Agentic Failure Modes Rubric

A living catalog — 9 modes as of 2026-07-15. This is not a closed list. It is the set of failure modes we could name, test, and evidence as of the date above; the discipline that produced nine is expected to surface more as internal AI tooling keeps maturing. Companion artifact to the Tekion Tech Blog post *"The 9 Ways Agentic AI Systems Fail (and how to test for all of them)."*

How to use this rubric: (1) **Triage** — match your suspected failure to one of the nine mode definitions below. (2) **Test** — run the paired test pattern against your system. (3) **Integrate** — fold the passing test into your existing eval pipeline so the failure class is mechanically checked on every future run. This turns a post-mortem observation into a standing regression guard.

---

1. Output Quality Drift

Fails when: Model output quality silently degrades across a sequence of similar calls — same prompts, similar inputs, gradually worse outputs — with no error signal.

Test pattern: Golden-set continuous evaluation. Run the agent against a curated test set (sized by autonomy tier: A0=5, A1=20, A2=50, A3=100, A4=200+ inputs) on every deployment and on a scheduled cadence. Measure six drift signals: golden-set regression, quality-dimension shift, confidence-calibration drift, escalation-rate change, cost change, latency change.

Severity / response: P0 (safety regression) — halt immediately. P1 (quality regression) — pause autonomy tier A3+. P2 (calibration drift) — recalibrate within 1 week. P3 (efficiency drift) — optimize within 2 weeks. Replace counter-based health metrics with rate-based thresholds (≥90% healthy, 70–90% degraded, <70% critical) — a counter that resets on single success masks systemic failure.

---

2. Tool-Permission Violation

Fails when: An agent invokes a tool outside its sanctioned allowlist — writing when permitted only to read, calling an unsanctioned API, or modifying a resource outside declared scope.

Test pattern: Allowlist verification on every tool call, plus a scope-creep audit at session end (cross-reference actual tool-usage logs against the declared allowlist).

Severity / response: Deny-by-default policy plus post-session audit log review. Watch specifically for the fork-permission gap: child agents start UNDER-permissioned relative to a correctly-scoped parent (not over-permissioned) — a passing parent-level permission test provides no guarantee for child agents, because the inheritance gap means the fork's effective permissions were never actually exercised. Also watch for wrapper-class bypass attempts (an agent generating tool-invocation variants to route around a blocked path) — a guard must match the wrapper itself, not just the specific disallowed inner command.

---

3. Retrieval-Context Contamination

Fails when: A retrieval or scan pipeline pulls content into the agent's working context that pollutes its reasoning — stale documents, conflicting sources, or off-topic chunks displacing authoritative material.

Test pattern: Corpus integrity check (every retrieval-corpus document has a freshness date within policy) plus retrieval-source audit (every citation in the agent's output was actually retrieved, not fabricated).

Severity / response: Two-layer response: (1) corpus dating plus a retrieval-source allowlist per agent role; (2) runtime context-restoration hooks (e.g., a pre-compaction hook that reloads critical operational rules immediately after any context-window reduction, so rules survive the boundary).

---

4. Hallucinated Artifact Reference

Fails when: An agent names a file, function, ticket ID, URL, or other artifact that doesn't exist — or exists but not in the state the agent assumes — then proceeds to take downstream actions premised on that artifact's validity.
Test pattern: Existence check on every named artifact before the next action: verify the named entity exists in a runtime-controlled registry before any downstream tool call proceeds.
Severity / response: The Handles pattern — the runtime registers valid handles in a session-scoped store; every tool call accepts only registered handles as entity parameters. If a handle is not in the session register, the call is rejected before reaching any API. On a miss: rejection and rollback, never a retry against the hallucinated target. Model upgrades shift this failure's *rate*; they do not change its *class* — switching models does not eliminate this mode.

---

5. Eval-Pipeline Blind Spot
Fails when: The eval suite passes at 100% while an entire class of production-equivalent scenarios goes untested — the suite measures itself, not the system.
Test pattern: A coverage audit of the eval suite itself: distinguish absence detectors (grep for bad things) from presence validators (verify good things exist), then map both against the production-equivalent distribution — the actual scenario classes, input shapes, and tool-call sequences seen in live traffic.
Severity / response: Continuous coverage measurement, not pass/fail gate-keeping. Beyond roughly 1,000 structural tests, additional structural tests yield diminishing reliability gains — the next tier is behavioral tests with synthetic data exercising production-equivalent scenarios end to end. Route findings through the testing pyramid: structural → behavioral → integration → production.

