ISO 26262: Road Vehicles - Functional Safety Standard

Technical Analysis of ASIL Allocation, Freedom from Interference, and Software Verification

CRITICAL GOVERNANCE PRINCIPLE (ISO 26262 (PARTS 2, 3, 4, 6 & 8)): Software controls are classified as programmable risk mitigation elements. Under international functional safety law, software cannot be certified in isolation from physical hardware interlocks. Independent verification (Level I1/I2/I3) and multidisciplinary sign-off are mandatory requirements prior to operational release.

1. Architectural Paradigms & Safety Governance

ISO 26262 adapts IEC 61508 for embedded electronic control systems interacting with dynamic physical environments, defining Automotive Safety Integrity Levels (ASIL A through ASIL D).

Part 2: Management of Functional Safety

Establishes safety culture and governance rules. Formally separates the role of the Software Developer from the Functional Safety Manager and Independent Safety Assessor.

- Conflict of Interest Mitigation: Project delivery schedules must never override safety sign-off gates.
- Confirmation Review: Formal safety reviews must be conducted by individuals with verified independence from the design team.

Part 4: System Level Development & Hardware-Software Interface (HSI)

Clause 7 defines the Technical Safety Concept (TSC) and the Hardware-Software Interface specification. Software developers must implement controls strictly mapped to documented HSI specifications.

- Fault Tolerant Time Interval (FTTI): The maximum time allowed between fault occurrence and transition to a safe state.
- Hardware Independence: Software commands must be paired with independent hardware watchdogs and monitor circuits.

Part 6: Product Development at the Software Level

Clause 7 and 8 mandate spatial and temporal freedom from interference (FFI), defensive programming, and fault injection testing.

- Freedom from Interference: A fault in non-safety software must not cause failure in safety-critical software partitions.
- Fault Handling: Systems must detect invalid states, memory corruptions, and communication drops, forcing safe default outputs.