



GovernedContext
Architecture



Authority Beyond Expertise

Governing AI-Assisted Software Construction When Capability Exceeds Personal
Technical Knowledge

By Jamaal Saalik Belt-Daniels

Contents

Abstract

1. The new asymmetry
2. Capability is not authority
3. The hidden failure is not always a wrong answer
4. When authority becomes real
5. Making unknowns governable
6. What the construction record shows
7. What the evidence does not show
8. Where this fits with existing work
9. Limits
10. A path for nontraditional builders
11. Conclusion

Evidence notes

References

Abstract

AI can give one person access to technical capabilities that exceed the boundaries of that person's own expertise. In software construction, that is both an opportunity and a governance problem. A builder may be able to obtain architecture, code, security analysis, interface design, tests, and specialist explanations without personally being an expert in every discipline involved. But the same capability that allows the work to continue can also allow consequential decisions to be made where the human is least equipped to notice that a decision has occurred.

This paper asks whether a person whose expertise does not span every discipline required by a complex software system can use AI to build that system while retaining meaningful authority over what gets built. It argues that the answer can be yes—not because AI eliminates expertise, but because capability, expertise, execution, and authority are different things. Technical work can be distributed across AI systems, tools, tests, research, audits, and specialists while higher-order authority remains with the human owner, provided that material decisions are surfaced, delegation is bounded, consequential progression is actually gated, and durable records preserve what was decided and what changed.

This argument is grounded in my own practice. I developed and used the governance method discussed here while building software systems involving multiple technical disciplines and construction protocols with AI. That record supports the viability of the arrangement as I have practiced it. It does not establish population-level effectiveness, comparative superiority, professional engineering maturity, or a universal threshold for specialist intervention.

1. The new asymmetry

Software systems cross many disciplines. Product definition, user experience, frontend and backend engineering, data storage, authentication, authorization, deployment, testing, security, recovery, and observability can all matter in one application. Conventional teams distribute those concerns across people with different expertise.

AI changes who can access that distributed capability. A person who could not independently design every part of a system can now ask a capable model to explain technical choices, propose architectures, generate code, construct tests, challenge assumptions, and help execute work. Research on professional software engineers already describes AI-assisted work shifting toward verification, direction, evaluation, and correction rather than only direct code creation. [7]

That creates a question more important than whether AI can generate software:

Can a person who lacks deep technical expertise across every discipline required by a complex software system use AI to build that system while still retaining meaningful authority over what gets built?

The subject here is not someone with zero technical competence. Expertise is uneven and changes over time. The relevant condition is that the builder's personal expertise does not cover every discipline implicated by the system.

The danger is not ignorance alone. The danger is that a capable model can continue through an expertise gap without forcing that gap to become visible. It can make a choice that appears to be implementation detail but actually determines architecture, security policy, data behavior, user experience, or accepted system state.

A weak system that cannot proceed creates an obvious interruption. A stronger system may produce a coherent answer and keep moving. That can make the governance problem less visible. The model may be technically competent and still execute a consequential decision the human never meaningfully saw, challenged, or authorized.

The objective, then, is not to make the human omniscient.

It is to prevent what they do not know from silently becoming authority over the system they are trying to build.

2. Capability is not authority

Four ideas need to be separated.

Capability is the ability to perform a task. A model may be capable of writing code, proposing a database schema, explaining a security mechanism, or generating a test harness.

Expertise is the knowledge and judgment possessed by an actor. A human may be strong in one domain, competent in another, and dependent on outside help in a third.

Execution is the performance of the work. Models, tools, tests, automated systems, and specialists can all execute work.

Authority is different. It concerns who determines what the system is, which material tradeoffs are accepted, when consequential progression is allowed, and what becomes accepted project state.

Organizations commonly separate expertise from authority. An owner can approve a product without writing its code. A technical lead can rely on a specialist's analysis without independently reproducing it. Delegation does not automatically transfer the right to redefine the system.

AI compresses that delegation relationship. The same model can move from explanation to recommendation to implementation almost instantly. That makes it easier for assistance to become silent decision-making unless the surrounding process preserves the distinction.

Meaningful authority therefore does not require the human to derive every technical answer from first principles. It is evaluated at the material decisions, gates, and state transitions that determine what the system becomes.

