

From SRS Requirements to Architectural Drivers

What Truly Shapes the Design of a Software Architecture

Mira Kim, Soo Dong Kim • September 2026 • Version 1.0

A recurring pattern in industrial practice is that an architect receives a Software Requirements Specification (SRS), studies it, and proceeds directly to architecture design, treating the document as the principal input and as the standard against which the resulting architecture is judged. The practice is understandable, since an SRS does supply substantial design input: architecturally significant functional requirements guide the allocation of functionality to architectural elements, while non-functional requirements constrain the choice of architectural styles, architectural tactics, and other design decisions.

However, an SRS and a set of architectural drivers are not equivalent. Architecturally significant information may lie outside the specification, requirements stated within it may need refinement before they can reliably guide design, and some documented requirements may carry little architectural consequence. Figure 1 shows the resulting relationship between the two sets. This article examines these three gaps and clarifies what qualifies information as an architectural driver.

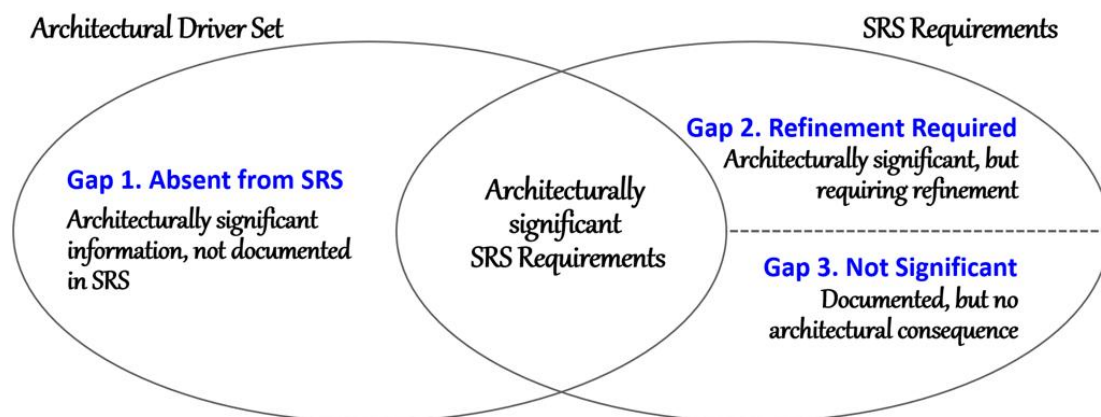


Figure 1. Gaps Between SRS and Architectural Driver Set

The difference between the two sets can be understood as three distinct gaps.

Gap 1 — Absent from SRS. Architecturally significant information that the specification never records, such as stakeholder concerns, organizational policies, and constraints inferred from required functionality and behavior.

Gap 2 — Refinement Required. Documented requirements that are architecturally significant but too ambiguous, incomplete, or inconsistent to guide design as stated.

Gap 3 — Not Significant. Documented requirements that can be implemented locally without materially affecting architectural decisions.

1. What Is an Architectural Driver?

An architectural driver is a factor that materially influences one or more architectural decisions. The term is commonly used for requirements, quality attributes, constraints, and other factors that shape consequential architectural choices [1]. The defining criterion is not where the information is documented, but whether it has sufficient architectural significance to affect the structure, behavior, deployment, quality-related strategies, or other consequential aspects of the architecture.

Architectural drivers may originate from sources such as the following, which are not mutually exclusive [2]:

Functional Requirements. Architecturally significant functions whose allocation to components shapes the structure of the system.

Quality Requirements. Architecturally significant NFRs that constrain the styles, tactics, and design decisions available to the architect.

Structural Constraints. Structural requirements stated in the specification or inferred from the functionality and behavior the system must support.

Stakeholder Concerns. Expectations and restrictions held by stakeholders, which are the primary origin of quality-related drivers.

Business Constraints. Organizational objectives, policies, and restrictions that limit the design alternatives available to the architect.

Operational Conditions. Characteristics of the operating environment and the technologies on which the system depends.

Facts and Policies. Contextual conditions underlying an NFR that determine how the required quality can actually be achieved.

The architectural driver set therefore extends beyond the requirements explicitly documented in an SRS.

Stakeholder Architectural Concerns

Stakeholder concerns constitute an important source of architectural drivers. Architectural concerns represent high-level requirements, expectations, or constraints held by stakeholders that may influence the architecture of the target system, including quality attributes and design restrictions. At the time of elicitation, a concern may be imprecise, technically infeasible, or not yet verifiable, and in that state it cannot yet guide architectural decisions. ISO/IEC/IEEE 42010 likewise places stakeholder concerns at the center of architecture description, but does not prescribe a process for transforming them into architectural inputs.

A concern is therefore not yet an architectural driver. It becomes one only when it carries architectural significance, whether it is formalized as a non-functional requirement or constrains design decisions directly without being documented at all. A concern that is marginal to system objectives, or that ordinary design practice already addresses, yields no driver.

