

Why Software Architecture Design Needs a Methodology

From Ad Hoc Design to a Systematic Architecture Process

Soo Dong Kim • September 2026 • Version 1.0

Software architecture is one of the most consequential design activities in software engineering. Architectural decisions determine the fundamental organization of a system and strongly influence its functionality, quality attributes, implementation, operation, and evolution. For complex systems, architects must reason across requirements, system structure, multiple architectural views, runtime behavior, deployment, technology choices, and non-functional requirements. Despite this importance, architecture design is often conducted through **ad hoc or organization-specific practices**, with substantial reliance on the experience and judgment of individual architects [1, 2]. When the design process remains largely implicit, important concerns can be overlooked, decisions can be difficult to justify or reproduce, and architecture quality can vary substantially across teams and projects.

A methodological approach addresses this problem by making architecture design more explicit, systematic, and repeatable. It provides a structured basis for organizing architectural work, guiding design decisions, and improving consistency across projects. This article examines why software architecture design benefits from such an approach, identifies the core elements of a software architecture methodology, and illustrates these concepts through the *Unified Architecture Process (UAP)* [3].

1. Why Software Architecture Design Is Challenging

Architecture design is frequently characterized as the identification of a system's principal components and their relationships. Such a characterization understates the undertaking. The architect must reconcile concerns that interact and frequently conflict, and commit to decisions under information that is both incomplete at the time of decision and subject to revision thereafter. The resulting architecture must be concrete enough to guide implementation, while satisfying the functional requirements of the system together with the non-functional requirements that constrain how those functions are realized. These demands intensify in large-scale, distributed, high-assurance, and AI-based systems.

Seven factors account for the greater part of this challenge.

Comprehending Requirements and Stakeholder Concerns. Architecture design proceeds from requirements that are rarely complete, precise, or stable at the time the design must be established. Much stakeholder knowledge remains tacit and surfaces only when a candidate design is presented for review. Stakeholders also hold divergent priorities — a sponsor may favor time-to-market, an operations group availability, a security authority confidentiality — and these concerns are individually legitimate yet cannot be simultaneously satisfied. The architect must elicit what is unstated, consolidate what is stated inconsistently, and adjudicate among concerns that genuinely conflict.

Deciding Early under Maximum Uncertainty. Architectural decisions are made at the earliest stage of development, when least is known about the system and its eventual behavior, yet they are the most consequential and least reversible decisions of the design lifecycle. A choice of decomposition strategy,

distribution model, or persistence paradigm conditions all subsequent work, and revising it after implementation has begun costs far more than the change itself. This asymmetry between what is known at the moment of decision and what follows from it distinguishes architectural design from detailed design, and it cannot be deferred away without halting development.

Handling Structural and Scale Complexity. Modern systems comprise numerous modules distributed across tiers and layers that must interoperate seamlessly, and integration with external systems, third-party services, and legacy subsystems compounds the structure further. As elements and interdependencies accumulate, the interactions an architect must reason about grow disproportionately, and the design ceases to be comprehensible by inspection. Decomposing such a system into elements whose responsibilities are cohesive and whose couplings remain deliberately constrained admits no mechanical solution.

Satisfying Non-Functional Requirements and Resolving Quality Trade-offs. Non-functional requirements are the principal determinant of architectural structure, and they are frequently stated in terms that are imprecise or unmeasurable. More fundamentally, quality attributes interfere with one another: replication improves availability at the expense of consistency, encryption strengthens confidentiality at the expense of throughput, and abstraction introduced for modifiability imposes a performance penalty. No design optimizes all attributes at once, so the architect must identify these interactions, evaluate the trade-offs available, and justify the balance selected rather than assert it.

Integrating Multi-Disciplinary Technologies. Architecture design draws on technologies from three distinct groups, as shown in Figure 1.