The depth of direct human control can also change. I have experienced that directly. As I learn more about a domain, I govern more of its technical details myself. Where my expertise is weaker, I lean more heavily on models, research, explanation, tests, and review. The granularity of my intervention changes; the higher-order authority relationship does not have to.

Responsibility is related but not identical. I remain a real-world actor who can decide whether to rely on the work and release the system. Delegating technical work to AI does not erase the consequences of that choice. The governance proposition in this paper is that consequential authority should remain coupled to that responsibility. That is not a universal claim about legal liability.

Research on meaningful human control offers useful adjacent thinking. It has argued that responsibility should be commensurate with a human's ability and authority to control a system and that operational boundaries and links between AI actions and responsible human actions matter. [6] This paper addresses a different object—technical construction rather than the behavior of an autonomous deployed system—but the overlap is important: human presence by itself is not meaningful control.

3. The hidden failure is not always a wrong answer

Most criticism of AI-assisted coding focuses on bad output: insecure code, fabricated APIs, broken logic, weak tests, or hallucinated facts. Those failures matter, but they are not the central problem here.

The more subtle failure is **competent execution of an unapproved decision**.

Suppose an AI-assisted build reaches authentication. The model knows that a sign-in experience is required and is fully capable of creating one. But imagine that the product or experience work never actually decided what sign-in should look like or how users should move through it. If the implementation model invents the experience because something has to exist, the result may work technically and still represent an authority failure. The implementation has become the accidental author of an upstream decision.

The same pattern can happen with storage, retry behavior, data retention, failure handling, security boundaries, or interface behavior. A technical choice can quietly determine something the owner never realized was being decided.

This is not unique to AI. Human implementers also fill requirement gaps. AI changes the scale and speed of the problem because a capable system can cross several expertise boundaries before the human notices that an unresolved choice existed.

The practical question becomes: **what may be delegated, and what must force the system to stop and surface a decision?**

The framework uses two kinds of materiality.

Structural materiality exists when a decision changes architecture or design, or forces an established upstream decision, phase, document, or authority boundary to be reopened.

Consequence materiality exists when the decision is important enough that silent delegation would be unacceptable even if no upstream artifact has to change. Security, customer data, money, durability, and consequential user experience are obvious examples.

This is not meant to create an approval step for every variable name or visual margin. The point is to separate ordinary implementation discretion from decisions capable of materially changing what the system is or what risks it carries.

4. When authority becomes real

An approval button does not prove that the human has authority. A process can ask for approval after the real decision has already been made. It can log a confirmation without allowing the person to stop anything. It can make the human a ceremonial participant.

In the governance arrangement I use, meaningful authority has two operational properties.

First, **the decision must have causal effect**. If a material decision requires owner authority, consequential progression must actually stop until that authority exists. Gates, phase closures, locks, and other material boundaries matter because the project cannot legitimately move beyond them simply because the AI is capable of continuing.

Second, **the decision must leave durable evidence**. The project has to preserve what was decided, under what authority, what state existed before the decision, what changed after it, and what remained unresolved.

This is what turns “we reviewed that” into something inspectable. A traceable system can show the chain of decision, evidence, correction, approval, and resulting state rather than relying on conversation memory or a later summary.

Together, those properties produce the paper's working definition:

Human authority is meaningful when consequential progression is actually prevented without the required authorized decision, and durable records preserve evidence that the decision was exercised and what state changed as a result.

That definition says nothing about whether the human chose correctly. Real authority includes the ability to make a bad decision. Governance is not a correctness oracle. Its purpose is to stop capability from silently acquiring authority merely because it can keep going.

5. Making unknowns governable

The hardest problem in building outside your expertise is often not answering questions. It is knowing which questions exist.

My first defense was interrogation. When a model recommended something technical, I kept asking why: Why this architecture? Why this security mechanism? What does this assumption mean? What breaks if we choose the alternative? Show me an example. Test it. Explain the tradeoff until I can understand the decision well enough to evaluate it.

That helps with known questions. It does not solve unknown unknowns. You cannot reliably ask about a class of failure you do not know exists.

That problem drove me toward more structured governance.

