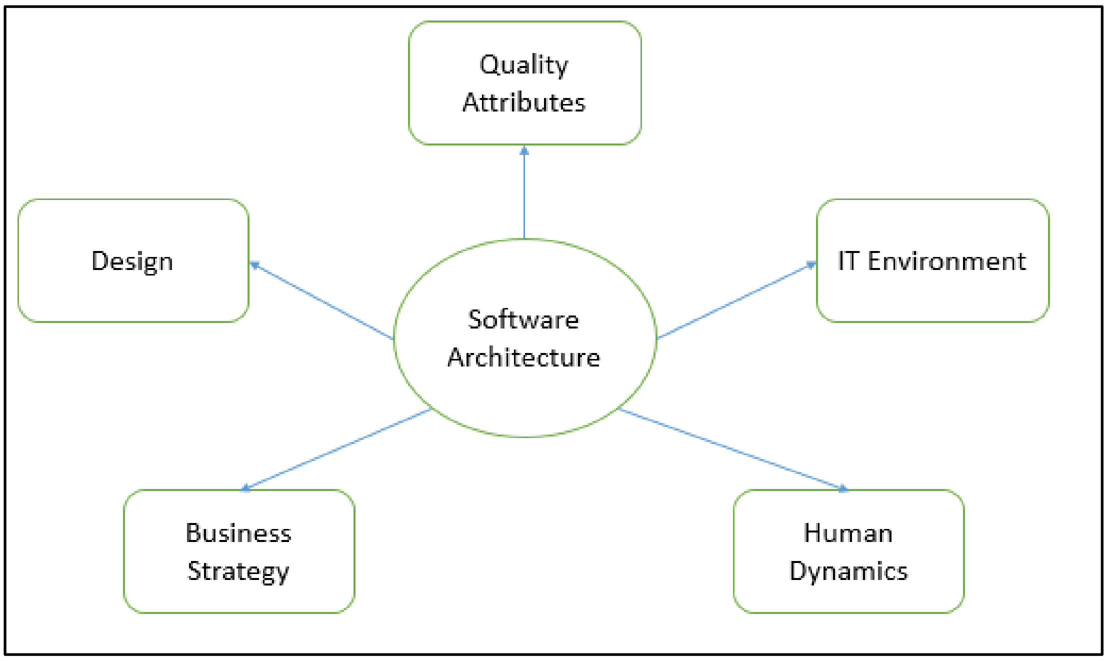
Table of Contents

1. Software Architecture and Design – Introduction

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design is a process that includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In **Design**, functional requirements are accomplished.

Software Architecture

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.

- It involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of:

    - Selection of structural elements and their interfaces by which the system is composed.

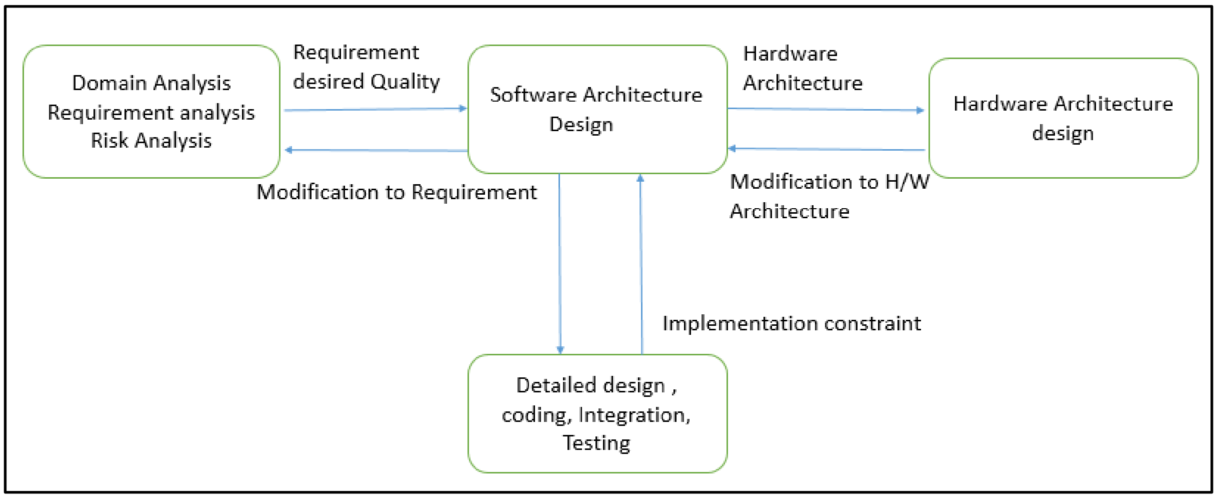
- Behavior as specified in collaborations among those elements.
- Composition of these structural and behavioral elements into large subsystem.
- Architectural decisions align with business objectives.
- Architectural styles that guide the organization.

