Software Engineering: A Comprehensive Study Guide

Review of Core Concepts

Before diving into specific details, ensure you understand these fundamental concepts:

1. **What is Software Engineering?** Understand the definition of software engineering as a discipline, its relationship to computer science, and its goals. Consider what differentiates "good" software from just any code.
2. **Software Development Life Cycle (SDLC):** Be familiar with the general phases of the SDLC (requirements, design, implementation, testing, deployment, maintenance). Know that different models (Waterfall, Iterative, Prototyping, Evolutionary, Spiral) exist and when each might be appropriate.
3. **Requirements Engineering:** Understand the processes of eliciting, analysing, specifying, validating, and managing requirements. Know the difference between functional and non-functional requirements and the importance of a well-defined SRS document.
4. **Software Design:** Know the meaning and goals of software design. Know the different levels of design abstraction (architectural, high-level, detailed). Be familiar with design approaches like top-down, bottom-up, structured design, function-oriented design, and object-oriented design.
5. **Software Testing:** Grasp the purpose and different levels of testing (unit, integration, system, acceptance). Know the difference between verification and validation. Be familiar with different testing techniques (black-box, white-box, etc.).
6. **Software Maintenance:** Understand the different types of software maintenance (corrective, perfective, adaptive, preventive) and their purpose.
7. **Software Configuration Management:** Be familiar with the concepts of version control, baselines, and change management.
8. **Dependability and Security:** Understand the basic concepts of system dependability (reliability, availability, safety, security) and related techniques like fault tolerance, hazard analysis, and security policies.

Quiz

Answer the following questions concisely (2-3 sentences each).

1. What are the two important techniques used in Software Engineering principles to reduce problem complexity?

2. Explain the purpose of a Software Requirements Specification (SRS) document.
3. Describe the main goal of the Design phase in the Classical Waterfall Model.
4. What is the core difference between Verification and Validation in software testing?
5. Explain the concept of "cohesion" in the context of software design.
6. Describe what functional and nonfunctional requirements are using an example of a banking application?
7. Explain what a System Analyst does and the significance of explicit content presentation in user interface design.
8. What are Lehman's Laws of Software Evolution and give one example?
9. What are the three known complexity metrics? Explain what is being measured.
10. Describe three activities of Software Re-engineering.

Quiz Answer Key

1. Abstraction and Decomposition are used in Software Engineering principles. Abstraction simplifies complex systems by focusing on essential details while hiding unnecessary complexity. Decomposition breaks down large problems into smaller, manageable parts.
2. An SRS document details customer requirements and identifies functional, non-functional and implementation goals. Developing the SRS ensures a structured approach and also ensures the software developer will know what is expected when developing the project.
3. The Design phase aims to transform the requirements specified in the SRS document into a suitable structure for implementation. This includes selecting design approaches like traditional or object-oriented design.
4. Verification determines whether the software output of one phase conforms to that of its previous phase. Validation focuses on whether the fully developed system conforms to its requirements specification and meets customer needs.
5. Cohesion refers to the degree to which the elements inside a module belong together. High cohesion implies that the elements are strongly related and contribute to a single, well-defined purpose.
6. Functional requirements specifies what actions and the process flow of an application that are to be preformed. An example for banking application would be when a user can create accounts or transfer funds. Nonfunctional

requirements are not directly functional but expected characteristics of the application, an example would be speed or the security measures.

7. A system analyst analyzes requirements and ensures accuracy and proper documentation. Explicit content presentation focuses on clear and accessible presentation of software content to prevent user confusion.
8. Lehman's Laws are based on the observation that software changes over time and involves making changes to the software system in order to meet new requirements. A software product would become progressively less satisfactory to its users unless it is maintained and adapted to the changes.
9. The three known complexity metrics are Halstead's Complexity Measures, Cyclomatic Complexity measures and Software design complexity. Halstead's measures software complexity based on the program's implementation. The Cyclomatic measure quantifies complexity based on decision-making constructs in the program. And lastly, Software design complexity measures irregular interactions in the software system.
10. Software re-engineering is a process of software development to improve the maintainability of software. Three activities are reverse engineering, code restructuring and forward engineering. Reverse engineering extracts data while code structuring begins with reverse engineering and forward engineering is used to re-engineer and re-design the current software.

Essay Questions

Consider these essay questions to test your in-depth understanding. While answers are not provided, you should draw heavily from your source material and be able to synthesise information across different areas.

1. Discuss the advantages and disadvantages of different Software Development Life Cycle (SDLC) models (Waterfall, Iterative, Prototyping, Spiral). Under what circumstances is each model most appropriate?
2. Explain the importance of requirements engineering in the software development process. Describe the key activities involved in requirements elicitation, analysis, and specification. Provide real-world examples to illustrate your points.
3. Compare and contrast structured design and object-oriented design approaches. How do these approaches differ in terms of their fundamental principles and techniques? Provide a sample design problem and show how it can be addressed using both techniques.

4. Discuss the different levels of software testing and the types of defects each level is designed to uncover. Provide a detailed example of how to design test cases using black-box testing techniques.
5. Analyse the ethical and professional responsibilities of software engineers in ensuring the dependability and security of critical systems. What are the potential consequences of failing to meet these responsibilities?

Glossary of Key Terms

- **Abstraction:** Simplifying complex systems by focusing on essential details while hiding unnecessary complexity.
- **Agile Development:** A software development methodology that emphasises iterative development, collaboration, and rapid response to change.
- **Availability:** The probability that a system will be operational when a demand is made for service.
- **Black-Box Testing:** Testing software without knowledge of its internal structure or code.
- **Cohesion:** The degree to which the elements inside a module belong together.
- **Coupling:** The degree of interdependence between software modules.
- **Cyclomatic Complexity:** A software metric that measures the complexity of a program by quantifying the number of independent paths through its control flow graph.
- **Decomposition:** Breaking down a complex problem into smaller, more manageable parts.
- **Dependability:** A system's ability to deliver services that can justifiably be trusted.
- **DFD (Data Flow Diagram):** A graphical representation of the flow of data through an information system.
- **Fault Tolerance:** The ability of a system to continue operating correctly despite the presence of faults.
- **Functional Requirements:** Statements of services the system should provide, detailing how the system should react to particular inputs and behave in particular situations.
- **Modularity:** The degree to which a system's components may be separated and recombined.
- **Non-Functional Requirements:** Constraints on the services or functions offered by the system, such as timing constraints, constraints on the development process, standards, etc.

- **POFOD (Probability of Failure on Demand):** The probability that the system will fail when a service request is made.
- **Reliability:** The probability of failure-free system operation for a specified time in a specified environment for a specified purpose.
- **Requirements Engineering:** The process of defining, documenting, and maintaining requirements in the engineering design process.
- **Security:** The ability of the system to protect itself against accidental or deliberate intrusion.
- **SDLC (Software Development Life Cycle):** A conceptual model that describes the stages involved in a software development project, from initial feasibility study to maintenance of the completed application.
- **Software Engineering:** An engineering branch associated with the development of software products using well-defined scientific principles, methods, and procedures.
- **SRS (Software Requirements Specification):** A document that contains customer requirements, identifying the functional, non-functional requirements, and goals of implementation for a software system.
- **UML (Unified Modeling Language):** A standard language for specifying, visualizing, constructing, and documenting the artefacts of software systems.
- **Validation:** Checking that the software is what the user really requires. "Are we building the right product?"
- **Verification:** Checking that the software conforms to its specification. "Are we building the product right?"
- **White-Box Testing:** Testing software with knowledge of its internal structure and code.