Figure 2 shows the resulting relationship, which is many-to-many in both directions. A single concern may yield several drivers, one, or none. A driver, in turn, may originate from one or more concerns, or from none, since structural, business, and technological sources produce drivers that no stakeholder ever articulated.



Figure 2. Relationship Between Architectural Concerns and Architectural Drivers

Because concerns are the primary source of quality-related drivers, eliciting them from stakeholder groups the specification's authors did not consult is the most effective means of exposing drivers the SRS never recorded.

Concerns are also where the three gaps of this article originate. Tracing drivers back to the concerns behind them is how an architect discovers what the SRS is missing, what it states too loosely, and what it contains that does not matter architecturally. The sections that follow examine each gap in turn.

2. Gap 1: Absent from SRS

Some architecturally significant information is not documented in the SRS. Its absence does not necessarily indicate poor requirements engineering. An SRS primarily specifies what the system is required to do and the qualities it should possess; it does not necessarily capture every stakeholder concern, contextual condition, structural implication, or organizational restriction that materially affects architecture design. Three sources account for most of what is missing.

Stakeholder Concerns

Different stakeholder groups hold different architectural concerns. A maintenance team may be concerned with modifiability, a legal or compliance group with auditability and traceability, an operations group with availability and recovery, and users with usability and responsiveness.

Such concerns emerge through interviews, workshops, observation, document analysis, and other elicitation techniques rather than through analysis of the existing SRS alone. An architect who relies exclusively on the document therefore risks designing against an incomplete set of architectural forces.

Structural Constraints

An SRS may explicitly identify external systems, services, platforms, tiers, interfaces, deployment restrictions, or other structural requirements. Additional structural constraints, however, must be inferred during architectural analysis.

Intensive data manipulation, substantial computational demand, or extensive interaction with users and external systems may imply parallel processing, data caching, or load balancing. Behavioral characteristics introduce further constraints: data streaming, event-driven invocation, closed-loop processing, parallel execution, or timed invocation may imply asynchronous communication, event-driven components, concurrent execution, or distributed coordination mechanisms [2].

These constraints are architecturally significant even though they are not documented in the SRS. They emerge only when the architect interprets what the required functionality and behavior imply for the architecture.

Facts and Policies

A non-functional requirement may depend on facts and organizational policies that the requirement statement itself does not contain. An SRS may specify high availability for a headquarters server, for example. The requirement expresses the desired quality, but the conditions that determine how it can be achieved include expected workload, infrastructure characteristics, recovery restrictions, operational practices, cost constraints, and organizational policies concerning deployment or redundancy.

These conditions materially affect architecture design even though they are not conventional SRS requirements.

These sources, together with business restrictions and operational conditions that the specification rarely records, contribute architecturally significant information the SRS does not document. The specification provides major input to architecture design, but it does not contain the complete architectural driver set.

3. Gap 2: Refinement Required

The presence of a requirement in an SRS does not make it suitable for architectural decision-making. A requirement may be architecturally significant and still be unusable as stated, because deficiencies in its formulation propagate into subsequent architectural models unless they are identified and resolved. Six recurring types of deficiency affect specifications: incompleteness, incorrectness, inconsistency, ambiguity, redundancy, and irrelevance [2].

Incompleteness. Necessary information is missing.

Incorrectness. A requirement contains inaccurate information.

Inconsistency. Requirements conflict with one another.

Ambiguity. A requirement permits multiple reasonable interpretations.

Redundancy. The same information is stated repeatedly, increasing the possibility of inconsistency.

Irrelevance. Material is included that does not meaningfully contribute to the specification of the target system.

These deficiencies matter to architecture because architectural decisions amplify assumptions. An ambiguity left unresolved at the requirements level may become a component responsibility, interface, control flow, persistent information structure, or deployment decision that is expensive to reverse.

What Refinement Changes

Consider an AI-based *English Speaking Tutor*, a mobile application in which a learner speaks a response to a prompt and receives feedback on it. The system uses two models. A compact automatic speech recognition model runs on the device and produces both a transcription and acoustic confidence for each utterance. A large language model, accessed as an external cloud service under per-request latency and cost limits, assesses the transcription and generates corrective feedback. Learners practice in short, intermittent sessions in which network connectivity is not guaranteed.

The specification for this system states FR-7 as follows: the system shall evaluate a learner's spoken response and provide corrective feedback within three seconds. The statement names an actor, an action, and a measurable bound, and it passes review. What it leaves unspecified is architectural: which model performs which part of the evaluation, where the three seconds is measured, and how the system behaves when the cloud model is slow or unavailable.

