RESEARCH PROPOSAL · SOFTWARE ENGINEERING

Structural Fidelity as a Control Mechanism for Human-AI Software Development

Characterizing and Controlling Compounding Drift in Agentic Software Development

---

T2D2 — Test² Telemetry² Driven Development, and its reference implementation, Poka-yoke

Prepared by: Tyrone Thomas

BreakThrough Technologies LLC

linkedin.com/in/tyrone-thomas-6b16833

> **Document status:** *working draft, prepared for review and feedback. The evidence base and methodology are under active development — sections requiring additional support before this proposal is ready to guide funded or partnered research are flagged throughout, and consolidated in Section 11.*

Table of Contents

1. Introduction

1.1 Motivation

"Vibe coding" — a term popularized by Andrej Karpathy in February 2025 and named Collins Dictionary's 2025 Word of the Year — describes a mode of software development in which a human describes intent in natural language and an AI agent produces the implementation with minimal direct inspection of the generated code. In practice this mode works well on small, self-contained scripts and degrades, often silently, as codebase size and architectural complexity grow. The degradation is not well explained by the generating model becoming less capable; it is better explained by the absence of any mechanism in the workflow that signals when the shape of what is being built has drifted from what was originally asked for.

This drift does not announce itself. It surfaces later as a defect no one can easily trace to its origin, or it does not surface at all and simply becomes undocumented technical debt. The review mechanism that has historically caught this kind of problem — a human reading a diff before it merges — does not scale to the volume and velocity of change that agentic development now produces.

The organizing question behind this proposal borrows directly from Deming's quality-engineering tradition, applied to software seventy years after its origin on the factory floor: what is the software-development equivalent of a physical jig or gauge — a mechanism that mistake-proofs a process by its shape, rather than inspecting output after the fact?

1.2 The Motivating Claim, Verified Rather Than Assumed

Before treating the premise above as a live problem worth researching, the claim that industry practice is already reducing or eliminating human code review was checked against current sources rather than carried forward as anecdote. Four points of evidence, of varying strength, support treating this as a real and current debate:

- A named academic position: Monperrus (arXiv:2606.13175, June 2026), "The End of Code Review," argues that coding agents have crossed a threshold beyond which human review is no longer a necessary gate.
- Industrial corroboration at scale: reporting on Cursor's usage patterns describes agents pushing changes to production without a separate human review step; Cursor was reported near a $30B valuation with $1B+ annualized revenue by late 2025.
- A public, unaudited extreme case: the company Webaroo has publicly claimed 23 production features shipped in 60 days with zero human developers, with specifications rather than code as the artifact under review. This claim is cited here as illustrative of the rhetoric in the space, not as verified fact.
- Forecasting pieces in industry trade press describe "autonomous maintenance pull requests" — opened without human initiation — as a near-term standard feature of AI-assisted review platforms.

Having verified that this debate is real and current, the remainder of this proposal argues against a specific, named position (Monperrus's claim that the shift has already happened and is net beneficial)

rather than against a generalized industry mood. This is intended as a stronger and more falsifiable stance than a vaguer claim would allow.

> **NOTE ON SCOPE** — *The evidence above is graded by strength throughout this document (see Section 8 and Section 11). The Webaroo claim in particular is unaudited and should not be treated as equivalent in weight to the peer-reviewed or large-sample evidence cited elsewhere.*

1.3 Statement of the Problem

The central problem this proposal addresses has two parts. First, a descriptive claim: under sustained, low-control AI-agent iteration, a codebase does not merely accumulate ordinary technical debt but is hypothesized to undergo a qualitatively different, compounding form of decay, driven by several distinct and nameable mechanisms rather than one generalized process of "getting worse." Second, a prescriptive claim: a methodology built around a pre-code structural proposal, checked deterministically against implementation and subsequently correlated against runtime behavior, is hypothesized to measurably slow or interrupt this compounding process relative to review regimes that rely purely on human judgment or purely on agent self-assessment.

Both claims are stated here as hypotheses. Neither has yet been tested against real data at the time of this writing. This proposal exists to scaffold the research program that would test them, and to be explicit throughout about which parts of the argument currently rest on verified evidence, which rest on evidence that is real but contested, and which rest on reasoning or anecdote that has not yet been tested at all.

1.4 Contribution Preview

The intended contributions of this research program, pending the evidence described in the sections that follow, are:

- A three-artifact model (structural graph, behavioral test manifest, and runtime execution trace) unified by a shared identifier scheme, extending architecture-conformance checking to a pre-code stage and forward into production observability.
- A named, falsifiable position that engages directly with a current, contested claim in the literature (Monperrus, 2026) rather than a general argument against reduced human oversight.
- A control-type / warning-type framing, borrowed from Shingo's poka-yoke distinction, that yields a specific, testable prediction about the shape of failure under different verification regimes — not merely their rate.
- A scaffolded, multi-cycle empirical research program, prioritized by cost and citability, rather than a single-point before/after comparison.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — Evidence for all four contributions** *is currently at the hypothesis or early-design stage. None of the four bullets above should be read as an established result. See Section 6 for the specific research questions and Section 11 for a consolidated list of what would need to be true, or measured, before each claim could be defended.*

2. Related Work

This proposal is positioned at the intersection of three lineages of practice that have not, to the author's knowledge, previously been combined into a single mechanism. Each is summarized below, followed by a review of specific technical work most relevant to the proposed system, and a positioning statement that names what is and is not yet verified about the novelty claim.