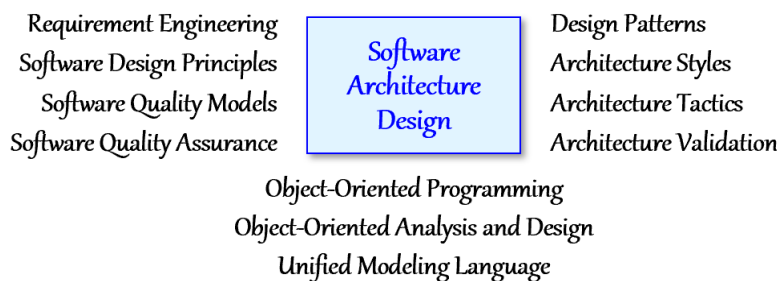


Figure 1. Key Technologies of Software Architecture Design

The figure arranges these along three axes: foundational software engineering disciplines, technologies intrinsic to architecture design, and object-oriented technologies. Each group carries its own design principles, assumptions, and operational constraints, and they rarely compose without friction — what one treats as settled may be precisely what another leaves open, and the conflict surfaces only when both are placed in the same design. The body of knowledge is also expansive and continually displaced by new platforms, frameworks, and computing paradigms. Reconciling these technologies is design work in its own right, and does not follow from proficiency in the constituent technologies.

Documenting the Architecture Description. Architecture must ultimately be communicated through models, views, decisions, rationale, and supporting documentation, and it must serve audiences — sponsors, project managers, developers, testers, operations staff — who require different levels of abstraction and different selections of content. Producing an Architecture Description that is sufficiently complete, precise, consistent across views, and understandable to all of its readers is itself a demanding engineering task. It is also routinely deferred, at which point the description records what was built rather than guiding what is to be built.

Validating the Architecture Before Implementation. The architecture must be assessed before an executable system exists to be measured. Conformance to requirements, satisfaction of quality attributes, feasibility within schedule and budget, and consistency among design elements must all be established through models, scenarios, and analytical reasoning rather than observation of a running system. Establishing confidence by such indirect means, with enough rigor to justify committing an organization to the design, requires systematic evaluation methods that are not widely practiced. In their absence, architectural defects propagate into implementation, where correction is substantially more expensive.

These factors are compounded by the state of practice surrounding them: systematic end-to-end design procedures are scarce, and what best practice exists remains embedded in the experience of senior practitioners and individual organizations, difficult to transfer and unevenly documented. Architecture design consequently becomes ad hoc, varying in rigor and outcome from one architect and project to the next. A methodology is intended to reduce this variability without eliminating the engineering judgment and creativity that architecture requires.

2. What Is a Methodology?

In general, a methodology is a structured framework for performing a class of work or solving a defined class of problems. It organizes relevant knowledge into a systematic process and provides guidance for executing that process. Methodologies are used across research, engineering, education, business, sports, and many other domains because complex work becomes more reliable when its objectives, procedures, and supporting knowledge are made explicit [3].

A methodology is a coherent framework that combines a structured process with explicit instructions, grounded in an established knowledge base, for performing a class of work systematically and repeatably.

This definition emphasizes three complementary elements.

Structured process. A structured process specifies the major units of work, their objectives, and their logical sequence, thereby organizing a complex undertaking into manageable stages.

Explicit instructions. Explicit instructions explain how each unit of work should be performed, supplying methods, procedures, techniques, rules, and practical guidance.

Established knowledge base. An established knowledge base supplies the principles, concepts, technologies, and best practices on which the process and instructions are grounded.

When these elements are integrated, the methodology provides a disciplined basis for performing work systematically, consistently, and repeatably rather than relying primarily on individual experience or ad hoc practices.

Therefore, a mature methodology should be distinguished from several common misconceptions:

A design process alone. A process specifies what activities should be performed and in what sequence, but without explicit instructions, methods, and supporting knowledge, it provides limited guidance on how the work should actually be carried out.

A collection of guidelines without a well-defined process. Guidelines may provide useful recommendations and best practices, but when they are not organized within a coherent sequence of activities, their application can become inconsistent and difficult to manage.

A checklist of recommended practices. Checklists are valuable for verification and quality assurance, but they do not by themselves define how design work should proceed or how decisions should be made.

A set of tools or techniques. Individual techniques, modeling notations, templates, or tools can support design activities, but they constitute only parts of a methodology unless they are integrated into a structured process with explicit guidance.

A rigid procedure that eliminates professional judgment. A methodology should provide discipline and consistency while still allowing architects to exercise engineering judgment, adapt the process to project context, and make trade-offs among alternative design options.

These characteristics bear directly on the factors identified in Section 1. Architecture design requires that work be ordered explicitly, that decisions be recorded with their rationale, that the resulting design be documented in a consistent structure, and that it be evaluated before implementation — none of which follows from technical knowledge alone.

