, biased algorithms, or intellectual property issues.
- Promotes critical thinking: By encouraging reflection and discussion on ethical matters, the Code helps engineers develop the ability to critically analyze situations and make informed decisions, even in the absence of clear-cut answers.

2. Promoting Professional Responsibility:

- Establishes accountability: The Code sets expectations for ethical conduct and professionalism, holding engineers accountable for their actions and decisions.
- Fosters trust and integrity: By adhering to the Code, engineers build trust with stakeholders and contribute to a positive reputation for the profession.
- Supports collaboration and teamwork: The Code promotes a culture of cooperation and mutual respect, encouraging engineers to work together towards shared ethical goals.

h. Explain the meaning of scope Creep and explain how it can affect a software production.

[3 Marks]

1. What is Scope Creep?

- **Definition:** Scope creep occurs when additional features, functionalities, or tasks are added to a software project after the initial scope has been agreed upon.

- Causes: It can be caused by various factors, like changing client needs, miscommunication, underestimating complexity, or even "feature envy" where developers get excited about adding new features.

2. Impact of Scope Creep on Software Production:

- **Project Delays:** Adding new tasks inevitably requires more time and resources, pushing back deadlines and potentially making the original timeline unrealistic.
- **Increased Costs:** More work translates to higher expenses, potentially exceeding the planned budget and straining financial resources.
- **Reduced Quality:** Rushing to meet new deadlines or cramming extra features can lead to compromised quality, bugs, and performance issues.
- **Team Morale:** Unrealistic expectations and constant changes can demotivate developers and create frustration within the team.

3. Preventing Scope Creep:

- **Clearly define the scope:** Precisely documenting the initial features, functionalities, and deadlines in a written agreement is crucial.
- **Open communication**: Foster regular communication with all stakeholders, including clients, developers, and project managers, to address concerns and manage expectations.
- **Change management process:** Establish a formal process for handling changes, ensuring proper evaluation and prioritization before adding new features.
- **Regular monitoring and reporting:** Monitor progress, track any changes, and report potential scope creep issues early on to prevent them from snowballing.

i. What is meant by Computer Aided Systems Engineering (CASE) tools? What are the TWO main reasons why analysts rely on CASE-tools

[3 marks]

CASE TOOLS are tools that provide automated assistance for software development

<u>reasons why analysts rely on CASE-tools</u>

-time and cost of software development is reduced.

-Quality system development.

j. Discuss any two error common in system engineering according to peter DeGrace and Leslie Stahl.

[3 marks]

Peter DeGrace and Leslie Stahl, in their book "The Olduvai Imperative: CASE and the State of Software Engineering Practice" [1], have identified several common errors in system engineering. Here are two of them:

1. **Misunderstanding the problem**: This error occurs when the problem to be solved is not well understood. This can lead to the development of a system that does not meet the needs of the users or stakeholders. To avoid this error, it is important to have a clear understanding of the problem before starting the development process.
2. **Poor communication**: This error occurs when there is a lack of communication between the different stakeholders involved in the development process. Poor communication can lead to misunderstandings, delays, and errors. To avoid this error, it is important to establish clear lines of communication between all stakeholders and to ensure that everyone is on the same page.

k. Compare and contrast the Greece and Roman cultures explain how they affected the quality

13 marks]

Greek and Roman cultures have had a profound impact on the world, and their influence can still be felt today. While there are many similarities between the two cultures, there are also some significant differences.
One of the most significant differences between Greek and Roman cultures is their form of government. The Greeks were known for their direct democracy, where all citizens had a say in the government. In contrast, the Romans had a representative republic, where citizens elected officials to represent them in government [1].
Another difference between the two cultures is their approach to art. The Greeks were known for their idealized sculptures, which depicted gods, statesmen, and generals as physically perfect beings. In contrast, the Romans developed a more naturalistic approach to their art, which portrayed people with physical quirks and nuances of expression that made them more human [1].
Both cultures had a significant impact on the quality of life. The Greeks were known for their contributions to philosophy, mathematics, and science, which laid the foundation

for modern Western thought. The Romans, on the other hand, were known for their engineering feats, such as aqueducts and roads, which helped to improve the quality of life for people living in the empire [2].