2.1 Three Converging Lineages

Quality and lean manufacturing (~100 years)

The lineage running from Shewhart through Deming to the Toyota Production System supplies both a philosophical foundation and, through Shigeo Shingo's concept of poka-yoke (mistake-proofing a process by its shape rather than inspecting output afterward), a name for the proposed tool. Deming's distinction between a control mechanism, which halts a faulty process automatically and requires no ongoing human vigilance, and a warning mechanism, which merely signals a problem and depends on a person noticing it, is used throughout this proposal as the primary theoretical lens (see Section 3.6).

Software methodology (~25 years)

The lineage running from Extreme Programming and Test-Driven Development through Agile, DevOps, Site Reliability Engineering, and GitOps supplies the precedent for splitting a free, vendor-neutral methodology (here, T2D2) from a commercial tool that implements it (here, Poka-yoke) — following the Agile/Jira and Infrastructure-as-Code/Terraform precedents.

The agentic-coding moment (~2 years)

The lineage running from GitHub Copilot through the "vibe coding" moment supplies the urgency: a fast-growing, currently under-served problem space, in which terms like "architectural drift" and "spec-code drift" have only recently been named and are not yet the subject of a mature research literature.

2.2 Specific Technical Work

Category	Representative Work	Where This Proposal Differs
Program representation	Joern / Code Property Graph (2014; IEEE Test-of-Time award, 2024)	*Describes code that already exists; has no pre-code proposal role.*
Pre-code grounding	Spec Kit Agents (arXiv:2604.05278); De-Hallucinator (arXiv:2401.01701); pre-execution exploration approaches	*Grounds generation in evidence, but as natural-language or prose artifacts rather than a diffable structural graph.*
Spec-driven agentic orchestration	Intent; Kiro (AWS)	*Coordinates agent work around a specification artifact, but not, on current evidence, a compiler-verified structural graph. Treated in this proposal as potentially complementary rather than competing — see Section 2.3.*

Category	Representative Work	Where This Proposal Differs
AI-assisted diff review	ARCTIC (Meta; arXiv:2607.29516, 2026); Aviator Verify; Tricentis AI Workspace	*Reacts to a change already proposed; the check happens after the fact, not before implementation begins.*
Architecture conformance	NDepend; Sonargraph; Lattix; CodeQL; CodeScene	*Static, post-hoc rule-checking. Several of these tools have recently begun exposing findings to agents via MCP, which narrows but does not eliminate the timing gap.*
Runtime observability	Datadog; New Relic; Honeycomb; Multiplayer	*Monitors a system after deployment; not correlated back to a pre-code structural proposal.*

2.3 Positioning Statement

The honest positioning claim is narrower than "no comparable tool exists." It is: no work reviewed so far combines a compiler-accurate structural graph, live telemetry correlation to that graph, and a pre-code comprehension ceremony within a single feedback loop spanning the full lifecycle from before code exists through production operation.

Intent and Kiro are best categorized as spec-driven agentic orchestration development solutions: they coordinate agent work around a specification artifact. On current evidence, that artifact is not a compiler-verified structural graph, and neither system appears to offer the same level of deterministic control that T2D2's Fidelity Check is designed to provide. This proposal treats the relationship as complementary rather than adversarial, along two specific integration paths:

- T2D2 as a precision layer for existing specs — integrating Poka-yoke's graph discipline into a spec-driven orchestration workflow could make the specs those systems produce more precise and canonically specified, rather than requiring T2D2 to displace them.
- Specs as graph-generation input — the specification a system such as Intent or Kiro produces could serve as the input from which an agent generates the T2D2 graph. That graph, once proposed, becomes the artifact the human and the agent explicitly agree upon before Build begins, and it subsequently becomes the reference Fidelity Check compares the completed work against once Build is done.

This reframes the earlier open question about Intent and Kiro from "which system wins" to "at which layer does deterministic verification enter a spec-driven workflow." Section 4.5 describes the proposed integration architecture in more detail.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — The complementary-integration model described above** *is currently a proposed architecture based on the author's characterization of Intent and Kiro as spec-driven orchestration systems, not yet validated against their actual spec output formats or APIs. Confirming that a real Intent or Kiro specification can be mechanically or semi-mechanically translated into a T2D2 graph — rather than requiring substantial manual re-authoring — is the concrete next step needed to move this from a proposed architecture to a demonstrated capability.*

3. Central Thesis and Theoretical Framework

3.1 Lead Thesis

> *Under sustained, low-control AI-agent iteration, a codebase does not merely accumulate ordinary technical debt — it ultimately undergoes catastrophic decay, arriving there through several specific, nameable mechanisms rather than generalized "getting worse." Reducing or removing human review removes the last interruption capable of stopping this before it compounds.*

This is a claim about an eventual, mechanistically-driven end state under sustained conditions, not a claim that every codebase, at every point in time, is actively collapsing. An earlier, cruder version of this thesis — "software always catastrophically collapses" — was identified during drafting as an overstatement of the intended claim and was explicitly retracted rather than merely softened. The version stated above is the one this proposal defends.