3. Methodological Approach to Architecture Design

Software architecture does not currently have a single, generally accepted standard methodology for architecture design, nor has any one published methodology become dominant across the field. ISO/IEC/IEEE 42010:2022 establishes concepts and requirements for Architecture Descriptions, but it explicitly does not prescribe a particular architecting method, modeling notation, process, or tool [4]. Consequently, organizations and architects frequently assemble their own approaches from selected methods, frameworks, internal practices, and accumulated experience.

Architecture design has not lacked for methods. Design-oriented approaches such as Attribute-Driven Design, evaluation methods such as ATAM, documentation frameworks such as Views and Beyond, enterprise frameworks such as TOGAF, and general development processes such as RUP each contribute something to architectural work. What none provides on its own is an end-to-end account of how an architecture is designed: some prescribe documentation without a design procedure, others evaluation without a design procedure, others a lifecycle in which architecture is one phase among many. A comparative study of five industrial approaches nevertheless identified a common underlying structure across them [1], which indicates that architecture design is amenable to systematic treatment even where no single published method has become standard.

The freedom to compose one's own approach can be useful, but it also leaves the quality of an architecture dependent on which practices a given organization happens to have assembled [2]. A methodological approach for architecture design must therefore do more than order the work: it must ensure that each of the factors identified in Section 1 is addressed deliberately rather than left to the discretion of the individual architect.

A methodological approach provides several important benefits:

Systematic Progression. Architecture design can be decomposed into manageable activities and performed in a logical sequence, enabling architects to address complex concerns in a disciplined and controlled manner.

Improved Completeness. Explicit activities, steps, and design procedures reduce the likelihood that architecturally significant concerns, requirements, or design decisions are inadvertently omitted.

Consistency. Common terminology, procedures, modeling conventions, and architectural artifacts promote consistent design practices across architects, teams, and projects.

Traceability. Architectural decisions can be explicitly related to requirements, constraints, design inputs, and resulting artifacts, and recorded together with the rationale and the alternatives considered at the time they were taken. This makes the architecture easier to justify, review, maintain, and evolve, and allows decisions made under incomplete information to be revisited deliberately when that information improves.

Quality Assurance. Defined design guidelines, evaluation procedures, and checklists provide systematic means for detecting deficiencies and validating architectural decisions before implementation.

Efficiency. Reusable processes, methods, templates, and best practices reduce trial-and-error and unnecessary rework, allowing architects to focus effort on the most architecturally significant concerns.

A methodological approach is sometimes taken to imply extensive up-front design, and thus to sit uneasily with iterative and incremental development. The opposition is more apparent than real. What a methodology fixes is the order and rigor of architectural reasoning, not the quantity of design performed before implementation begins. Architecturally significant decisions must be taken deliberately whenever they arise — in the first iteration or the fifteenth — and the same activities, steps, and evaluation criteria apply whether the architecture is established in a single pass or elaborated across several increments.

Architectural effort should be proportioned to the system. A practical methodology is therefore tailorable, preserving systematic reasoning for architecturally significant concerns while permitting activities to be reduced, adapted, or omitted where the project context warrants.

4. Elements of a Software Architecture Methodology

A methodology comprises a structured process together with the instructions for executing it, both resting on the knowledge base described in Section 2 rather than standing alongside it. Software architecture methodology follows this general form. A process model specifies what architectural work is to be performed and in what sequence; design instructions specify how each unit of that work is to be carried out. These two elements constitute the methodology proper. Applying them to an actual system, however, depends on two further resources: a standardized Architecture Description template, which governs how the resulting artifacts are documented, and a body of case studies, which demonstrates how the methodology performs when exercised on real systems.

Core Elements of Architecture Methodology

Process Model. The process model defines what architectural work is to be performed and in what sequence. It decomposes architecture design into identifiable units of work, each associated with a specific objective and expected outcomes, so that significant concerns are addressed systematically rather than opportunistically. The common structure of such a model is shown in Figure 2: a requirements specification enters the process, is elaborated through a sequence of activities that decompose in turn into steps and tasks, and yields an Architecture Description as its result.