Software Design

Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows:

- To negotiate system requirements, and to set expectations with customers, marketing and management personnel.
- Act as a blueprint during the development process.
- Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.


Domain Analysis
Requirement analysis
Risk Analysis
Requirement
desired Quality
Modification to Requirement
Software Architecture Design
Hardware Architecture
Modification to H/W Architecture
Hardware Architecture design
Implementation constraint
Detailed design , coding, Integration, Testing

Goals of Architecture

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements. Some of the other goals are as follows:

- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.

- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization's market position.
- Improve quality and functionality offered by the system.
- Improve external confidence in either the organization or system.

Limitations

Software architecture is still an emerging discipline within software engineering. It has the following limitations:

- Lack of tools and standardized ways to represent architecture
- Lack of analysis methods to predict whether architecture will result in an implementation that meets the requirements.
- Lack of awareness of the importance of architectural design to software development
- Lack of understanding of the role of software architect and poor communication among stakeholders.
- Lack of understanding of the design process, design experience and evaluation of design

Role of Software Architect

A Software Architect provides a solution that the technical team can create and design for the entire application. A software architect should have expertise in the following areas:

Design Expertise

- Expert in software design, including diverse methods and approaches such as object-oriented design, event-driven design, etc.
- Lead the development team and coordinate the development efforts for the integrity of the design.
- Should be able to review design proposals and tradeoffs among them.

Domain Expertise

- Expert on the system being developed and plan for software evolution.
- Assist in the requirement investigation process assuring completeness and consistency.
- Coordinate the definition of domain model for the system being developed.

Technology Expertise

- Expert on available technologies that helps in the implementation of the system.
- Coordinate the selection of programming language, framework, platforms, databases, etc.

Methodological Expertise

- Expert on software development methodologies that may be adopted during SDLC (Software Development Life Cycle).
- Choose the appropriate approaches for development that helps the entire team.

Hidden Role of Software Architect

- Facilitates the technical work among team members and reinforcing the trust relationship in the team.
- Information specialist who shares knowledge and has vast experience.
- Protect the team members from external forces that would distract them and bring less value to the project.

Deliverables of the Architect

- A clear, complete, consistent, and achievable set of functional goals
- A functional description of the system, with at least two layers of decomposition
- A concept for the system
- A design in the form of the system, with at least two layers of decomposition
- A notion of the timing, operator attributes, and the implementation and operation plans
- A document or process which ensures functional decomposition is followed, and the form of interfaces is controlled

Quality Attributes

Quality is a measure of excellence or the state of being free from deficiencies or defects. Quality attributes are system properties that are separate from the functionality of the system.

Implementing quality attributes makes it is easier to differentiate a good system from a bad one. Attributes are overall factors that affect runtime behavior, system design, and user experience. They can be classified as follows:

Static Quality Attributes

Reflect the structure of system and organization, directly related to architecture, design, source code. They are invisible to end-user, but affect the development and maintenance cost, e.g.: modularity, testability, maintainability, etc.

Dynamic Quality Attributes

Reflect the behavior of the system during its execution. They are directly related to system's architecture, design, source code and also the configuration, deployment parameters, environment, and platform.

They are visible to the end-user and exist at runtime, e.g.: throughput, robustness, scalability, etc.

Quality Scenarios

Quality scenarios specify how to prevent a fault from becoming a failure. They can be divided into six parts based on their attribute specifications:

- **Source** An internal or external entity such as people, hardware, software, or physical infrastructure that generates the stimulus.
- **Stimulus** A condition that needs to be considered when it arrives on a system.
- **Environment** The stimulus occurs within certain conditions.
- **Artifact** A whole system or some part of it such as processors, communication channel, persistent storage, processes etc.
- **Response** An activity undertaken after the arrival of stimulus such as detect faults, recover from fault, disable event source etc.
- **Response measure** Should measure the occurred responses so that the requirements can be tested.

Common Quality Attributes

The following table lists the common quality attributes a software architecture must have.