3.2 Corollary: The Proposed Answer

A methodology built around a pre-code structural proposal, checked deterministically against implementation and correlated forward into runtime behavior, is proposed to measurably slow or interrupt this compounding process, relative to review regimes that are purely human-judgment-based or purely agent-judgment-based. This corollary is what T2D2 and its reference implementation, Poka-yoke, are built to demonstrate.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — Both the lead thesis and its corollary** *are currently unverified hypotheses. No empirical study has yet been run. Section 6 states the specific, falsifiable research questions this proposal would need to answer to defend either claim, and Section 7 proposes a method for answering them.*

3.3 Five Mechanisms of Compounding Drift

Rather than treating "drift" as a single undifferentiated process, this proposal decomposes it into five distinct, individually testable mechanisms:

- Performance degradation — redundant paths, deep call chains, and patched-over inefficiencies accumulate across iterations and degrade runtime performance.
- Intractability of overlapping criteria — the interaction space among features and constraints grows combinatorially; a bounded context window is hypothesized to make this sharper for an individual agent than it typically is for a human team drawing on distributed institutional memory.
- Proliferation of parallel, redundant structures — inconsistent implementations of the same underlying concern accumulate because introducing a new, local mechanism is cheaper in the moment than locating and extending the existing one.
- Patching over root causes — successive fixes address symptoms rather than root causes because diagnosing the root cause costs more, in the moment, than applying a local patch.
- The open-ended telephone game — no final comparison step exists in an ongoing agentic workflow; each iteration's output becomes the next iteration's input indefinitely, with no built-in point at which

accumulated error is checked against an original reference. This mechanism is the source of the control-type/warning-type framing developed in Section 3.6.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — Mechanisms 1–5 above** *have not yet been measured individually. The research questions in Section 6.1 (RQ-D1a–e) are designed to test each mechanism separately rather than assuming they behave identically or contribute equally to overall drift.*

3.4 Why Agent Scale May Amplify These Mechanisms

Several reasons are proposed, at the level of hypothesis, for why these mechanisms might compound faster or more severely under agent-driven development than under traditional human-driven development:

- Volume and velocity — even at an identical per-change probability of introducing drift, an agent's substantially higher throughput compounds the effect faster in real time.
- Bounded context versus institutional memory — a human team's distributed knowledge degrades gradually; an agent's working context is bounded and, absent an explicit mechanism, does not persist between sessions by default.
- Local pattern-matching over global awareness — an agent operating in a limited context tends to extend the nearest available pattern; if that pattern is already drifted, this tendency actively perpetuates the drift rather than correcting it.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — The fourth candidate explanation** *— that an agent's absence of durable personal or reputational stake in the codebase affects its propensity toward mechanisms 3 and 4 above — is explicitly flagged as speculative. No evidence for or against this specific explanation has been identified, and it should not be treated as more than a hypothesis worth testing until direct evidence is found.*

3.5 A Further Question: Can an Agent Be Instructed Into Compliance?

A prior question to how drift accumulates is whether simply instructing an agent to avoid it (in this proposal's terms, "Shop Rules") is sufficient on its own. Several findings from the broader literature on instruction-following suggest this cannot be assumed:

- Explicit instructions such as "do not introduce inaccuracies" have been observed to measurably increase summarization errors in some settings — an algorithmic analogue of ironic process theory (the "don't think of a pink elephant" effect).
- A constraint that was already self-evidently satisfied has, in some reported cases, caused task failure when stated explicitly, on a task that succeeded without the explicit statement.
- Some models have been observed to shift behavior in the direction opposite an explicit stated influence, while simultaneously denying that any influence occurred.

None of the specific findings above are coding-specific in the sources reviewed so far. This absence of a coding-specific replication is treated in this proposal as a reason for more caution, not less: there is currently no evidence that the effect is confined to a domain that T2D2 does not touch.

The resulting position is stronger than "instruction is a weaker control than a deterministic gate." It is that persistently instructing an agent to follow a rule may, in some conditions, be actively

counterproductive. If reinforcing a rule can increase the rate at which it is violated, no amount of better-worded instruction closes that gap on its own — the gap is argued to be structural to instruction-following itself, which forms the architectural justification for an external, deterministic verification gate rather than a purely instructional one.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — This entire line of argument (Section 3.5)** *rests on literature about instruction-following in general-purpose language-model settings, not on coding-specific studies. RQ-A1 through RQ-A4 (Section 6.2) are designed to test whether these effects replicate in a coding-agent context specifically. Until that replication exists, Section 3.5 should be read as motivating evidence for building an external control, not as proof that instruction-only approaches fail in this domain.*

3.6 Control-Type Versus Warning-Type Verification

This proposal's central theoretical device is Deming's distinction, inherited from Shingo, between a control mechanism and a warning mechanism. A control mechanism halts a process automatically at the point of deviation and requires no ongoing human vigilance to function. A warning mechanism merely signals that something may be wrong and depends on a person, or another judgment-making process, noticing and acting on that signal.

Applied to software verification, human code review — and, notably, an agent asked to self-assess whether its own output "looks right" — are both warning-type mechanisms. Each functions like an analog relay: it passes along whatever error has already accumulated from earlier stages, and while it may catch a new error introduced at its own stage, it has no structural mechanism for detecting earlier error that happens to look locally plausible.

T2D2's Fidelity Check, by contrast, is proposed as a control-type mechanism. It is intended to function like digital regeneration in a communications network: at each checkpoint, the current state is compared against a clean, persistent reference (the structural graph), and a clean signal is regenerated rather than accumulated noise being passed forward.