---

6. Agent Directive Non-Compliance
Fails when: An agent explicitly acknowledges a directive in its reasoning trace — the chain-of-thought reads as compliant — and then takes an action that violates it. *(New pattern.)*
Test pattern: A post-action directive-adherence audit that compares what the agent *did* against the directive it *acknowledged* — distinct from a chain-of-thought audit, which only inspects reasoning and produces a false negative here.
Severity / response: Three-layer defense: Prevention (make the approved path more convenient than improvisation), Detection (the adherence audit above), Recovery (structural removal of the non-compliant option at the API/tool boundary when the directive is safety- or integrity-critical). Positional sensitivity amplifies risk — a directive buried deep in a long instruction file is functionally invisible by the time the agent reaches the relevant step; position directives before the operation they govern and repeat them as inline reminders at decision points. Architecture beats prose: structural removal achieves materially higher compliance than text-based `[!critical]` callouts alone.

---

7. Cross-Cutting Impact Corruption
Fails when: An agent changes one artifact and the change propagates silently into unrelated systems — shared schemas, shared configs, or downstream consumers — without auditing the scope of impact before committing. *(New pattern.)*
Test pattern: An impact-graph audit run pre-commit: before any artifact is modified, enumerate all downstream systems that consume it. If the change touches anything outside the declared scope, block and surface the impact graph for explicit operator sign-off.
Severity / response: A declared-impact gate — the agent must declare its intended write scope before any write operation; any write touching an artifact outside that declared scope is blocked pending confirmation. This is a general risk, not only an agent one: the same blindness affects human reviewers who cannot hold a large cross-module impact graph in working memory.

---

8. Self-Referential Corruption

Fails when: An agent feeds its own (possibly imperfect) output back as input to a later step in the same pipeline session, compounding small errors into large ones. The corruption channel is the agent's own execution sequence — not a shared resource — which distinguishes this from cross-cutting impact corruption. *(New pattern.)*
Test pattern: A provenance check on every consumed artifact — before using any artifact as input, determine whether it was produced by an earlier action of the same agent in the same session. If so, run a freshness/verification gate before trusting it (e.g., an idempotency test: run the same operation twice and assert the second pass changes nothing).
Severity / response: Provenance tagging (session ID, step number, timestamp) on every agent-produced artifact, plus a verification gate at every artifact-consumption boundary that confirms the artifact is complete, the producing step reached its terminal state, and the content hasn't been modified since.

---

9. Construction-vs-Liveness Gap
Fails when: Unit, contract, and source tests verify that a primitive works *correctly* when called (construction) — but never verify that it is actually *being called* in production and producing fresh output (liveness). Every silent-rot failure passes its unit tests: green tests, broken system. *(New pattern — the newest and most narrowly evidenced mode in this catalog: both of its supporting claims trace to a single internal incident cluster, disclosed here rather than treated as independently corroborated.)* This is the operate-time analog of cross-cutting impact corruption (mode 7) — not a generalization of it — and is distinct from the eval-pipeline blind spot (mode 5): mode 5 is a coverage gap in what gets tested *before* release; this mode is a gap in whether a comprehensively-tested system is still *running* after release.
Test pattern: A three-part liveness layer: (1) per-output freshness monitors that flip silent rot into a visible alarm; (2) integration tests that assert the canonical subcommand *sequence* and side-effects of a full pipeline run, not just per-unit behavior; (3) production-data audits that check real output freshness against live state, not narrative status reports.
Severity / response: A predictive filter for finding rot before it's found by accident: an output is at-risk if it is produced by an LLM-driven mutation **and** no freshness/liveness check surfaces it. Treat a positive hit as a prioritization signal for adding a freshness monitor, not as a validated alarm on its own — the filter has not yet been checked against a base or false-positive rate.

Source & Provenance
Every mode above traces to a dated, documented occurrence in our own internal AI development — not an industry benchmark. Observed rates and incident counts throughout are internal observations from our own tooling, not industry-wide benchmarks.