How Comprehensive Should Software Architecture Be?

A Perspective from ISO/IEC/IEEE 42010

Soo Dong Kim | September 2026 | Version 1.0

Software architecture is widely recognized as a fundamental design artifact for complex software systems. However, its scope is interpreted quite differently across the software engineering community. In some contexts, architecture is viewed primarily as a high-level representation of major components and their relationships. In others, it encompasses multiple architectural perspectives, quality-related design decisions, design rationale, and sufficient technical detail to guide implementation and evolution.

This variation raises a fundamental question: What should software architecture actually encompass?

This article examines this question from the perspective of ISO/IEC/IEEE 42010 and argues that, particularly for large-scale and AI-based systems, software architecture should be regarded as a comprehensive architectural design rather than merely a high-level structural representation.

1. Why the Scope of Software Architecture Is Ambiguous

There is no single, universally adopted interpretation of the scope and level of detail of software architecture. Definitions in the literature emphasize different aspects. Some characterize architecture primarily in terms of a system's fundamental structure and the relationships among its components. Others emphasize architecture styles, significant design decisions, stakeholder concerns, architecture views, quality attributes, or principles governing system evolution.

Another source of ambiguity is the common characterization of software architecture as high-level design. Architecture certainly operates at a higher level of abstraction than source code or detailed algorithm design. However, the term high-level can inadvertently imply that architecture should contain only coarse-grained structural information.

Such an interpretation may be adequate for a relatively simple system. For a large and complex system, however, identifying only major components and their relationships is unlikely to provide sufficient architectural guidance. Architects may also need to specify the allocation of functionality, organization and persistence of information, runtime interactions, deployment configurations, interfaces, quality-related design strategies, and other architecturally significant decisions.

The issue, therefore, is not simply whether software architecture should be high-level or detailed. A more useful criterion is whether the architecture provides sufficient coverage of architecturally significant concerns and sufficient technical detail for its intended purposes.

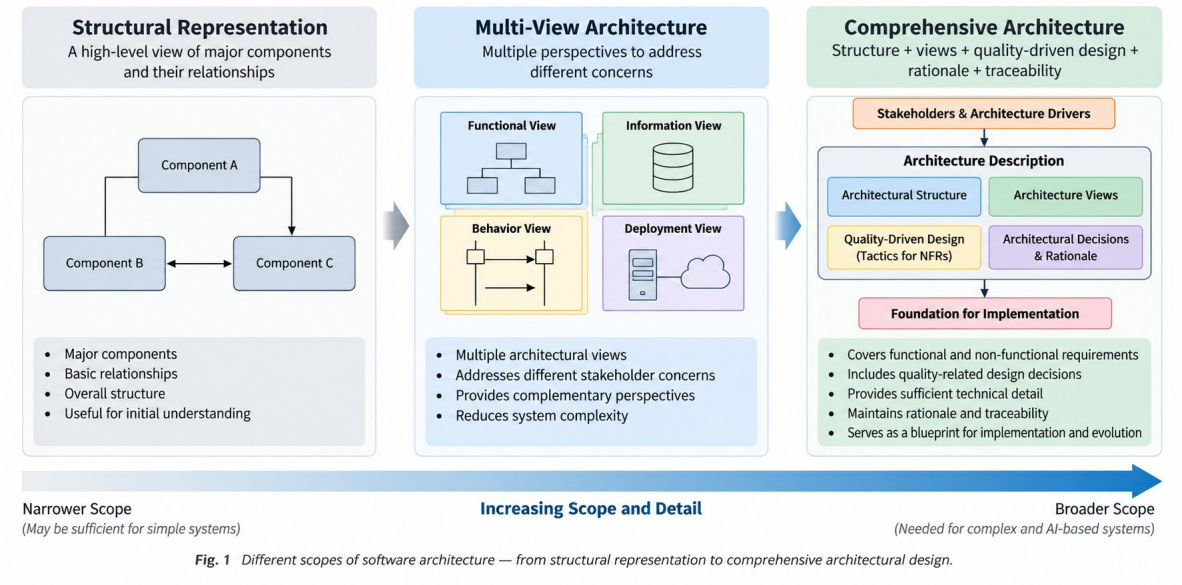

Figure 1. Different scopes of software architecture - from structural representation to comprehensive architectural design.