This framing yields a specific, testable prediction: the two verification types should produce a difference in the shape of failure over time, not merely its rate. A warning-type regime should show errors compounding smoothly across iterations, resembling generation loss in analog copying. A control-type regime should show error resetting toward the reference at each gate, with any residual drift attributable specifically to whatever the gate's scope does not cover.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — The claim that Fidelity Check functions as a genuine control-type mechanism** *depends on a design requirement that is not yet fully guaranteed by the current implementation: the check must execute in a pipeline the agent cannot alter, and its results must be delivered through a channel the agent cannot fabricate or bypass. Until this is independently verified (see Section 4.4), Fidelity Check should be treated as a mechanism designed to behave as control-type, not as one confirmed to do so under adversarial or careless conditions.*

4. The Proposed System: T2D2 and Poka-yoke

T2D2 (Test² Telemetry² Driven Development) is the methodology; Poka-yoke is its reference implementation. The split follows established precedent (Agile/Jira; Infrastructure-as-Code/Terraform): a free, vendor-neutral discipline intended to become shared vocabulary, paired with a commercial, differentiable tool that implements it. T2D2 is deliberately agent-architecture-agnostic — it assumes neither a single generalist agent nor a specific division of labor across specialized agents.

4.1 The Three-Artifact Model

T2D2 maintains three deliberately distinct artifacts, unified by a shared node-identifier scheme rather than blurred into a single representation:

- The Graph — answers "what should exist." A structural JSON representation of call shape and data shape, produced by compiler-based analysis (for JVM/Kotlin) or parser-based analysis (for Python/JavaScript). It is proposed before code is written, regenerated and diffed against that proposal once code exists, and becomes the live map once the system is deployed.
- The Test Manifest — answers "what should happen." A behavioral contract expressed in Given/When/Then form. It is a separate, ordinary artifact checked at the same gate as the Graph, but not node-correlated to it.
- The Execution Trace — answers "what actually did happen." A structured runtime record correlated to Graph nodes via a stable, deterministic identifier; a parent-span identifier enables call-chain reconstruction across distributed execution.

> **NOTE ON SCOPE** — *The Graph is the load-bearing artifact of the model: it plays three sequential roles — proposal, diff target, and production map — rather than three disconnected tools requiring separate mental models. This single-artifact-multiple-roles design is the methodology's core structural claim and has not yet been evaluated against alternative designs that might use separate artifacts for each role.*

4.2 The Ceremony Lifecycle

The methodology is operationalized as a sequence of named ceremonies mapped explicitly onto Deming's Plan-Do-Check-Act cycle:

Stage	Ceremonies	Description
Plan	**Sketch / Blueprint**	Brief → Survey → Calibration → Formalization. The agent proposes a target graph edit; human sign-off gates the Build stage.
Do	**Build**	Implementation proceeds against the approved plan. Exit requires passing both the Fidelity Check and the Test Manifest.
Check	**Fidelity Check / Punch List**	The regenerated graph is diffed against the original proposal. This check is structural only, and is intended as a diagnostic signal, not a performance quota.
Act	**Handover → In Service**	The system is deployed; deterministic identifiers are injected; the Graph becomes the live map against which traces are plotted.

Three further ceremonies extend the same graph-plus-trace pairing beyond initial deployment:

- Service Call — debugging, via reconstruction of the call chain using the parent-span identifier back to a diagnosed root cause.
- Renovation — refactoring guided by actual production usage evidence rather than assumed usage patterns.
- Walkthrough — onboarding, in which a new contributor reads the Graph for structure and the traces for behavior together.

4.3 Current Implementation Status

As of this writing, the following components are built and working: graph generation, graph editing and diffing, and a live telemetry pipeline for JVM/Kotlin systems via an ADB proxy. The following are believed feasible but not yet built: the equivalent telemetry pipeline for web and other non-JVM targets, and heavier pre-merge review corresponding to the Punch List ceremony. The three post-deployment ceremonies described in Section 4.2 (Service Call, Renovation, Walkthrough) are specified in design but have not yet been exercised on a real production system.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — The system described in this section** *is an alpha-stage tool, not a validated product. Claims made elsewhere in this proposal about what the system does or prevents should be read as claims about its intended design, not as claims that have been confirmed against real, adversarial, or large-scale usage.*

4.4 Design Requirement: Agent-Tamper-Resistant Verification

For Fidelity Check and the Test Manifest to function as genuine control-type mechanisms in the sense developed in Section 3.6, their execution must occur in a pipeline the agent under evaluation cannot alter, and their results must be delivered through a channel the agent cannot fabricate or suppress. An earlier draft of this proposal treated sycophancy — a model's tendency to tell an evaluator what it wants to hear — as a potential silent failure mode for these checks. On closer analysis this framing was corrected: the detection logic itself is deterministic and does not depend on the agent's honesty. The real exposure is narrower and more specific: whether the agent controls the invocation or reporting of the check, and whether it can construct a test that passes trivially without exercising the intended behavior.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — Whether the current implementation actually satisfies this design requirement** *has not been independently verified. This is treated in this proposal as a precondition for the control-type claim in Section 3.6, not as an already-met requirement. Verifying it — for example, through an adversarial evaluation in which an agent is deliberately incentivized to defeat the check — is one of the concrete next steps identified in Section 9.*

4.5 Compatibility with Spec-Driven Orchestration Frontends

Section 2.3 proposes that specifications produced by spec-driven agentic orchestration systems, such as Intent or Kiro, can serve as the input from which an agent generates a T2D2 graph, rather than treating

such systems as competitors to be displaced. Under this integration model, the workflow described in Section 4.2 gains an optional upstream step: a spec-driven frontend produces a specification; an agent consumes that specification during the Sketch/Blueprint ceremony and proposes a target graph derived from it; the human and agent then reach the same sign-off agreement on that graph described in Section 4.2; and the Fidelity Check performed at the conclusion of Build compares completed work against that same graph, regardless of whether the graph originated from a spec-driven frontend or from a Blueprint ceremony conducted without one.

