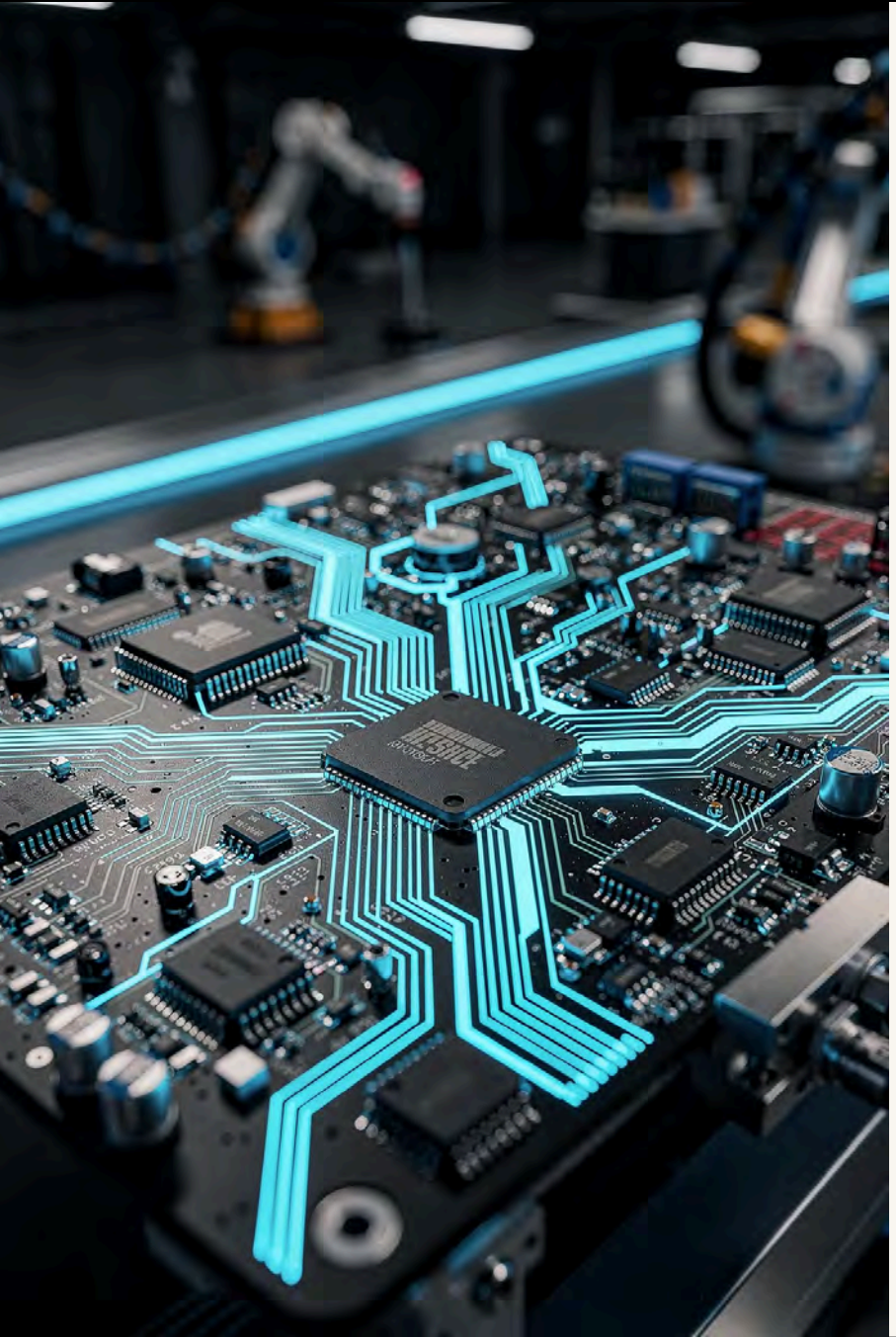


From Code Correctness to Behaviour Validation Detecting Defects Early in Embedded Automotive Software

By Sana Fatima
General Motors
Conf42 Golang 2026

The Core Challenge

Correct Code Is Not Enough

In embedded automotive software, a module can be **correctly written, compiled, and unit-tested** and still fail at the system level.

The gap is not syntax or logic. It is **behaviour**: what the system is expected to do versus what it actually does when integrated with other modules under real operating conditions.

The Stakes Are Rising

Chassis and powertrain platforms are becoming increasingly software-defined and tightly interconnected. Modern vehicles can contain over 100 ECUs running tens of millions of lines of code, with modules from multiple suppliers sharing buses and dependencies, so a single violated behavioural assumption can cascade into safety-relevant failures. As ADAS and electrification add further complexity, early behavioural validation is becoming both a competitive advantage and a regulatory necessity.