2. Architecture in ISO/IEC/IEEE 42010

ISO/IEC/IEEE 42010 provides an important conceptual foundation for examining the scope of architecture. ISO/IEC/IEEE 42010:2022 specifies requirements for the structure and expression of an Architecture Description (AD) and distinguishes the architecture of an entity from an Architecture Description that expresses that architecture [1]. The standard does not prescribe a specific architecting method, modeling notation, process, or tool.

Of particular relevance is the standard's concern-oriented perspective. Architecture descriptions involve stakeholders and their concerns, together with architectural concepts used to address and express those concerns. Architecture viewpoints establish conventions for constructing and interpreting architecture views, while architecture views provide representations that address particular concerns.

Some concerns exert substantial influence on the architecture and therefore become important architecture drivers. Such drivers may originate from functional requirements, quality requirements, business objectives, technological constraints, operational conditions, organizational policies, or other factors that materially influence architectural decisions.

This perspective has an important implication: an Architecture Description cannot necessarily be reduced to a single structural diagram. Different architectural concerns may require different representations and different architectural decisions. Functionality, information, runtime behavior, deployment, security, reliability, performance, and operation, for example, cannot ordinarily be expressed adequately through one structural model.

ISO/IEC/IEEE 42010 therefore provides a conceptual basis for regarding an Architecture Description as a multi-faceted expression of architecture, with its scope determined by the architectural concerns that need to be addressed.

FIGURE 2 PLACEHOLDER

Candidate book figure: Fig. 1.10 (Architecture Views in Unified Architecture Process), or an adapted ISO-oriented conceptual diagram.

Figure 2. Stakeholders, concerns, architecture drivers, viewpoints, views, and Architecture Description.

3. A Practical Definition of Software Architecture

Considering representative definitions of software architecture and the practical demands of industrial software systems, Hands-on Software Architecture: Unified Architecture Process [2] presents the following definition:

> *Software architecture is the design of a system's structure, architectural views, and architectural tactics aimed at fulfilling both functional and non-functional requirements. It provides a comprehensive and detailed design blueprint that serves as the foundation for system implementation. [2]*

This definition identifies three essential dimensions of software architecture.

Structural design establishes the fundamental organization of the system. It defines major architectural elements - such as tiers, layers, partitions, components, and connectors - and the relationships among them. The resulting structure provides a stable framework for subsequent architectural decisions.

Design for architecture views represents the system from multiple perspectives. Different views enable architects to focus on distinct architectural concerns and reduce the complexity of reasoning about the entire system at once. Functional, information, behavior, and deployment views are representative examples.

Design for quality requirements addresses architecturally significant non-functional requirements through appropriate architectural decisions and tactics. Requirements concerning performance, scalability, reliability, availability, security, maintainability, and similar quality attributes often have substantial architectural consequences.

These three dimensions are closely related. Quality-related architectural decisions may alter the structural organization of the system, change component interactions, modify runtime behavior, or require different deployment configurations. Consequently, quality design should not be treated as an isolated supplement to an otherwise completed architecture.

The second sentence of the definition is equally important. It characterizes architecture as a comprehensive and sufficiently detailed design blueprint for implementation. Comprehensiveness, however, does not imply that architecture should descend to source-code or algorithmic detail.

Rather, it means that architecturally significant design information should be sufficiently specified to support implementation, analysis, evaluation, and evolution.

FIGURE 3 PLACEHOLDER

Candidate book figure: Fig. 1.1 (Key Components of Software Architecture Definition).

Figure 3. Key components of the practical definition of software architecture.

4. Is This Definition Appropriate for Large-Scale and AI-Based Systems?

The need for a comprehensive conception of architecture becomes more apparent as the scale, heterogeneity, and quality requirements of software systems increase.

A large-scale software system may comprise numerous components, distributed computing resources, external systems, heterogeneous technologies, persistent data stores, communication networks, and multiple stakeholder groups. A high-level structural representation is useful for understanding the overall organization, but by itself it provides limited guidance for many critical design questions.

Architects must also determine how functionality is allocated among components, how information is organized and persisted, how components interact at runtime, where software elements are deployed, how external systems are integrated, and how the architecture satisfies significant quality requirements.