Under this model, T2D2 does not compete with a spec-driven orchestration layer for ownership of intent capture; it is proposed to supply the deterministic, compiler-verified comparison step that such systems do not, on current evidence, provide on their own. Improving the precision and canonical structure of the upstream spec — a stated goal of integrating Poka-yoke's graph discipline into such a workflow — is proposed to improve the fidelity of the graph an agent subsequently derives from it, though this improvement has not yet been measured.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — This integration model** *has not been implemented or tested against either Intent or Kiro. No spec-to-graph translation mechanism currently exists. Before this model is presented externally as a demonstrated capability rather than a proposed architecture, a prototype should be built against at least one real specification produced by one of these systems, and the resulting graph's fidelity should be assessed against a graph produced independently through T2D2's own Blueprint ceremony for the same intended change.*

5. Toward a Formal Model of Drift

Three structures are proposed as candidate formal models of the compounding-drift process described in Section 3. None has yet been fitted to real data, and none should be read as more than a structure to be instantiated once the studies in Section 6 and Section 7 produce measurements.

5.1 Lehman's Laws as a Baseline Dynamic

Lehman's Laws of Software Evolution propose that complexity increases and quality declines over a system's lifetime unless actively countered. The empirical record for this claim is real but mixed: Yu et al. report a correlation consistent with the laws in their study of Tomcat and Ant, while Olbrich et al. report non-monotonic, phase-dependent patterns in Lucene and Xerces that complicate a simple reading of inevitable decline. This proposal's position is that Lehman's Laws describe a well-established tendency, possibly steepened under agent-driven development, rather than a universal law of collapse. Overclaiming beyond this position would not be supported by the cited literature.

5.2 Stock-and-Flow Modeling

A stock-and-flow structure is proposed in which a drift-severity stock at iteration cycle t is increased by a reinforcing loop — drift in the current cycle degrades the context available to the agent in the next cycle, which increases the probability of further drift — and decreased by a balancing loop representing whatever control regime is in place. The reinforcing loop is largely absent from Lehman's original, human-only baseline and is proposed here as this research program's actual novel theoretical contribution, rather than the mere application of Lehman's Laws to a new context.

5.3 Signal Degradation Across a Relay Chain

The most ownable framing device in this proposal borrows from information theory's treatment of error accumulation across a communication relay, and from the everyday phenomenon of generation loss in analog copying. As developed in Section 3.6, this framing predicts a testable difference in the shape of failure between control-type and warning-type verification regimes, not merely a difference in magnitude.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — All three structures in this section** *are unfitted. No simulation using invented or assumed parameter values has been run, and none should be, prior to the data described in Section 6.1 (RQ-D1, RQ-D2) becoming available. A fitted model built on placeholder numbers would misrepresent the state of the evidence and would be a weaker contribution than no model at all for a review process focused on rigor. The immediate next step for this section is not further modeling but the empirical work in Section 7.*

6. Research Questions

Every research question below is stated as currently open. None has yet been answered by data collected for this proposal. They are grouped into four families.

6.1 Family D — Drift and Control

ID	Question
D1a–e	For each of the five mechanisms in Section 3.3 individually, does it compound measurably across successive iterations of agent-driven change?
D2	Across the four-level control-regime spectrum proposed in Section 7.2, what is the observed rate of drift accumulation at each level?
D3	Can the drift process be formally modeled in a way that is calibrated to real data and produces predictions that can subsequently be tested?
D4	Does a control-type verification gate outperform a warning-type gate specifically on the terms Monperrus (2026) uses to argue for reduced review — and if so, by how much, and under what conditions does the advantage disappear?

NOTE ON SCOPE — *RQ-D4 is identified in Section 7.4 as the single highest-priority study in this family: it is the most directly citable against a specific, current claim in the literature, and the most likely to be publishable as a standalone result even before the full multi-cycle program is complete.*

6.2 Family A — Agent Instruction-Adherence

ID	Question
A1	Does reinforcing a rule to an agent measurably increase, rather than decrease, the rate at which that rule is violated?
A2	Does a measurable tendency toward sycophancy in a given agent or model correlate with a higher rate of silent failure at the Survey stage of the Sketch/Blueprint ceremony?
A3	Is a model's trained-in truthfulness or instruction-following a substitute for external, deterministic verification, or a complement to it — including the possibility that external verification becomes redundant as training improves?
A4	Does an agent's accuracy at flagging risk differ measurably between well-documented risk classes and genuinely novel ones not well represented in training data?

6.3 Family S — Statistical Control (Safety-Engineering Lens)

ID	Question
S1	Does the current defect rate of lightly-reviewed AI-generated code meet the bar that other safety-critical engineering disciplines require before removing a human check?

ID	Question
S2	Does Monperrus's own proposed alternative to human review (a "structured agent sign-off" process) achieve genuine statistical control, or does it substitute one uncontrolled process for another?

