What AI Gets Wrong When You Iterate

A proven failure mechanism in AI-assisted work — and what designers, startup leaders, and enterprise executives can do about it

Gregory Tomlinson
Probabilistic Systems Engineering
ai.gtzilla.com
March 2026

This document summarizes findings from a published body of experimental research on iterative AI-assisted work. The full research corpus, including papers, replication materials, and operational contracts, is available at ai.gtzilla.com.

The Core Finding

AI tools are remarkably good at making the change you ask for. They are structurally bad at propagating that change to every place it needs to go.

This is not just a performance limitation. It is not mainly a matter of choosing a better model or writing a better prompt. The research points to a repeatable mechanism: when a rule applies in multiple places but the instruction names only one of those places, the AI updates the named place and leaves the others untouched.

The rule is implemented where it is mentioned. It is silently absent everywhere else.

The shortest version is this:

AI binds to the scope you name.

Everything you do not name tends to remain unchanged, even when the same rule should apply there too.

In the underlying experiments, the system under test had two file-handling paths: a deletion path and an overwrite path. A new requirement — preserve manually created files — applied to both. In every code-only run that reached the cross-surface stage, the deletion path was updated and the overwrite path was not. Manual files were protected during deletion and silently destroyed during overwrite.

The fix was not “prompt better.” The fix was explicit scope. When the full set of affected surfaces was written down in a durable artifact — what the research calls a contract — the failure disappeared. When that artifact did not exist, the failure reproduced with striking consistency.

The Mechanism

To understand why this happens, it helps to distinguish between two kinds of changes.

Same-surface changes affect one thing. “Make the button blue.” “Fix the deletion logic.” These are comparatively reliable under iteration because the scope of the request and the scope of the required change are the same.

Cross-surface changes affect multiple things. “Preserve manual files” applies to deletion, overwrite, and directory reset. “Update the brand color” applies to the product, the marketing site, the email templates, and the pitch deck. “Enforce this compliance rule” applies to every workflow that touches the regulated data.

The research shows a clear stability gradient:

Change Type	Scope	AI Stability
Same-surface, behavioral	One place, one behavior	Stable
Same-surface, architectural	One place, structural change	Prompt-sensitive
Cross-surface invariant	Multiple places, one rule	Fails without enumeration

The critical point is that this is not primarily a model-quality problem. A prompt-gradient experiment showed that the failure disappears only when the prompt explicitly lists every affected surface — at which point the prompt has become structurally equivalent to a contract.

The immediate mechanism is surface-scope binding: AI changes what you mention, and only what you mention.

Why This Is Not Obvious

This failure is easy to miss for three reasons.

First, the changed surface usually works correctly. Every test of the thing you asked about passes.

Second, the unchanged surfaces still function. They keep doing what they were doing before. Nothing visibly explodes. They are simply now wrong relative to the new rule.

Third, each individual iteration can look reasonable. The drift accumulates across steps rather than announcing itself in a single dramatic failure.

This is what the research describes as **authority drift**: reasonable-seeming change without explicit consent, compounding across iterations until the system no longer reflects what anyone actually decided it should do.

For Designers

Design systems, brand consistency, and multi-surface deliverables

If you use AI to iterate on design work — layouts, component systems, brand deliverables, or multi-format assets — you are very likely already experiencing this failure mechanism. You may not have named it that way, but you have probably seen the pattern.

Design work is inherently multi-surface. A brand identity is not one artifact. It is a set of rules that must hold across the product, the marketing site, the pitch deck, the social assets, and the documentation. A component library is not one component. It is a set of invariants — spacing, color, type hierarchy, interaction patterns — that must hold across every instance.

When you ask an AI tool to update the primary color, it updates the surface you are currently working on. The email template, onboarding flow, settings page, and error states can easily retain the old color. Not because the model “made a mistake” in the usual sense, but because those other surfaces were never named.

Scenarios from design practice

Design system iteration

You ask AI to update a card component’s border radius from 8px to 12px and adjust the shadow. It does that. The modal, dropdown, tooltip, and notification toast all keep the old radius. They are all card-like surfaces. None were named.

