Software Architecture - Unit 1 Notes

1. What is Software Architecture? How software architecture represents a System's earliest set of design decisions?

Software Architecture refers to the fundamental structure of a software system, encompassing its components, their interactions, and the guiding principles behind its design and evolution. It is a high-level abstraction that outlines how a system is organized and how different elements work together to achieve desired functionality.

Representation of Earliest Design Decisions:

- Defines the system's structure before coding begins.
- Establishes the blueprint for developers, guiding implementation and integration.
- Captures decisions about modules, data flow, communication protocols, performance goals, and security.
- Ensures early identification of risks and system constraints.
- Provides a foundation for stakeholders to evaluate feasibility and cost-effectiveness.

2. Different Models of Software Development and Issues

a) Waterfall Model:

- Linear sequential process: Requirements → Design → Implementation → Testing → Maintenance.
- *Issues:* Inflexibility to accommodate changes, late testing stage, poor adaptability to evolving needs.

b) Iterative Model:

- Development in repeated cycles, with feedback incorporated in each iteration.
- *Issues:* Requires significant planning, risk of scope creep, higher cost.

c) Spiral Model:

- Combines iterative development with risk assessment. Each phase begins with objectives, risk analysis, prototyping, and then evaluation.
- *Issues:* Complex to manage, costly for small projects, demands strong risk analysis skills.

d) Agile Model:

- Incremental and iterative approach emphasizing collaboration, flexibility, and customer feedback.
- *Issues:* Difficult for large, distributed teams, requires high customer involvement, lacks detailed documentation.

3. Spiral Model in Software Development

The Spiral Model, introduced by Barry Boehm, combines elements of iterative development and systematic risk assessment. The process is visualized as a spiral with four main phases:

1. Objective setting: Identify goals, alternatives, and constraints.
2. Risk analysis: Identify and resolve risks.
3. Development & validation: Develop prototype, code, and test.
4. Planning: Review and plan next iteration.

Advantages:
- Strong risk management.
- Early detection of design flaws.
- Supports customer feedback and prototyping.
- Suitable for large, complex projects.

Disadvantages:
- Expensive and complex to implement.
- Requires expertise in risk management.
- Not suitable for small projects due to cost.

4. Software Components and Connectors

Software Components: Independent units of software with well-defined interfaces that encapsulate functionality. Examples: user interface modules, database modules, APIs.

Connectors: Define the communication and coordination among components. Examples: procedure calls, message queues, data streams, network protocols.

Importance:
- Promote modularity and reusability.
- Enhance scalability and maintainability.
- Enable flexible system integration.

5. Need of Integration in Software Development Environments

Integration is the process of combining different modules and tools into a unified environment to improve development efficiency and consistency.

Need:
- Ensures interoperability among different tools like compilers, debuggers, and version control.
- Reduces redundancy by enabling shared data and repositories.
- Facilitates Continuous Integration (CI) and Continuous Deployment (CD).
- Provides better collaboration between teams working on different modules.

Examples:
- Integrated Development Environments (IDEs) like Eclipse, Visual Studio.
- DevOps pipelines integrating tools like Git, Jenkins, Docker, Kubernetes.

6. Software Quality Model and McCall's Model

Software Quality Model: A framework that defines attributes to assess and ensure the quality of software products, such as functionality, reliability, usability, efficiency, maintainability, and portability.

McCall's Model:

Proposed by James A. McCall, this model evaluates software quality across three major perspectives:

1. **Product Operation:** Focuses on runtime qualities like correctness, reliability, efficiency, integrity, usability.
2. **Product Revision:** Measures maintainability aspects including maintainability, flexibility, testability.
3. **Product Transition:** Considers adaptability aspects like portability, reusability, interoperability.

Significance:

- Provides a structured way to measure and ensure software quality.
- Helps in identifying trade-offs between different quality factors.
- Acts as a guideline for both development and evaluation phases.