6.4 Family M — Direct Engagement with Monperrus (2026)

ID	Question
M1	Does Monperrus's proposed "specifier/orchestrator" division of labor functionally converge with T2D2's ceremony-gated oversight model, despite differing terminology?
M2	Can the cost-benefit "flip" that Monperrus argues has already occurred be modeled as a function of codebase size and iteration count, such that it holds in some regimes and not others?
M3	Is there a real, defensible taxonomy of defect classes that are structurally unreachable by agent self-review — and if so, what is it?
M4	Do the specific studies Monperrus cites for the claim that agents now match trained human reviewers actually test out-of-distribution defect types, or primarily defect types well represented in training data?

⚠️ **NEEDS ADDITIONAL SUPPORT — Sixteen research questions are listed above** *across four families, and none currently has a completed study behind it. Section 7 proposes a method and a prioritization for answering a subset of these; this proposal does not assume all sixteen will or should be pursued before initial results are shared.*

7. Proposed Methodology

7.1 Why a Multi-Cycle Design Is Required

RQ-D1 and RQ-D2 (Section 6.1) are claims about trajectories over successive iterations, not about a single point-in-time comparison. A study design that compares one "before" state to one "after" state cannot test a compounding-drift hypothesis; the design must apply repeated rounds of agent-driven change and track how a drift-severity proxy evolves across those rounds. The minimum feasible version of this design applies N successive rounds of agent-driven change to a codebase — synthetic or real — under each of several control regimes, and measures how the chosen drift-severity proxy evolves within each regime across rounds.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — A specific, validated drift-severity proxy metric** *has not yet been chosen or validated. Candidate proxies (for example, structural-diff magnitude against a reference graph, count of newly introduced parallel/redundant structures, or defect-injection recovery rate) each have known measurement limitations that would need to be addressed before a study could be defended. This is one of the most consequential open items in this proposal, since every downstream research question in Family D depends on it.*

7.2 The Control-Regime Spectrum

Four levels of verification control are proposed as the independent variable across which drift accumulation would be compared:

Level	Regime	Notes
Level 0	**Prompt-and-rules only**	*No structural gate — baseline condition.*
Level 1	**Harness + basic automated testing**	*Standard continuous-integration testing; no architectural check.*
Level 2	**Post-hoc human or agent-judgment review**	*A warning-type gate, per the Section 3.6 framing.*
Level 3	**T2D2's pre-code structural gate**	*A control-type gate (Fidelity Check), per the Section 3.6 framing.*

7.3 Data Sources and Partner Requirements

Two data sources are anticipated: synthetic or open-source codebases, on which iterative agent-driven change can be applied under controlled conditions without confidentiality constraints, and real, in-production codebases, made available through a research partnership, which would allow the studies to be grounded in realistic scale and complexity.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — No data-sharing agreement or research partner is currently in place.** *A prospectus describing a possible partnership exists as a separate document, but the specific data-sharing scope described there (anonymized or aggregated codebase history) is intentionally general and would need to be substantially narrowed once a partner's own legal and confidentiality constraints are known before any real-codebase study could proceed.*

⚠️ **NEEDS ADDITIONAL SUPPORT — Sample size, codebase selection criteria, and a pre-specified statistical analysis plan** *have not yet been defined for any of the four studies implied by Sections 6.1 and 7.2. These would need to be specified and, ideally, pre-registered before data collection begins, to support the rigor commitment described in Section 7.4.*

7.4 Prioritization, Sequencing, and Rigor Commitments

Given limited resources, the studies described above are prioritized as follows. A study of Fidelity Check's precision and recall against real, already-classified code changes is proposed first, since it is the cheapest to run and directly informs whether the larger, multi-cycle studies are worth the additional investment they require. RQ-D4 — the direct, head-to-head comparison against Monperrus's own terms — is identified as the single highest-priority study if only one can be completed before an initial set of results is shared externally, since it is the most citable and the most novel engagement with a live, current debate.

Three rigor commitments are proposed for all studies in this program: independent evaluation, meaning a partner's own blind or independently-designed trials rather than evaluation solely by the author; pre-registration of hypotheses and metrics before data collection begins; and publication of a replication package. These commitments are proposed specifically because the author is the creator of the system under evaluation and is not a neutral party — see Section 8.4.

8. Threats to Validity, Scope Limits, and Conflict of Interest

8.1 Construct Validity

The central unresolved construct-validity question is the choice of drift-severity proxy discussed in Section 7.1. Until a specific proxy is chosen and validated against an independent measure of "does this codebase actually work worse or become harder to maintain," any measured effect risks reflecting an artifact of the chosen metric rather than the underlying phenomenon this proposal is trying to characterize.

8.2 Internal Validity

A multi-cycle design introduces the risk that observed differences between control regimes reflect differences in the underlying task difficulty assigned to each regime, rather than the regime itself, unless task assignment is randomized or otherwise controlled. This has not yet been addressed in a concrete experimental protocol.

8.3 External Validity and Generalizability

Results obtained on synthetic or open-source codebases may not generalize to the real, in-production codebases this proposal ultimately cares about, given differences in scale, organizational context, and the presence of institutional knowledge not captured in the code itself. Results may also be specific to the particular agent models and tooling available at the time of study, given how quickly the underlying model capabilities are changing.

8.4 Conflict of Interest

The author of this proposal is the creator of the system (T2D2 / Poka-yoke) that the proposed research is intended to evaluate. This is a direct conflict of interest with respect to any claim that the system performs well. The rigor commitments in Section 7.4 — independent evaluation, pre-registration, and a published replication package — are proposed specifically to mitigate this conflict, and should be treated as a required condition for any of the results in this program to be credible to an external audience, not as an optional enhancement.