Why Defects Escape

Where Validation Falls Short

Unit Testing Stops at the Boundary

Individual modules pass all tests in isolation, but cross-module assumptions are never made explicit or verified.

Behavioural Intent Is Implicit

Design intent lives in documents and tribal knowledge not in executable artefacts that can be continuously checked.

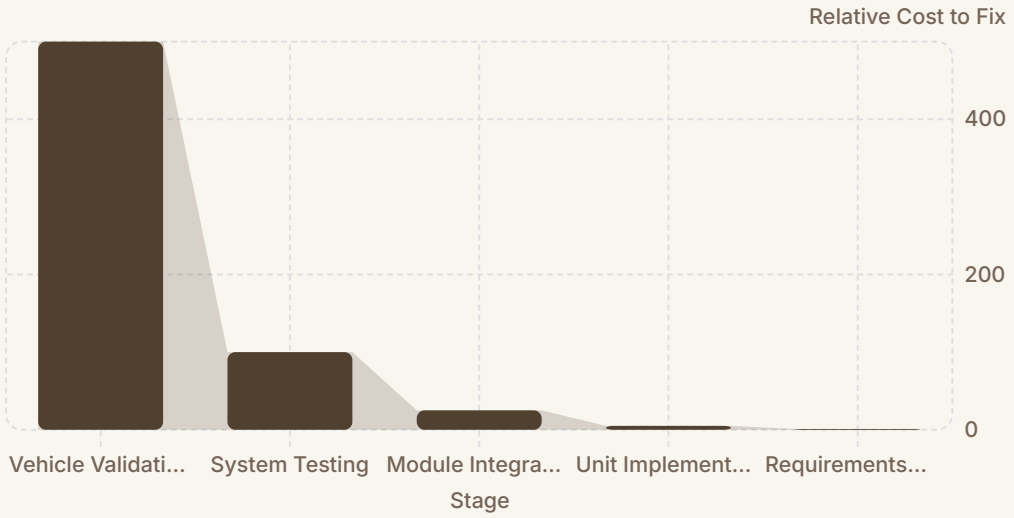
Integration Is Tested Too Late

System-level and vehicle-level testing surfaces defects at the point where diagnosis and rework are most expensive.

The Cost Curve of Late Defects

0.2%

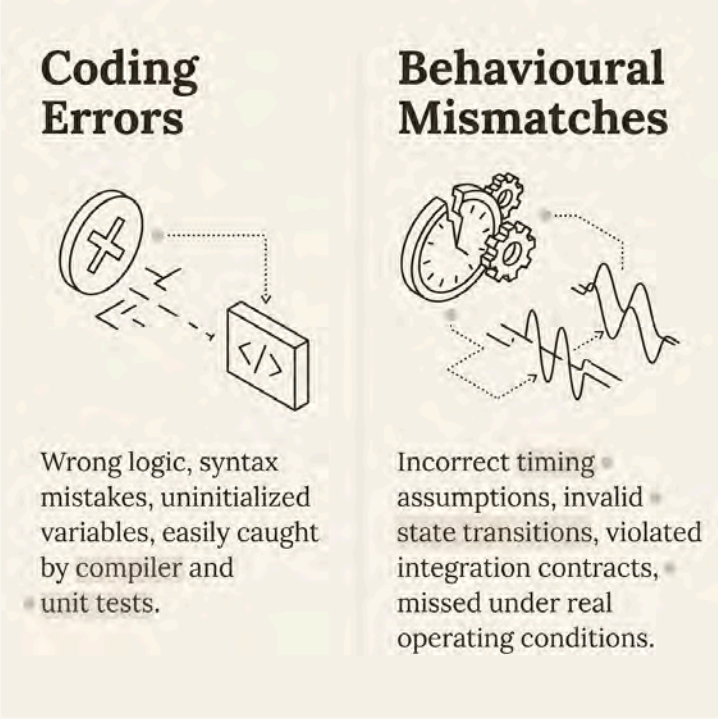
Total conversion rate



Defect remediation cost grows exponentially the later an issue is discovered. In automotive development, mismatches found at vehicle validation can cost **hundreds of times more** than the same defect caught at design time, triggering full regression cycles across supplier teams, hardware re-spins, and homologation delays.

Root Cause

Behavioural Mismatch vs. Coding Error



Modern toolchains are highly effective at catching coding errors. The emerging defect class in software-defined vehicles is **behavioural mismatch** where individually correct modules produce incorrect system-level outcomes.