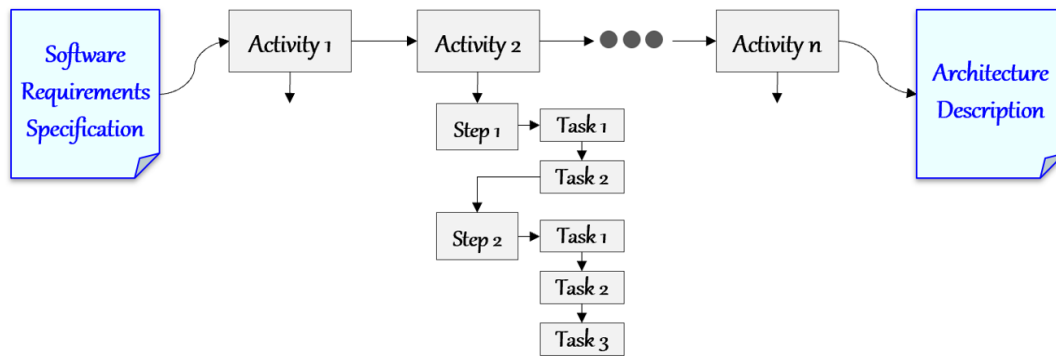


Figure 2. Common Structure of Process Model

A process model must further identify the relationships and dependencies among these units, since architectural work is rarely a collection of independent tasks and the output of one unit commonly serves as input to those that follow. Making such dependencies explicit improves the coherence and traceability of the process as a whole. The Unified Architecture Process realizes this structure through three hierarchical work units — Activity, Step, and Task.

Design Instructions. A process model identifies what is to be done but does not by itself explain how the work is to be performed. Design instructions supply this operational guidance through detailed procedures, methods, techniques, decision criteria, examples, diagrams, and checklists. Instructions of this kind render explicit, transferable, and reusable the design knowledge that would otherwise remain implicit in the experience of individual architects. Three criteria govern their adequacy.

Logical sequencing. Each step is decomposed into logically ordered, fine-grained tasks, so that the output of one task furnishes a sound basis for those that follow.

Specificity. Instructions supply explicit procedures and decision guidance, accompanied by figures, diagrams, tables, and examples, so that the work may be interpreted and executed with minimal ambiguity.

Consistency. Terminology, notation, and methodological convention are maintained uniformly across all activities, steps, and tasks, which aids comprehension, reduces the incidence of error, and supports collaborative design.

Supporting Resources for the Methodology

The process model and design instructions together define the methodology. Applying it to an actual system, however, raises two questions that neither answers: where the resulting design is to be recorded, and how the guidance appears when exercised on a real system.

Architecture Description Template. An Architecture Description template provides a standardized structure for organizing and documenting the architectural artifacts, decisions, rationale, and evaluation results produced during design. When the template is aligned with the process model, every design activity has a corresponding location in the Architecture Description. Such alignment improves documentation consistency, reduces the likelihood that significant information is omitted, and strengthens traceability between the design process and its resulting artifacts. The Unified Architecture Process provides a template organized according to its activities, steps, and tasks.

Case Studies. Case studies supply evidence of how a methodology performs in application. A process and its instructions define the intended method of work; realistic examples show how architects interpret that guidance, reach design decisions, resolve trade-offs, and adapt the methodology to differing system contexts. They are particularly valuable for conveying architectural knowledge that resists expression as formal procedure, illustrating recurring problems, effective design strategies, typical artifacts, and lessons drawn from actual applications. Consolidated over time into reusable best practices, such experience strengthens the methodology and reduces excessive dependence on individual expertise. Case studies accompanying the Unified Architecture Process span enterprise systems, mobile applications, embedded systems, cloud platforms, autonomous systems, and machine-learning systems.

Together, these elements provide a foundation for systematic and repeatable architecture design: the methodology establishes how the work is conducted, and the supporting resources make it applicable and demonstrably effective in practice.

5. An Illustration: the Unified Architecture Process

The Unified Architecture Process (UAP) is a methodology developed specifically for software architecture design. It pairs a structured process model with detailed instructions and aligned supporting artifacts, so that architects may proceed systematically from requirements and system context through architectural design to evaluation [3].

Process Model

The Unified Architecture Process organizes architectural work into three hierarchical units. An Activity constitutes a major phase of the design process; a Step is a structured procedure within an activity; and a Task is the most granular unit, denoting a concrete action required to complete a step. The hierarchy applies the divide-and-conquer principle to architectural work, so that complex design objectives are addressed through smaller units of manageable scope.

At the highest level, the process comprises six major activities, shown in Figure 3: requirements refinement, system context analysis, schematic architecture design, design for architecture views, design for non-functional requirements, and architecture evaluation. Together they constitute an end-to-end design workflow that carries a specification from stakeholder requirements to an evaluated architecture.

