GENERAL SPECIFICATION USAGE GUIDE

1. Purpose of This Guide

This guide explains how to read, interpret, and use the specification documents that define a system. It applies to all architecture specifications, technical specifications, and related artefacts produced within Inquisitor Labs documentation ecosystem. Its goal is to give engineers, reviewers, developers, coders and stakeholders a consistent, predictable workflow for understanding and applying the specifications correctly.

2. What an Architecture Specification Is

The architecture specification defines the system at a conceptual and structural level. It describes the mechanisms, constraints, behaviours, and design principles that shape the system. It answers **what the system is**, **why it works the way it does**, and **which architectural decisions are fixed**.

It must **not** contain:

- Implementation detail
- Code-level decisions
- Technology choices that belong in engineering
- Changes that alter the system's conceptual model

The architecture is the stable foundation. Everything else flows from it.

3. What a Technical Specification Is

The technical specification translates the architecture into implementable engineering detail. It defines the components, interfaces, data structures, workflows, and operational rules required to build the system.

It answers **how the system is implemented**, **how components interact**, and **what engineering decisions are required**.

It must **not** contain:

- New architectural mechanisms
- Conceptual changes
- Redefinitions of system behaviour
- Decisions that contradict the architecture

The technical specification is authoritative for implementation, but subordinate to the architecture.

4. How the Documents Relate

The documents form a strict hierarchy:

Architecture → Technical Specification → Implementation

- The architecture defines the conceptual model.
- The technical specification defines the engineering model.
- The implementation realises the technical model in code.

Decisions flow **downward**. Constraints flow **upward**.

If a change is required at a lower layer that contradicts a higher layer, the higher layer must be revised first.

5. Correct Reading Order

To understand a system correctly:

1. **Read the architecture specification** Understand the conceptual model, mechanisms, and constraints.

2. **Read the technical specification** Understand how the architecture is realised in engineering terms.
3. **Begin implementation** Use the technical specification as the primary reference.
4. **Refer back to the architecture only for conceptual alignment** Do not reinterpret or extend the architecture during implementation.

This order prevents misinterpretation and ensures consistency.

6. How to Use the Documents During Development

During development:

- Use the **architecture** to maintain conceptual integrity.
- Use the **technical specification** to guide engineering decisions.
- Use the **implementation** to validate that the system behaves as specified.
- Escalate any contradictions or ambiguities immediately.

The architecture is the primary source of conceptual truth. All other documents must align with it.

7. Where Changes Should Be Made

Changes must be made at the correct layer:

- **Architecture changes** Only when the conceptual model, mechanisms, or system behaviour must change.
- **Technical specification changes** When engineering detail, interfaces, or workflows need revision.
- **Implementation changes** When code needs to be corrected or improved without altering the spec.

Never push changes downward or upward incorrectly. Never “fix” an architectural issue in the technical spec. Never “fix” a technical issue in the architecture.

8. Common Mistakes to Avoid

- Treating the architecture as a technical spec
- Adding implementation detail to the architecture
- Introducing new mechanisms in the technical spec
- Allowing implementation to drift from the technical spec
- Updating documents out of order
- Mixing conceptual and engineering concerns
- Using the architecture to justify implementation shortcuts

These mistakes break alignment and create long-term system instability.

9. How to Evaluate Compliance

Compliance should be checked at three levels:

- **Architecture compliance** Does the system behave according to the conceptual model?
- **Technical specification compliance** Does the implementation match the defined engineering detail?
- **Implementation alignment** Does the code reflect the technical specification without deviation?

Any deviation must be documented, justified, and escalated.

10. Document Set Lifecycle

Specifications evolve over time. The lifecycle is:

- **Versioning** Each document must have a clear version and change history.
- **Architecture revisions** Only when conceptual changes are required.
- **Technical specification revisions** When engineering detail changes or new components are added.
- **Long-term consistency** All documents must remain aligned across versions.

A disciplined lifecycle prevents drift and preserves system integrity.

11. How to Get Started

1. Identify the documents you need

Confirm which architecture and technical specifications apply to the system or component you are working on.

2. Read the architecture first

Understand the system's conceptual model, mechanisms, and constraints before looking at engineering detail.

3. Read the technical specification next

Use it to understand the required components, interfaces, workflows, and operational rules.

4. Clarify any gaps early

If something is unclear or appears inconsistent, escalate before implementation begins. Do not interpret or improvise.

5. Plan implementation based on the technical specification

Break down the engineering work using the technical specification as the authoritative reference.

6. Use the architecture only for conceptual alignment

Do not derive engineering detail from the architecture. Use it to ensure the implementation remains conceptually correct.

7. Review work against both documents

- Check conceptual alignment against the architecture
- Check engineering alignment against the technical specification

8. Update the correct document if changes are needed

- Conceptual change → architecture
- Engineering change → technical specification
- Code-only change → implementation

9. Keep documentation in sync

After implementation stabilises, ensure all dependent documentation reflects the current technical specification.

12. Definitions

Architecture

The conceptual structure of a system. It defines mechanisms, behaviours, constraints, and the relationships between components. It does not include implementation detail.

Technical Specification

The engineering-level description of how the architecture is realised. It defines components, interfaces, data structures, workflows, and operational rules.

Implementation

The concrete realisation of the technical specification in code, configuration, or operational systems.

Mechanism

A defined process, rule, or behaviour within the architecture that produces a specific outcome. Mechanisms are conceptual, not code.

Constraint

A rule that limits or shapes system behaviour. Constraints may be conceptual (architecture) or operational (technical spec).

Component

A discrete part of the system with a defined purpose and boundaries. Components exist at both architectural and technical levels.

Interface

A defined point of interaction between components. Interfaces specify inputs, outputs, and rules of interaction.

Behaviour

How a component or mechanism acts under specific conditions. Behaviour is defined conceptually in the architecture and operationally in the technical spec.

Invariant

A rule that must always hold true for the system to function correctly. Invariants cannot be overridden, bypassed, or reinterpreted at the technical or implementation level.

Assumption

A condition believed to be true for the system to operate correctly. Assumptions must be validated or explicitly documented.

Dependency

A requirement that one component, mechanism, or behaviour relies on another. Dependencies must be stable and explicitly defined.

Scope

The boundaries within which a component, mechanism, or document applies. Scope prevents overreach and ambiguity.

Version

A numbered state of a document or system that reflects its evolution. Versions must be tracked and referenced consistently.

-Document Ends-