Category	Quality Attribute	Description
Design Qualities	Conceptual Integrity	Defines the consistency and coherence of the overall design. This includes the way components or modules are designed
	Maintainability	Ability of the system to undergo changes with a degree of ease.
	Reusability	Defines the capability for components and subsystems to be suitable for use in other applications
	Interoperability	Ability of a system or different systems to operate successfully by communicating and exchanging

Run-time Qualities		information with other external systems written and run by external parties.
	Manageability	Defines how easy it is for system administrators to manage the application
	Reliability	Ability of a system to remain operational over time
	Scalability	Ability of a system to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged.
	Security	Capability of a system to prevent malicious or accidental actions outside of the designed usages
	Performance	Indication of the responsiveness of a system to execute any action within a given time interval.
	Availability	Defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period.
System Qualities	Supportability	Ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly.
	Testability	Measure of how easy it is to create test criteria for the system and its components
User Qualities	Usability	Defines how well the application meets the requirements of the user and consumer by being intuitive
Architecture Quality	Correctness	Accountability for satisfying all the requirements of the system.
Non-runtime Quality	Portability	Ability of the system to run under different computing environment.
	Integrality	Ability to make separately developed components of the system work correctly together.
	Modifiability	Ease with which each software system can accommodate changes to its software.
Business quality attributes	Cost and schedule	Cost of the system with respect to time to market, expected project lifetime & utilization of legacy
	Marketability	Use of system with respect to market competition.

2. Software Architecture and Design – Key Principles

Software architecture is described as the organization of a system, where the system represents a collection of components that accomplish a specific set of functions.

Architectural Style

The **architectural style**, also called as **architectural pattern,** is a set of principles which shapes an application. It defines an abstract framework for a family of system in terms of the pattern of structural organization. The architectural style

- Provides a lexicon of components and connectors with rules on how they can be combined.
- Improves partitioning and allows the reuse of design by giving solutions to frequently occurring problems.
- Describes a particular way to configure a collection of components (a module with well-defined interfaces, reusable, replaceable) and connectors (communication link between modules).

The software that is built for computer-based systems exhibit one of many architectural styles. Each style describes a system category that encompasses:

- A set of component types which perform a required function by the system
- A set of connectors (subroutine call, remote procedure call, data stream, socket) that enable communication, coordination, and cooperation among components
- Semantic constraints which define how components can be integrated to form the system
- A topological layout of the components indicating their runtime interrelationships

Common Architectural Design

The following table lists architectural styles that can be organized by their key focus area.

Category	Architectural Design	Description
	Message bus	Prescribes use of a software system that can receive and send messages using one or more communication channels

Communication	Service–Oriented Architecture (SOA)	Defines the applications that expose and consume functionality as a service using contracts and messages.
Deployment	Client/server	Separate the system into two applications, where the client makes requests to the server
	3-tier or N-tier	Separates the functionality into separate segments with each segment being a tier located on a physically separate computer.
Domain	Domain Driven Design	Focused on modeling a business domain and defining business objects based on entities within the business domain.
Structure	Component Based	Breakdown the application design into reusable functional or logical components that expose well-defined communication interfaces.
	Layered	Divide the concerns of the application into stacked groups (layers).
	Object oriented	Based on the division of responsibilities of an application or system into objects, each containing the data and the behavior relevant to the object.

Types of Architecture

There are four types of architecture from the viewpoint of an enterprise and collectively, these architectures are referred to as **enterprise architecture**.

- **Business architecture**: Defines the strategy of business, governance, organization, and key business processes within an enterprise and focuses on the analysis and design of business processes.

- **Application (software) architecture**: Serves as the blueprint for individual application systems, their interactions, and their relationships to the business processes of the organization

- **Information architecture**: Defines the logical and physical data assets and data management resources.

- **Information technology (IT) architecture**: Defines the hardware and software building blocks that make up the overall information system of the organization.

Architecture Design Process

The architecture design process focuses on the decomposition of a system into components and their interactions to satisfy functional and nonfunctional requirements. The key inputs to software architecture design are:

- The requirements produced by the analysis tasks
- The hardware architecture (the software architect in turn provides requirements to the system architect, who configures the hardware architecture)

The result or output of the architecture design process is an **architectural description**. The basic architecture design process is composed of the following steps:

Understand the Problem

- This is the most crucial step because it affects the quality of the design that follows.
- Without a clear understanding of the problem, it is not possible to create an effective solution.
- Many software projects and products are considered failures because they did not actually solve a valid business problem or have a recognizable return on investment (ROI).

Identify Design Elements and their Relationships

- In this phase, build a baseline for defining the boundaries and context of the system.
- Decomposition of the system into its main components based on functional requirements. The decomposition can be modeled using a design structure matrix (DSM), which shows the dependencies between design elements without specifying the granularity of the elements.
- In this step, the first validation of the architecture is done by describing a number of system instances and this step is referred as functionality based architectural design.

Evaluate the Architecture Design

- Each quality attribute is given an estimate so in order to gather qualitative measures or quantitative data, the design is evaluated.
- It involves evaluating the architecture for conformance to architectural quality attributes requirements.
- If all estimated quality attributes are as per the required standard, the architectural design process is finished.
- If not, the third phase of software architecture design is entered: architecture transformation. If the observed quality attribute does not meet its requirements, then a new design must be created.