8.5 Scope Limits

The safety-critical-engineering framing used in Section 6.3 (Family S) is explicitly stronger for software where a defect carries meaningful harm potential, and explicitly weaker for low-stakes code — for example, a cosmetic bug in a marketing website's styling. This proposal does not claim the safety-engineering argument applies uniformly across all software, and any presentation of Family S findings should preserve this distinction rather than generalize beyond it.

8.6 Evidence Graded by Strength

To keep the varying reliability of the evidence used throughout this proposal visible in one place, it is graded here into four tiers:

- Strong, independently verified: a reported dataset of 20,574 real agent sessions showing that constraint violations grow as a share of total failures even as raw failure rates decline; two independent benchmark methodologies (AgentIF and IFScale) converging on the same finding of instruction-following degradation under complex or dense constraints.
- Real industrial corroboration, though not this proposal's own data: Meta's ARCTIC system (arXiv:2607.29516) as independent confirmation that a major organization treats this class of problem as worth solving at industrial scale.
- Live, genuinely contested debate, not a settled matter: the symbol-grounding problem and related "epistemic parasitism" arguments; reported cases of models learning to game tests so they appear to pass (sycophancy-to-subterfuge); the mixed empirical record on Lehman's Laws described in Section 5.1.
- Explicitly anecdotal or unaudited: three firsthand observations informing the mechanism list in Section 3.3 (a bracket-matching error missed by an agent that a linter would have caught instantly; degrading find-and-replace reliability as file size grows; an agent avoiding a good pattern because it "looked too similar" to something already present); and Webaroo's public claim of zero human developers, cited in Section 1.2 as illustrative of industry rhetoric, not as verified fact.

> ⚠️ **NEEDS ADDITIONAL SUPPORT — The evidence tiers above** *should be treated as a live document, not a final accounting. As Family D, A, S, and M studies (Section 6) produce results, items should move between tiers, and this section should be revised accordingly rather than left as a static snapshot of the evidence available at the time of this draft.*

9. Near-Term Steps

Rather than a fixed calendar timeline — which would depend heavily on partner availability and data access not yet secured — the following sequence of near-term steps is proposed, in priority order:

- Define and validate a drift-severity proxy metric (closes the gap identified in Section 7.1). This blocks every other study in Family D and should be resolved first.
- Complete the Fidelity Check precision/recall study against real, already-classified code changes (Section 7.4). This is the cheapest study to run and directly informs whether the larger multi-cycle studies are worth pursuing.
- Independently verify the agent-tamper-resistance design requirement described in Section 4.4, ideally through an adversarial evaluation.
- Pursue RQ-D4 (Section 6.1) as the highest-priority head-to-head study, given its direct engagement with a current, named position in the literature.
- Prototype a spec-to-graph translation path from a spec-driven orchestration frontend (Intent or Kiro) into a T2D2 graph, and assess its fidelity against a graph produced through the standard Blueprint ceremony, per Section 4.5.
- Identify and formalize a data-sharing agreement with at least one research partner willing to provide access to real, in-production codebases under the constraints discussed in Section 7.3.

> **NOTE ON SCOPE** — *Candidate venues for eventual publication of results include software-engineering research and practice tracks (for example, ICSE-SEIP, ICSME, IEEE Software) as well as earlier-stage vision or new-ideas tracks, which may be a more appropriate first outlet given how much of this proposal currently rests on unverified hypotheses.*

10. Expected Contributions

Pending the evidence described throughout this proposal, the intended contributions of this research program are:

- A three-artifact model (structural graph, behavioral test manifest, and runtime execution trace) unified by a shared identifier scheme, extending structural verification to a pre-code stage and forward into production observability within a single feedback loop.
- A named, falsifiable position that engages directly with a specific, current claim in the literature (Monperrus, 2026), rather than a general argument about industry practice.
- A control-type / warning-type theoretical framing, adapted from Shingo's poka-yoke distinction, yielding a specific, testable prediction about the shape of failure under different verification regimes.
- A scaffolded, multi-cycle empirical research program with an explicit prioritization, rather than a single-point before/after comparison.

As stated in Section 1.4, none of the four contributions above should currently be read as an established result. They describe the intended shape of the contribution this research program is designed to produce, contingent on the open items catalogued in Section 11.

11. Summary of Items Requiring Additional Support

The table below consolidates every flagged gap raised in the body of this proposal, in the order it first appears, together with the section where it is discussed and a brief statement of what would close it. This section is intended to function as a working checklist, not as a static appendix — items should be removed or updated as evidence is gathered.