Bounded delegation

When I was challenged with the possibility that AI might still control the frame by choosing which alternatives, risks, and evidence I ever see, my answer was: **“We built the menu. We built the restaurant.”** The point is not that I personally supply every technical option. The point is that I establish the governing environment in which the model is allowed to reason, propose, execute, derive, and escalate.

The central rule is better stated this way:

Authority can remain human while expertise is distributed when the process distinguishes authoritative intent from derived technical consequence and governs the boundary between them.

That distinction matters because I do not secretly possess every technical requirement that later appears in the work. I can decide what the product is for, what outcome I am willing to accept, which user or business behaviors are material, what risks require explicit treatment, and which classes of decision I am willing to delegate. Those are authority-bearing inputs. They do not tell me, by themselves, every security property, recovery obligation, state transition, interface constraint, or test that competent implementation will require.

Technical expertise can legitimately derive obligations from those authoritative inputs and from applicable external constraints. If a system is authorized to hold account-bound user data, for example, security and backend reasoning may derive requirements around authentication, authorization, trust boundaries, storage, or recovery that I did not know to name beforehand. The fact that I did not originate those technical requirements does not make them arbitrary model preferences. Their legitimacy depends on whether the derivation can be traced to an authoritative requirement, a real external constraint, or an explicitly delegated engineering decision class.

That is also where the boundary becomes useful. A derived obligation is not permission to invent new product intent. If technical analysis reveals that two materially different product behaviors are both viable, or that an upstream experience decision was never made, the downstream layer does not get authority simply because it needs an answer. It can explain the consequence, narrow the question, propose alternatives, and route the unresolved decision back to the authority that owns it.

Inside those boundaries, AI can draft, investigate, audit, patch, implement, test, and perform routine procedural work. It can originate proposals and derive technical consequences. Neither act, by itself, creates authority to redefine the governing objective.

Focused reasoning environments

I also developed specialized protocols for different classes of work: backend construction, frontend construction, user-experience resolution, proof and testing, and protocol creation, among others.

These are not claimed to be complete checklists of everything that can go wrong. Their purpose is to focus the model on the professional concern-space relevant to the current work and the actual project at the same time. A backend-oriented process can force trust boundaries, state, interfaces, security, resilience, and unresolved decisions into view. A frontend-oriented process can prevent implementation from becoming the accidental author of unresolved engineering policy. An experience-oriented process can route missing user-facing decisions upstream instead of allowing an implementer to invent them.

The goal is not omniscience. It is to reduce dependence on the owner's memory of what an experienced practitioner would normally think to inspect.

The operating architecture underneath the questions

Seen one piece at a time, this can look like careful prompting plus good recordkeeping. That description misses the relationship among the pieces. In my implementation, the work is treated as a governed project object rather than as a conversation that happens to have a long memory.

The chain begins with authority-bearing inputs: purpose, accepted requirements, material owner decisions, approved experience, and other controlling constraints. Those are preserved with explicit state and provenance so that an old draft, a review, a newer-looking file, and an approved artifact do not become interchangeable merely because all of them are available to the model.

Specialized reasoning processes then work from that state. Their job is not only to answer technical questions, but to translate the authoritative inputs into the concern-space of the work being performed. A backend process may derive trust, state, interface, resilience, or security obligations. A frontend process may derive engineering obligations from an approved human experience. An experience process may discover that the product definition is insufficient to decide what a user should see or be allowed to do. The technical reasoning can be sophisticated without being allowed to rewrite the authority it consumes.

Execution happens downstream of those distinctions. Models and tools can produce code, specifications, tests, audits, patches, and other artifacts, but the artifacts carry state. A test result can support a technical claim without approving the artifact. An audit can discover a defect without authorizing its correction. A newer version can exist without becoming current. A completed internal artifact can remain unreleased. The system therefore tracks not only what exists, but what each object is allowed to mean for the next move.

When a downstream discovery exposes a missing material decision, the valid path can run backward. The affected branch can stop, return to the earliest authority that owns the unresolved question, resolve it there, and then propagate the result forward. That is re-entry. It prevents technical necessity from turning into silent authorization.

The whole mechanism can be summarized as a controlled loop: **authoritative intent** → **governed state** → **derived technical obligations** → **bounded execution** → **evidence and audit** → **acceptance or re-entry**. The loop does not guarantee a good answer. It makes the origin, authority, evidence, and consequences of the answer inspectable enough that capability does not have to become authority simply because it is technically impressive or convenient.