L. explain way validation and verification is a necessary process in software requirement specification

Ensuring the Right Product is Built:

- Validation: Checks if the requirements actually address the needs and goals of the stakeholders. This prevents building a product that solves the wrong problem or misses crucial functionality.
- Verification: Makes sure the specified requirements are translated correctly into concrete design and implementation. This avoids building a product that meets the spec but not the actual user needs.

2. Catching Errors Early:

- Validation: Identifies flaws, inconsistencies, or ambiguities in the requirements document itself before development begins. Fixing these early saves time and cost compared to catching them later in the development cycle.
- Verification: Ensures the actual development accurately reflects the verified requirements. This minimizes bugs and rework caused by misinterpretations or discrepancies.

3. Improving Quality and User Satisfaction:

- Validation: Leads to a more complete and user-centric set of requirements, resulting in a higher quality product with greater user satisfaction.
- Verification: Confirms that the software actually does what it's supposed to, enhancing user experience and reducing frustration with unexpected behavior.

4. Risk Management and Cost Control:

- Validation: Helps identify potential risks and challenges early on, allowing for proactive mitigation strategies to avoid costly rework or project delays.
- Verification: Reduces the potential for bugs and defects, leading to smoother development and lower maintenance costs in the long run.

5. Communication and Collaboration:

- V&V activities involve stakeholders from different disciplines, fostering better communication and understanding of the project goals and requirements between developers, users, and other stakeholders.
- Clear and validated requirements provide a common reference point for everyone involved, leading to smoother collaboration and reduced misunderstandings.

QUESTION 2

A company is looking forward to develop a new copyrighted and open source software applications that can compete amongst the current social media platforms.

a. The company tasks you to lead the team carrying out feasibility study/ analysis. As a team leader, explain various items you will consider during analysis. In each case explain tools and techniques you will employ in order to deliver a complete feasibility study report [6marks]

1. Market and Competitor Analysis:

- Tools and techniques: Market research reports, competitor analysis tools (e.g., Similarweb, Alexa), social media listening tools (e.g., Brandwatch, Sprout Social), focus groups, surveys.
- Analyze: Current market trends in social media, identify user pain points and unmet needs, understand the strengths and weaknesses of existing platforms, assess potential market share and user base.

2. Technical Feasibility:

- Tools and techniques: Prototyping tools, software development frameworks, technical assessments by developers, feasibility estimates (e.g., function point analysis).
- Analyze: Evaluate the technical viability of developing the proposed platform, consider scalability, security, and performance requirements, assess available resources and technical expertise.

3. Financial Feasibility:

- Tools and techniques: Financial modeling tools, cost-benefit analysis, break-even analysis, funding strategy assessments.
- Analyze: Estimate the development and maintenance costs, project potential revenue streams (e.g., advertising, subscriptions), determine financial viability and identify potential funding sources.

4. Legal and Regulatory Considerations:

- Tools and techniques: Legal advice, copyright and intellectual property analysis, privacy and data security assessments.
- Analyze: Ensure compliance with relevant copyright and open-source licensing regulations, address data privacy concerns, identify potential legal risks and barriers.

5. Team and Organizational Feasibility:

- Tools and techniques: Skill gap analysis, team evaluation, project management assessments.
- Analyze: Assess the company's ability to build and manage the platform, evaluate existing team skills and identify potential resource needs, develop a realistic project timeline and resource allocation plan.

b. As Chief Analyst for initial investigation and analysis. Give a detailed outline of the different stages of requirements engineering, tools and techniques that you will embrace to deliver a complete and consistent requirements specification document (SRS) to the company.

[6 Marks]

1. Requirement Elicitation:

- Tools and Techniques: Interviews, questionnaires, workshops, use case analysis, observation of existing platforms.
- Information gathered: Stakeholder needs, motivations, pain points, existing workflows, competitor analysis.

2. Requirement Analysis:

- Tools and Techniques: Requirement prioritization, feasibility analysis, dependency mapping, data flow diagrams.
- Information gathered: Refining gathered requirements, validating against constraints, identifying inconsistencies, establishing relationships between requirements.