§	Item	Current Gap	What Would Close It
1.4	**Preview contributions**	*All four intended contributions are unverified; no study has been completed.*	Complete the studies in Section 6/7; revisit Section 10 once results exist.
2.3 / 4.5	**Spec-driven orchestration integration (Intent, Kiro)**	*A complementary integration model is proposed — their specs as graph-generation input — but no spec-to-graph translation mechanism has been built or tested.*	Prototype spec-to-graph translation against real Intent or Kiro output; assess fidelity against a graph from the standard Blueprint ceremony.
3.2	**Lead thesis and corollary**	*Both are hypotheses; no empirical study has run.*	RQ-D1–D4 (Section 6.1).
3.3	**Five drift mechanisms**	*None measured individually; assumed distinct but untested.*	RQ-D1a–e, once a drift-severity proxy exists (Section 7.1).
3.4	**Amplification hypothesis: absence of durable stake**	*Speculative; no supporting or opposing evidence identified.*	Targeted literature search or a dedicated study design.
3.5	**Instruction-adherence findings applied to coding agents**	*Evidence is from general-purpose language-model settings, not coding-specific.*	RQ-A1–A4 (Section 6.2).
3.6	**Fidelity Check as a genuine control-type mechanism**	*Depends on a tamper-resistance design requirement not yet independently verified.*	Adversarial evaluation described in Section 4.4 and Section 9.
4.1	**Single-graph, three-role design vs. alternatives**	*Not evaluated against designs using separate artifacts per role.*	A comparative design study, not yet scoped.
4.3	**Overall system maturity**	*Alpha-stage; several ceremonies unexercised on real production systems.*	Real-world pilot once a research partner is identified (Section 7.3).
4.4	**Agent-tamper-resistance of Fidelity Check**	*Not independently verified.*	Adversarial evaluation (Section 9, item 3).
5.3	**All three formal-model structures**	*Unfitted; no real data used.*	Fit only after RQ-D1/D2 data exists (Section 6.1).
7.1	**Drift-severity proxy metric**	*Not yet chosen or validated; blocks all of Family D.*	Define and validate before any Family D study proceeds.
7.3	**Research partnership**	*No agreement in place; current scope*	Formalize a partner agreement with a

§	Item	Current Gap	What Would Close It
	and data-sharing scope	*description is intentionally general.*	specific, narrowed data-sharing scope.
7.3	**Sample size, selection criteria, analysis plan**	*Not defined for any proposed study.*	Specify and pre-register before data collection begins.
8.1	**Construct validity of the drift proxy**	*Risk that a chosen metric is an artifact rather than a true measure of drift.*	Validate proxy against an independent outcome measure.
8.2	**Internal validity of the multi-cycle design**	*Task-difficulty confounds across regimes not yet controlled for.*	Randomize or otherwise control task assignment in the protocol.
8.3	**External validity beyond synthetic/open-source code**	*Real production codebases may behave differently at scale.*	Replicate key studies on real codebases once partnered (Section 7.3).

References

NOTE ON SCOPE — *The entries below reflect the level of bibliographic detail available in the working source materials for this proposal. Several lack full venue, page, or publisher information. Full verification and formatting to a consistent citation style (e.g., ACM or IEEE) is a remaining task before this proposal is submitted anywhere formal — see the final row of Section 11 in spirit, though it is not included as a numbered item there since it affects the whole document rather than one section.*

Karpathy, A. (2025). Remarks introducing "vibe coding," February 2025 (widely reported; named Collins Dictionary Word of the Year, 2025).

Monperrus, M. (2026). The End of Code Review: Coding Agents Supersede Human Inspection. arXiv:2606.13175.

Maddila, C., Rashik, M., Khan, S., Jha, R., Saindon, C., Nagappan, N., & Rigby, P. (2026). From Code Review to Code Critique: Intent, Drift, and Spotlight for AI-Generated Diffs at Scale (ARCTIC). Meta. arXiv:2607.29516.

Yamaguchi, F., Golde, N., Arp, D., & Rieck, K. (2014). Modeling and Discovering Vulnerabilities with Code Property Graphs. IEEE Symposium on Security and Privacy. (Joern / CPG; IEEE Test-of-Time award, 2024.)

Spec Kit Agents (2026). arXiv:2604.05278.

De-Hallucinator (2024). arXiv:2401.01701.

Tang, X., Chen, Y., et al. (2026). How Coding Agents Fail Their Users (working title). arXiv:2605.29442.

Qi, C., et al. (2025). AgentIF. arXiv:2505.16944. NeurIPS 2025 Datasets & Benchmarks (Spotlight).

Jaroslawicz, D. (2025). IFScale. arXiv:2507.11538.

Yu, L., et al. (2013). An Empirical Study of Lehman's Law on Software Quality Evolution (Tomcat, Ant).

Olbrich, S., Cruzes, D., Basili, V., & Zazworka, N. (2009/2010). Empirical study on code-smell evolution (Lucene, Xerces).

Deming, W. E. (1986). Out of the Crisis. MIT Press.

Shingo, S. (1986). Zero Quality Control: Source Inspection and the Poka-yoke System. Productivity Press.

Poppendieck, M., & Poppendieck, T. (2003). Lean Software Development: An Agile Toolkit.

Senge, P. (1990). The Fifth Discipline: The Art and Practice of the Learning Organization. (Reinforcing/balancing loop archetypes.)

Harnad, S. (1990). The Symbol Grounding Problem. Physica D: Nonlinear Phenomena, 42(1–3).

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the Dangers of Stochastic Parrots. FAccT 2021.

Mitchell, M., & Krakauer, D. (2023). The Debate Over Understanding in AI's Large Language Models. PNAS.

Shanahan, M. (2024). Talking About Large Language Models. Communications of the ACM.

"Epistemic parasitism" (2026) — working title of a source discussing disquotation and re-quotational risk in language-model outputs; full citation to be completed.

Anthropic (2025–2026). Research on sycophancy and reward hacking leading to subterfuge in evaluated model behavior; full citation to be completed.

Reporting on Cursor usage, valuation (~$30B), and revenue (~$1B+ ARR) (2025–2026), Business Insider and related trade press; full citations to be completed.

Webaroo. Public claim of 23 production features shipped in 60 days with zero human developers (2026); primary source to be located and cited directly, or the claim removed if it cannot be sourced.