Result: visual inconsistency ships. Nobody notices until a design review catches it — or a user does.

Brand refresh across deliverables

You use AI to update a pitch deck with a new brand palette. The deck looks correct. The website hero, product UI, email headers, and social templates all retain the old palette. Each is a separate surface, and the prompt named only one of them.

Result: the deck looks updated while the rest of the customer-facing world does not.

Responsive behavior

You ask AI to fix the navigation layout for mobile. It does. The tablet breakpoint, landscape orientation, and accessibility zoom states are not mentioned, so they keep the old behavior.

Result: QA passes on mobile and fails on iPad. The fix was correct but incomplete.

What to do about it

The practical takeaway is specific: before iterating with AI, enumerate the surfaces a rule applies to. Not mentally. Explicitly — in the prompt or, better, in a persistent reference document the AI can see.

For design work, that means maintaining what is effectively a design contract: a living document that lists every surface where a given rule must hold. It does not need to be formal. It does need to be explicit.

Practical steps

- Build a **surface inventory**. For any rule that spans more than one component or deliverable, write down every place it applies.
- Keep a **reference document in context**. Include or attach that surface inventory during AI-assisted iteration.
- Make **cross-surface review** a deliberate practice. After any AI-assisted change, check the surfaces you did not mention.
- **Version the contract, not just the designs.** As the system evolves, the inventory of governed surfaces has to evolve with it.

The design-specific gap is not that design systems lack invariants. They already have them. The gap is that those invariants are often implicit: buried in token names, scattered through design files, or held in the designer's head. Making them explicit and visible during iteration is what turns silent drift into consistent propagation.

For Startup and Growth-Stage Executives

Speed is not the risk. Invisible drift is.

AI-assisted velocity is real. A small team can now ship at a pace that previously required much more headcount. This document is not an argument against speed. It is an argument that speed changes what kind of failure becomes likely.

The risk is not simply "the AI gets something wrong." The risk is that each change looks correct locally while cross-surface rules silently break elsewhere. The faster a team iterates, the more opportunities exist for that invisible drift to compound.

The underlying experiments reproduced this failure across different models, sessions, and starting conditions. The pattern is structural, not random. It arises from how AI scopes work, not from occasional bad luck.

Where this hits startup operations

Pricing logic across surfaces

A pricing rule applies at checkout, in the billing API, in the invoice generator, and in the dashboard display. An AI-assisted change to checkout logic does not automatically propagate to the other three surfaces unless they are explicitly named.

Result: customers see one price at checkout and another on their invoice. Support load rises. Trust erodes. The checkout implementation itself may still be correct.

Permission model changes

You tighten a permission rule so a feature is admin-only. AI updates the UI gate and the primary API endpoint. The export endpoint, webhook handler, and bulk operation path are not named, so they retain the old permission model.

Result: the security boundary works on the happy path and still has bypasses elsewhere.

Compliance requirements

You add a retention rule to satisfy a customer contract or regulatory requirement. The primary data store gets updated. The cache layer, analytics pipeline, log aggregator, and backup system do not.

Result: you believe you are compliant in general when you are only compliant in one surface.

Operational implications

The research does not suggest that startups should slow down. It suggests that the cost of maintaining explicit scope is far lower than the cost of discovering drift after it has compounded.

For a small, fast-moving team, that translates into a few concrete practices:

- Identify the **cross-surface invariants** that actually matter: pricing, permissions, data handling, compliance, branding.
- Write them down with **explicit surface enumeration**.
- Make that reference visible during AI-assisted iteration.

- Treat **cross-surface review** as a first-class step rather than a cleanup afterthought.

The startup-specific takeaway is simple: AI-assisted velocity is a real advantage, but only if the rules that must propagate are written down where the system — and the people using it — can see them. That is not bureaucracy. It is operational infrastructure for moving fast without accumulating silent drift.