3. Requirement Documentation:

- Tools and Techniques: SRS templates, UML diagrams, requirement management tools (e.g., Jama Connect, JIRA).
- Information gathered: Documenting refined requirements with clear descriptions, acceptance criteria, examples, prioritization, and traceability to stakeholders.

4. Requirement Validation and Verification:

- Tools and Techniques: Reviews, walkthroughs, prototyping, user testing.
- Information gathered: Confirming requirements meet user needs, ensuring consistency, completeness, and testability.

5. Requirements Management:

- Tools and Techniques: Version control systems, traceability matrix, change management procedures.
- Information gathered: Tracking changes, maintaining consistency throughout development, communicating updates to stakeholders.

Delivering a complete and consistent SRS:

- Ensure clear, concise, and unambiguous language.
- Organize requirements logically by functionality, priority, or stakeholder concerns.
- Include visuals like diagrams and tables for better understanding.
- Define acceptance criteria for each requirement for testing purposes.
- Establish a version control system and change management process.

c. For each of the following categories, give a briefly explain of what it entails and the type of information you need to gather when you are investigating the requirements for the proposed software Applications by the company?

i. Functional requirements

[2 Marks]

- Describe what the software system does (e.g., search for users, post updates, share content).
- Focus on specific actions, calculations, and data manipulation features.
- Information needed: Detailed descriptions of functionalities, user interactions, inputs/outputs, dataflows.

ii. Nonfunctional requirements

[2 Marks]

- Describe how the system should perform (e.g., response time, security, usability, reliability).
- Focus on performance, usability, maintainability, and security aspects.
- Information needed: Performance thresholds, security constraints, user interface preferences, accessibility considerations.

Usability requirements

[2 Marks]

- Focus on making the system easy to learn, use, and navigate for users.
- Include elements like intuitiveness, consistency, efficiency, and error prevention.
- Information needed: User profiles, typical tasks, interaction patterns, feedback mechanisms, interface design preferences.

iv. Emergent requirements

[2 Marks]

- New requirements identified during development and not addressed initially.
- Can arise from user feedback, technical challenges, or changing needs.
- Information needed: Capture new requirements clearly, assess impact on existing requirements, manage change requests effectively.

QUESTION 3

a. What is meant by software process model and how does its choice affect the software product?

[3 marks]

A software process model is a structured approach to defining and managing the software development lifecycle. It outlines the sequence of activities, roles, and responsibilities involved in creating and maintaining software. Choice of the model has a significant impact on the software product in several ways:

- **Development Pace:** Different models offer varying levels of flexibility and adaptability. Waterfall emphasizes sequential phases, leading to potentially slower development. Agile embraces iteration and feedback, facilitating faster adaptation to changes.
- **Quality and Risk Management:** Waterfall offers upfront planning and control, potentially reducing later defects. Agile relies on continuous testing and feedback, potentially identifying issues earlier.
- **Project Predictability:** Waterfall provides a roadmap with clear stages and predefined deadlines. Agile embraces continuous evolution, making long-term predictions less straightforward.
- **Resource Utilization:** Waterfall may require higher upfront investment in planning and documentation. Agile focuses on flexible allocation based on evolving needs.

b. Assume you are working with Microsoft Corporation as a senior software developer. Explain with example various conditions under which you can/ will recommend the company to use the following software process models:

i. Waterfall model

[3 marks]

- Conditions: Well-defined project with stable requirements, high emphasis on security and compliance (e.g., developing a new Windows update).
- Example: Designing the core functionality for a new Office application.

ii. Incremental process model

[3 marks]

- Conditions: Project with evolving requirements, need for early user feedback, manageable release chunks (e.g., developing a new Azure feature).
- Example: Building a cloud storage app in phases, gradually adding features based on user testing.

Prototyping model

[3 marks]

- Conditions: High uncertainty about user needs, need for early validation of key features, complex user interface (e.g., designing a new social media platform).
- Example: Creating a prototype for a new productivity tool to get user feedback before full development.

iv. Agile methodology

[3 marks]

- Conditions: Dynamic project with rapidly changing requirements, need for continuous delivery and adaptation, responsive team (e.g., developing new Bing search features).
- Example: Building a new search engine algorithm using scrum sprints with continuous integration and testing.

c. When you are assessing a legacy/old system, you have to look at it from a business perspective and a technical perspective. From a business perspective, you have to decide whether the business really needs the system. From a technical perspective, you have to assess the quality of the system and its related support software and hardware. You then use a combination of the business value and the system quality to take one of the following informed decisions: scrap the system, re- engineer the system, replace the system, or continue the system's maintenance. Assume that you assessed four systems and the results of the assessment are as follows: System A: high quality, low business value, System B: high quality, high business value, System C: low quality, low business value and System D: low quality, high business value.