Quality requirements are particularly influential in large-scale systems. Scalability may require partitioning, replication, load balancing, or asynchronous processing. Availability may require redundancy, failure detection, failover, and recovery mechanisms. Security can influence trust boundaries, interfaces, communication paths, data management, and deployment. Such requirements are not peripheral to architecture; they can fundamentally reshape it.

The same reasoning applies - and often more strongly - to AI-based systems. These systems combine conventional software elements with datasets, data-processing pipelines, machine-learning models, inference components, external AI services, and specialized computing infrastructure. Generative AI and LLM-based systems can introduce additional concerns involving contextual information, model interaction, inference services, runtime orchestration, and continuous monitoring.

AI-based systems must also satisfy system-level quality requirements. A highly accurate model alone does not guarantee an acceptable software system. The overall system may additionally need to meet requirements for latency, throughput, scalability, availability, reliability, security, privacy, explainability, maintainability, and operational resilience.

Thus, as a system becomes more complex, distributed, quality-critical, or intelligent, a purely structural interpretation of software architecture becomes increasingly insufficient. The architect must reason across structure, multiple architectural perspectives, and quality-driven design decisions.

FIGURE 4 PLACEHOLDER

Candidate book figure: Fig. 5.16 (STT and Sentiment Analysis Microservices for Audio Diary App), or another AI-system architecture figure.

Figure 4. Example architecture integrating AI components with conventional software components.

5. What Should a Comprehensive Architecture Description Contain?

If software architecture is intended to provide a design blueprint for implementation, what should a sufficiently comprehensive Architecture Description contain? From the preceding discussion and the treatment presented in [2], five categories of architectural information are particularly important.

System context and architecture drivers establish the circumstances under which the architecture is designed and explain why particular architectural decisions are necessary. They include significant stakeholder concerns, requirements, constraints, technological conditions, operational factors, and other influences on the architecture.

Architectural structure defines the fundamental organization of the system, including major architectural elements, boundaries, tiers, layers, components, connectors, and their relationships.

Architecture views provide complementary representations of architecturally significant aspects of the system. Depending on the target system and its concerns, these may include functional, information, behavior, deployment, development, operation, and other relevant views.

Quality-driven architectural design describes how significant quality requirements or NFRs are addressed through appropriate architectural strategies, tactics, and design decisions. These decisions should ultimately be incorporated into the corresponding structures and architecture views. In [2], architectural tactics are treated as design strategies for satisfying NFRs, and their effects are integrated into the architecture.

Architectural decisions, rationale, and traceability explain why significant decisions have been made and how they relate to the architecture drivers that motivate them. This information is particularly valuable when architects must evaluate alternatives or balance competing quality requirements.

Comprehensive Architecture Description = Context and Architecture Drivers + Architectural Structure + Architecture Views + Quality-Driven Design + Architectural Decisions, Rationale, and Traceability

Comprehensiveness should not, however, be confused with implementation-level completeness. Not every class, algorithm, variable, or source-code construct belongs in an Architecture Description. The appropriate boundary is architectural significance: information should be included when it materially affects the system's architecture, significant requirements, major implementation decisions, evaluation, or evolution.

Accordingly, software architecture lies between two undesirable extremes. At one extreme is an overly abstract description that provides little more than a top-level component diagram. At the other is an exhaustive specification that duplicates detailed design and implementation information.

A well-engineered Architecture Description occupies the appropriate middle ground: it is a comprehensive and sufficiently detailed architectural design that transforms significant architecture drivers into coherent and implementable architectural decisions. This view is consistent with [2], which treats schematic architecture, architecture views, design for non-functional requirements, and architectural rationale as central parts of an architecture description.

FIGURE 5 PLACEHOLDER

Recommended: create a new synthesis figure for this article.

Figure 5. Scope and contents of a comprehensive Architecture Description.

References

[1] ISO/IEC/IEEE, ISO/IEC/IEEE 42010:2022 - Software, systems and enterprise - Architecture description, 2nd ed., International Organization for Standardization, 2022.

[2] S. D. Kim and M. Kim, Hands-on Software Architecture: Unified Architecture Process. Cham, Switzerland: Springer Nature Switzerland AG, 2025. doi: 10.1007/978-3-032-01184-8.

Article ID: ART-SA-01 | Published: September 2026 | Version: 1.0 | handsonse.org