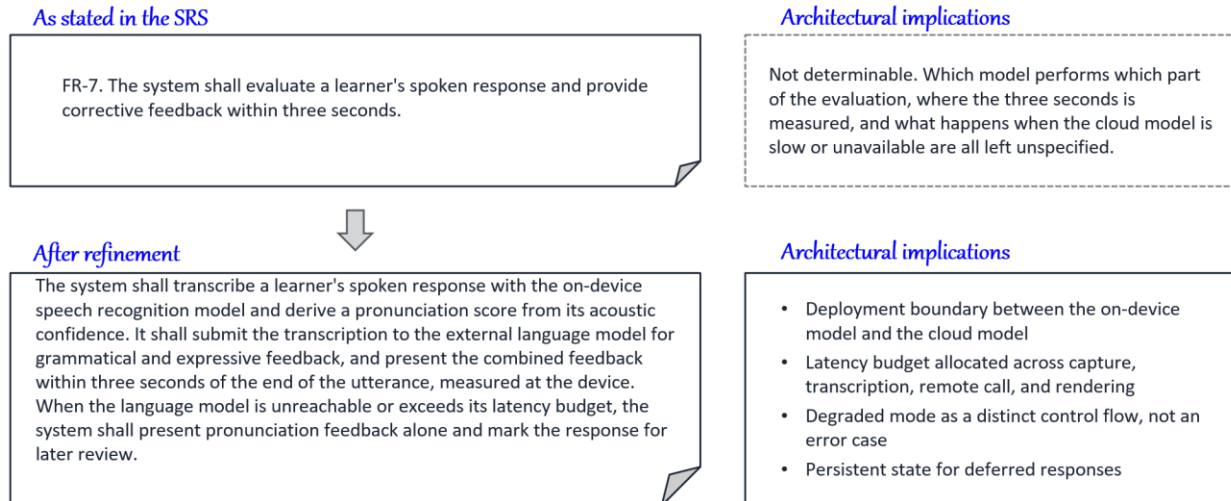


Figure 3. Architectural Implications Exposed by Refining a Requirement

As shown in Figure 3, the refined statement exposes implications that the original did not. Splitting evaluation between the on-device model and the cloud model fixes a deployment boundary and divides the functional decomposition accordingly. The three-second bound, once anchored at the device, must be allocated across capture, transcription, the remote call, and rendering, which constrains both model size and the timeout policy for the external service. The degraded mode is a distinct control flow rather than an error case, and the responses it defers require persistent state. The business intent is unchanged; only its precision is.

A requirement in this gap is therefore architecturally significant but not yet usable. Deficiencies that materially affect architectural decisions must be resolved before the requirement can serve as a design basis.

4. Gap 3: Not Significant

The third gap operates in the opposite direction: not every requirement documented in an SRS is architecturally significant.

Some requirements are essential for functional correctness, contractual compliance, business operation, or user satisfaction, yet can be implemented locally without materially affecting the architecture. Treating every documented requirement as an architectural driver enlarges the driver set unnecessarily and obscures the requirements and conditions that genuinely shape major design decisions. The criterion for inclusion is architectural significance, not documentation status.

A Practical Discriminating Test

Would a plausible alternative formulation of this requirement lead to a different decision in any of the following categories?

The categories are architectural styles, component boundaries and responsibilities, interactions among components, persistent information structures, deployment configuration, external interfaces, runtime behavior, and quality-related architectural strategies.

Consider two requirements from an AI-based tutoring system. One states that a practice streak increments after at least one session per day; the other states that speech is transcribed on the device and evaluated by

an external language model within three seconds. Raising the streak threshold from one session to three changes no decision in any of the categories above.

The second requirement behaves differently. Moving transcription to the cloud changes the deployment boundary and the allocation of responsibilities, and removes the system's ability to function without connectivity. Relaxing the latency bound changes which deployment options remain admissible.

A requirement whose plausible alternative formulations disturb none of these categories is not an architectural driver, however important it may be to the business or to users.

A requirement may acquire significance in aggregate. When many individually local rules must vary, be configured, or be audited together, their combined effect becomes an architectural concern even though no single rule is significant on its own.

Architectural significance is also contextual rather than permanent. A rule that is purely local under present conditions may acquire architectural consequences when it must later hold across devices, survive disconnected operation, or preserve a verifiable history. The same requirement becomes a driver when its context changes.

Documentation in the SRS therefore establishes that a requirement matters, not that it drives the architecture.

5. From Requirements to Drivers

The three gaps share a single implication. The architectural driver set is not obtained by extracting requirements from a specification. It is constructed, by judging which documented requirements actually shape the architecture, refining those that are not yet usable, and seeking out the concerns, constraints, and conditions the specification never recorded.

This also means that a driver is usually richer than the requirement that carries it. A quality requirement becomes usable only together with the conditions that determine how the quality can be achieved: the stakeholder expectation behind it, the operating facts it depends on, and the organizational policies it must respect. Read alone, the requirement states a goal. Read with its context, it states a design problem.

The SRS is therefore a source of architectural drivers, not the architectural driver set itself.

A complete treatment is provided in *Hands-on Software Architecture: Unified Architecture Process*, by Soo Dong Kim and Mira Kim (Springer, 2025), where Activity A1 addresses requirements refinement and the elicitation of architectural concerns, Activity A2 derives structural constraints through system context analysis, and Activity A5 identifies the facts and policies underlying non-functional requirements.

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, 2025. ISBN 978-3-032-01183-1.

Article ID: ART-SA-03 • Published: September 2026 • Version: 1.0 • handsonse.org