ARTICLE · SKA GLOBAL PARTNERS

AI Hallucinates Into Gaps: Why Requirements and Design Are the Guardrails That Matter Most

Senthil Parameswaran

Managing Director, SKA Global Partners

July 2026

After building production systems with AI agents, one lesson is clear: AI doesn't hallucinate randomly — it hallucinates into the gaps you left.

Executive Summary

After building production systems with AI coding agents, one pattern emerges with uncomfortable clarity: AI does not hallucinate randomly. It hallucinates into the gaps you left — missing requirements, stale documentation, undesigned decisions, and ambiguous architecture.

When the spec is precise, AI agents front-load work into cheap, correct increments. When the spec is absent or wrong, they generate confident, wrong code — and the cost of that confidence is paid later, in the most expensive phase of any project: rework, device testing, regression debugging, and deployment failures.

The economic argument is simple: requirements and design are cheap. Rework is not.

AI makes the cheap phase nearly free.
It makes the expensive phase brutally repeatable.

The Inversion Most Teams Miss

The standard narrative around AI development goes like this: AI makes you faster, so you can worry less about upfront planning and iterate your way to the right answer. That narrative is backwards — and it is costing teams real money.

Traditional development has a natural correction mechanism: developers notice when their mental model of the system is wrong. They pause, re-read the code, ask questions. This friction is inefficient, but it is also a

quality gate.

AI agents do not pause. They extrapolate from whatever context they have been given, fill the gaps with plausible-sounding assumptions, and generate code that compiles, passes the tests you wrote for it, and looks completely credible — right up until it hits the real system.

The corrective is not to use AI less. It is to invest more deliberately in the cheap phase — and to understand exactly what is cheap and what is not.

The Economics: Cheap Phase vs Expensive Phase

Every software project has two cost phases, but AI dramatically changes their relative weight.

The cheap phase is everything that happens before code: requirements documents, architecture decision records (ADRs), sequence diagrams, test vectors, explicit interface contracts. A sentence in an ADR costs almost nothing. A correct test vector can be written in minutes.

The expensive phase is everything that happens after a wrong decision has been encoded: rework loops, hallucinated API calls against outdated interfaces, confidently-wrong edits that propagate across a codebase, debugging sessions where the tests pass but the device fails.

With AI agents, the imbalance becomes acute: AI makes the cheap phase nearly free. An agent can synthesise a draft ADR from a conversation, enumerate edge cases from a spec, generate twenty test vectors from three examples. But it makes the expensive phase brutally repeatable — an agent will re-make the same wrong assumption across every file it touches, without fatigue, without doubt, and at speed a human rework cycle cannot match.

Six Lessons From Practice

These patterns emerged from building AI-native systems in production. The failure modes are universal.

1. Stale documentation is worse than no documentation.

AI agents trust what they are given. A specification or configuration file that is out of date is not a minor inconvenience — it is a lie the agent will act on with full confidence. In one case, an agent generated correct code against a deployment target that had been superseded. The code was well-structured, well-tested, and pointed at the wrong place. The fix: a live orientation script that queries the running system and reports its actual state. Source wins over docs.

2. A green test suite that skips the real path is the most dangerous failure mode.

An AI agent generates tests that are logically consistent with the code it just wrote. If that code is subtly wrong, the tests are wrong in exactly the same way — and they pass. In one case, a suite of unit tests ran green for weeks while a real-device integration test revealed that the core authentication flow was broken. At least one integration path must exercise the real system. If the only thing standing between you and a production incident is a test that cannot fail, you do not have a test suite. You have expensive reassurance.

3. Architecture-first, with verification vectors, beats trial-and-error measurably.

A week of AI trial-and-error produced working code that was architecturally wrong. The decision was then re-made with an ADR that specified the approach, documented the tradeoffs, and included explicit test vectors. The same problem was solved in a day. The difference was not the speed of the agent. It was the clarity of the brief.

4. Two competing mental models will produce one incoherent codebase.

When a system's architecture exists in two conflicting forms — legacy code vs. new design — the agent mixes them. It has no way to know which model is canonical. This manifests as code that is correct within each file but inconsistent across files. Make the canonical architecture explicit in one place before the agent starts. If the architecture has changed, update the ADR before touching the code.

5. Silently-swallowed failures turn five-minute bugs into hour-long expeditions.

AI agents tend to generate optimistic code — errors get caught, logged, and swallowed. The system continues in a degraded state. When something finally fails visibly, the root cause is several steps removed. Treat silent error handling as a bug, not a feature. Errors should fail fast, loudly, and with enough context to be actioned.

6. Small verifiable truths drift constantly, and someone must own their accuracy.

File paths, endpoints, token formats, schema shapes — the agent works from a snapshot. When those truths change, the agent doesn't know. It continues operating against the old truth, generating plausible code that is quietly wrong. A machine-readable source-of-truth catalogue, updated as part of every change and queried by the agent before any integration point, is the fix.

What Actually Worked

These six failure modes were encountered, diagnosed, and resolved. The practices that resolved them share a common thread: they move uncertainty from the agent's execution to the human's upfront decisions.

Source-derived orientation over documentation.

Before any agent session, run a script that queries the live system — build outputs, deployment targets, active configuration, real endpoints. Feed the results into the context. The agent works from truth, not memory.

Root-cause analysis before fix.

When something fails, do not ask the agent to fix it. Ask it to explain why it failed and which assumption was wrong. Fix the assumption first. This prevents the same wrong assumption from being re-encoded in the fix.

Architecture-first for complex components.

For any component involving cryptography, authentication, or third-party integration, write the ADR before writing any code. The agent's implementation is then a mechanical translation of a human decision.

Real-device verification as a hard gate.

Unit tests are necessary but not sufficient. Before any feature is considered complete, it must pass on the real target. This is slow, which is exactly why it must be a gate rather than a best-effort check.

Machine-checked guards turning incidents into pre-push failures.

Every incident that makes it to production contains a fact that could have been checked automatically before deployment. Build that check. A lint rule, a pre-commit hook, a CI validation step. Incidents are expensive. The check is free.

The Economic Case, Restated

AI-native development does not reduce the need for engineering discipline. It raises the stakes for it.

A disciplined team using AI will outperform a disciplined team without it. The cost of requirements, design, and verification stays roughly constant; the cost of implementation collapses. The economic leverage is real and significant.

An undisciplined team using AI will underperform an undisciplined team without it. The cost of rework multiplies because the agent encodes wrong decisions faster, at greater scale, with greater consistency. The errors are not obvious — they are credible, well-structured, and fully tested against the wrong spec.

The teams that are already pulling away from the field are not the ones that gave their agents more autonomy. They are the ones that invested in being precise about what they wanted, before they asked for it.

An AI agent is a brilliant contractor with no memory of yesterday and total faith in whatever you hand it.

Give it a clear blueprint and it builds fast, correctly, and at a cost that would have been unthinkable five years ago.

Give it a lie — and it builds that lie faithfully, completely, and at speed. Then it bills you to tear it down.

Senthil Parameswaran is Managing Director at SKA Global Partners. SKA Global advises boards, CIOs, and technology leadership on AI-native transformation, software architecture, and technology strategy.