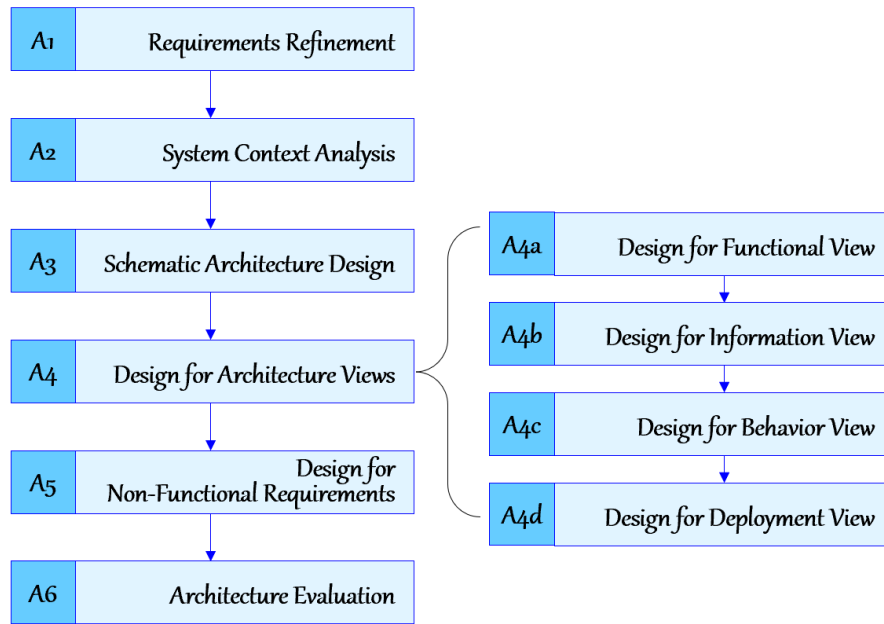


Figure 3. Process Model of Unified Architecture Process

Design Instructions

A process model alone does not suffice for practical application. Each step of the Unified Architecture Process is therefore accompanied by an instruction supplying the methods, procedures, best practices, and illustrative examples required to perform it correctly and consistently, developed according to the three criteria set out in Section 4 [3].

Supporting Artifacts and Tailoring

The Unified Architecture Process aligns its process model with an Architecture Description template in which each chapter corresponds to an activity, so that design decisions, rationale, models, and outcomes are documented in a structure mirroring the work that produced them. Activity-level checklists provide a further means of verifying critical design aspects and detecting omissions. Together these mechanisms strengthen completeness, traceability, communication, and review.

The process is tailored to the target system rather than applied uniformly. Individual activities may be omitted where conditions warrant: requirements refinement, where requirements are well defined and have been validated in prior projects; context analysis, where the system context closely mirrors that of previous work [3].

The Unified Architecture Process thus illustrates what a methodology contributes to architecture design: it organizes architectural knowledge into a defined process, supplies actionable guidance for performing the work, and supports consistent documentation and evaluation. Such a framework does not supplant the architect. It establishes the conditions under which architectural expertise can be exercised systematically, efficiently, and transparently.

A complete specification of the Unified Architecture Process — its process model, activities, steps, tasks, design instructions, examples, templates, checklists, and tailoring guidelines — is given in *Hands-on Software Architecture: Unified Architecture Process*, by Soo Dong Kim and Mira Kim (Springer, 2025).

References

- [1] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, “A general model of software architecture design derived from five industrial approaches,” *Journal of Systems and Software*, vol. 80, no. 1, pp. 106–126, 2007.
- [2] P. Y. Reyes-Delgado et al., “The strengths and weaknesses of software architecture design in the RUP, MSF, MBASE and RUP-SOA methodologies: A conceptual review,” *Computer Standards & Interfaces*, vol. 47, pp. 24–41, 2016.
- [3] S. D. Kim and M. Kim, *Hands-on Software Architecture: Unified Architecture Process*. Cham, Switzerland: Springer, 2025. ISBN 978-3-032-01183-1.
- [4] ISO/IEC/IEEE, *ISO/IEC/IEEE 42010:2022 - Software, systems and enterprise - Architecture Description*, 2nd ed., International Organization for Standardization, 2022.

Article ID: ART-SA-02 • Published: September 2026 • Version: 1.0 • handsonse.org