These defects demand a fundamentally different validation strategy.

Shift validation left.

Define and verify expected system behaviour throughout design and implementation not only at integration.

In practice, that means embedding behavioural checks at the design and implementation phases instead of waiting for system integration. Mismatches between modules are caught when they are cheapest to fix, before they spread across the system.

The earlier a team can verify that code conforms to its behavioural contract, the smaller the blast radius of any defect reducing rework, protecting schedules, and building confidence in the system incrementally.

The Approach

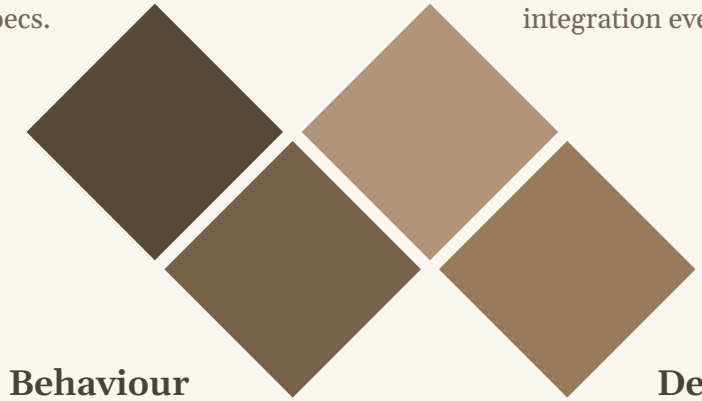
Behaviour-Driven Validation

Automate Checks

Generate executable behavioural tests from specs.

Validate Continuously

Run tests at every build and integration event.



Define Behaviour

Map explicit behavioural specs to design intent.

Detect & Trace

Trace defects back to original behavioural requirements.

By making behavioural expectations explicit and machine-checkable, teams validate intent and execution in lockstep throughout the development lifecycle.

Core Principle

Making Behavioural Intent Executable

Behaviour-driven validation bridges the gap between **what the system must do** and **what the code actually does** by turning natural-language intent into continuously executed artefacts. This creates a living, auditable record of conformance, which is critical for ISO 26262 safety argumentation.

Behavioural specifications are typically expressed in structured natural language, such as Gherkin-style Given/When/Then scenarios, or in formal representations like state machine descriptions and rule sets. These specifications are translated into executable test artefacts that map directly to expected system behaviour, creating a continuous feedback loop that surfaces violations immediately when implementation drifts from intent.

Three Pillars of the Framework

- **Explicit Behavioural Specifications**

  Define expected system responses, state transitions, and timing constraints as first-class engineering artefacts not documentation footnotes.

- **Continuous Validation**

  Integrate behavioural tests into CI pipelines so every commit is checked against the full set of behavioural expectations automatically.

- **End-to-End Traceability**

  Maintain a traceable link from design intent to specification to test result, supporting safety case construction and change impact analysis.

Integration Defects

Where Behaviour-Driven Validation Catches What Unit Tests Miss

- **Timing Violations**

  Latency and deadline constraints across module boundaries that are invisible to isolated unit tests.

- **Invalid State Transitions**

  Illegal state combinations that emerge only when multiple modules interact under specific sequences of inputs.

- **Violated Interface Contracts**

  Mismatched assumptions about signal ranges, data rates, or error-handling protocols between producer and consumer modules.

- **Fault Propagation Gaps**

  Fault conditions that are handled locally but whose system-level effects are never validated against safety requirements.

Safety-Critical Context

Alignment with Automotive Safety Standards

Behaviour-driven validation directly supports **ISO 26262** and **ASPICE** expectations for software verification:

- Bidirectional traceability from requirement to test evidence
- Verification of both functional correctness and behavioural robustness
- Regression coverage maintained across software releases
- Auditable test records for functional safety argumentation

Benefits at a Glance

Earlier

Defect Detection

Behavioural defects surface at design and integration time, not during vehicle-level testing.

Clearer

Design Intent

Explicit specs close the gap between what engineers intended and what the system delivers.

Greater

System Confidence

Continuously validated behavioural contracts reduce integration risk across complex platforms.

Key Takeaways

→ Unit testing is necessary but not sufficient

Behavioural mismatches between correct modules are the dominant defect class in modern software-defined vehicles.

→ Shift validation left with explicit behavioural specs

Executable specifications make design intent continuously verifiable throughout implementation.

→ Continuous behavioural validation reduces integration risk

Catching behavioural defects early dramatically lowers remediation cost and supports safety certification.

Thank You!