What would be your recommendations for each of these systems? Justify your decisions

- **System A (high quality, low business value):** Consider re-engineering if upgrading can significantly increase business value. Otherwise, scrap if maintenance costs outweigh benefits.
- **System B (high quality, high business value):** Continue maintenance and explore potential enhancements to further solidify its business value.
- **System C (low quality, low business value):** Scrap if costs and risks outweigh any potential benefits. Consider replacing if a similar, higher-value system exists.
- **System D (low quality, high business value):** Re-engineer if quality improvements can unlock the potential business value. Consider replacement if re-engineering costs are substantial or a better alternative system exists.

QUESTION 4

a. Differentiate between a milestone and a deliverable as used in the study of software

engineering. [2 marks]

Milestones:

- Significant checkpoints in the software development lifecycle, marking the completion of major phases or stages.
- Represent progress towards the final product without necessarily being tangible outputs.
- Example: Finishing system design documents, completing functional testing, launching the software in beta.

Deliverables:

- Concrete outputs produced at specific points in the project, showcasing completed work.
- Can be documents, code modules, prototypes, or functional features.

- Example: Requirements document, user interface mockups, tested modules, a deployed working feature.

Key differences:

- Tangibility: Milestones are progress markers, deliverables are tangible outputs.
- Specificity: Milestones mark broader phases, deliverables focus on specific pieces of work.
- Evaluation: Milestones assess project progress, deliverables assess completed work

b. There has been a notion that software design is expensive and require time. Justify this claim.

[2 marks]

- **Complexity:** Good design involves considering multiple factors like functionality, performance, maintainability, and scalability, making it intricate and time-consuming.
- **Iteration and Refinement**: Design often involves repeated cycles of prototyping, reviewing, and refining, incurring additional effort.
- **Early Impact:** Design decisions made early on have significant downstream implications, making it crucial to get them right, requiring careful analysis and attention to detail.
- **Abstraction and Communication:** Effectively translating abstract design concepts into concrete implementations for developers can be challenging and require effective communication.

b. Discus the THREE major constraints of software project, in each cases indicate how it affects software quality.

c. **Schedule:** Tight deadlines can pressure developers to prioritize speed over quality, leading to rushed code and potential defects.

d. **Budget:** Limited resources can lead to insufficient testing, incomplete features, and compromises in implementation, impacting overall quality and user experience.

e. **Scope**: Scope creep, where additional features are added late in the project, can stretch resources and compromise quality due to rushed development and integration.

d. What is software design and what are stages involved?

[6 marks]

Software design is the process of defining the blueprint for software, describing its components, relationships, and behavior. It involves translating requirements into a practical and feasible technical solution.

Stages of Software Design:

1. Architectural Design: Defines the overall system architecture, including components, modules, and their interactions.
2. Detailed Design: Breaks down the system into smaller components, specifies data structures, algorithms, and interfaces.
3. User Interface Design: Focuses on designing the user interface elements for optimal usability and user experience.

e. Explain the following software design strategies:

i. Functional design

[4 marks]

- Decomposes the system into modules based on functionalities, defining inputs, outputs, and processing steps.
- Promotes modularity, reusability, and easier maintenance.
- Tools: Data flow diagrams, function decomposition trees.

ii. Object-oriented design

- Organizes software around objects that encapsulate data and behavior.
- Promotes data encapsulation, code reusability, and inheritance-based flexibility.
- Tools: UML diagrams, object-oriented programming languages.

f. Explain cohesion and coupling as a software design attributes.

g. Cohesion: Measures the degree to which elements within a module are related and contribute to a single purpose. High cohesion leads to well-organized, understandable modules.

h. Coupling: Measures the degree of interdependence between modules. Low coupling makes modules independent and easier to maintain.