For Enterprise and Public-Company Executives

The governance gap AI just made visible

Most large organizations already have governance structures: review processes, compliance frameworks, change management procedures. The question this research raises is not whether those structures exist. It is whether they operate at the point where decisions actually execute.

The underlying research points to a precise mechanism: when iteration speeds up and authority is not explicitly encoded at execution time, outcomes drift from intent in predictable ways.

The distinction here is between **understanding** and **authority**. Understanding explains what a system does. Authority constrains what it is permitted to do. Organizations often invest heavily in understanding and assume authority follows from that. The experiments show that it does not.

AI did not create this gap. It made it easier to reproduce by collapsing iteration time. Safeguards that once existed partly because of slowness — review latency, human memory, coordination friction — no longer provide the same control. What once felt like governance may in many cases have been a side effect of reduced speed.

The authority problem in enterprise terms

In a regulatory or governance context, the key question is straightforward:

Can you demonstrate that what your systems do is what you decided they should do?

The research suggests that under AI-assisted iteration, that answer degrades unless operational intent is encoded in a persistent, versioned artifact. Conversational prompts do not accumulate into an auditable record of intent. They are ephemeral instructions that bind to the surfaces they name and leave the rest unchanged.

For organizations operating under SOX, GDPR, HIPAA, SOC 2, or similar control regimes, this is a structural exposure. If regulated changes are implemented through AI-assisted iteration without explicit invariant enumeration, the resulting system state may not match documented intent. In some cases, it may not even be possible to reconstruct what intent actually was.

Enterprise scenarios

Regulatory compliance change

A new data-handling requirement must apply across ingestion, processing, storage, access control, export, and retention. AI updates the primary data store and the API layer. The ETL pipeline, reporting database, partner data feed, and disaster-recovery system are not named.

Result: the organization believes it is compliant after updating only part of the governed surface.

Policy enforcement across business units

A board-level policy decision must propagate across multiple business units. Each unit uses AI to implement the change and names only the surfaces it already knows about. Shared services, common infrastructure, and vendor integrations go unnamed.

Result: every unit can pass its own local review while the organization still fails the cross-unit audit.

What boards and risk committees should ask

As AI-assisted iteration becomes more common, governance questions have to move closer to execution. Useful questions include:

- Where do our **cross-surface invariants** live?
- Can we demonstrate that actual system behavior matches documented intent?
- Do our change-management processes account for **propagation across surfaces**, not just the named change?
- Are our AI-assisted workflows producing **auditable authority artifacts**, or only ephemeral prompts?

The enterprise-specific takeaway is that AI-assisted iteration does not remove the need for governance. It changes where governance must operate. Authority has to be encoded at execution time in a persistent, versioned artifact, not reconstructed afterward from conversation logs.

Organizations that treat specification as a maintained artifact gain both speed and auditability. Organizations that do not may gain speed while losing demonstrable control.

The Research Behind This

This document summarizes findings from a published research corpus. The claims here are not presented as loose theory. They are grounded in controlled experiments with documented methodology, replication materials, and source artifacts.

The broader finding is not simply about code. Code was the experimental medium because it allowed controlled measurement. The mechanism itself — surface-scope binding — generalizes as a way of thinking about iteration wherever a rule must hold across multiple surfaces. That is why the bridge into design, operations, and governance is legitimate, even though the measured experiments were in software implementation.

Recommended reading path

Start here: Authority, Execution, and Refusal
Ten linked essays diagnosing where control actually lives in systems that execute.

The proof: Contract-Centered Iterative Stability
The experimental paper documenting the stability boundary, the failure mechanism, and the cross-run results.

The method: Contract-Centered Engineering
The practical loop for specification, derivation, evaluation, and refinement.

The mechanism: Breaking the Loop
A shorter treatment focused on why iterative AI edits produce structural drift.

Replication materials
Methodology, comparison summaries, failure-class documentation, and operational contracts used in the experiments.

The practical minimum is simple:

If a rule applies in more than one place, name every place.

The stronger structural solution is a versioned contract that preserves that scope across iteration.