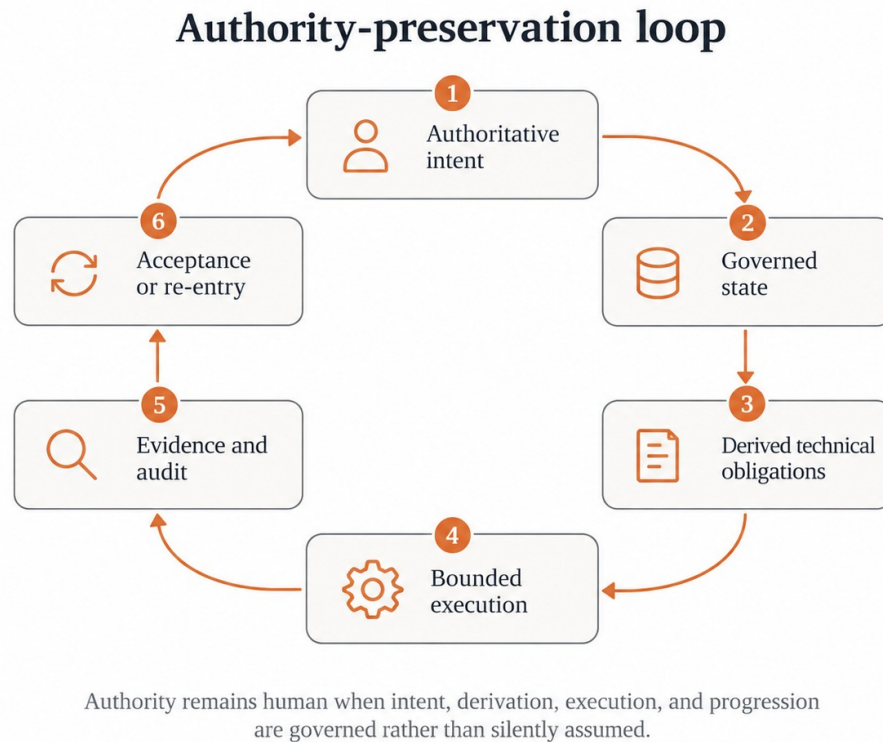


Figure 1. Authority-preservation loop.

Audit without authority capture

Audit introduces another problem. If the model that creates the work also reviews it, the review may share the same assumptions. But if the reviewer can directly rewrite the project, criticism can become another route for AI to acquire authority.

I separate **discovery authority** from **decision authority**. Audits are observer input. Findings must be classified before they create correction work. A valid issue can trigger a bounded patch, but the audit itself does not become permission to redesign, approve, lock, or release the system.

Using several model systems for review can add adversarial pressure, but this paper does not call those reviews epistemically independent merely because multiple models were used.

Re-entry instead of downstream invention

A visible failure is not always the location of the real problem. I have learned to treat implementation failures as possible evidence of a missing upstream decision. When that happens, I trace the dependency backward to the earliest authority that owns the missing decision, reopen that scope, resolve it, and propagate the result forward.

That matters because technical necessity should not masquerade as authorization. The fact that an implementation needs an answer does not entitle the implementation layer to decide a materially upstream question.

Evidence instead of “looks right”

Testing and proof serve the same general purpose. A passing test does not establish universal correctness, but it turns a bounded technical claim into something inspectable. My process repeatedly externalizes what would otherwise live only in my head: what must be checked, what requires authority, what evidence supports a claim, what state is current, and where a failure must return.

That is what **making the unknown governable** means in practice. Ignorance still exists. It is simply given fewer opportunities to change the system silently.

6. What the construction record shows

The evidence I offer in this paper is my own practitioner case, not a population study.

One case is a consumer application I am building with persistent user state, account boundaries, backend dependencies, in-use state changes, and security requirements. A bounded implementation revision addressed a machine-to-machine authentication boundary. The committed project record reports seven of seven workload-authentication tests passing, successful public/private key-pair verification, zero private-key material found in the canonical project tree and public carry directory, and rejection of prohibited or unknown scopes. [1]

Those results are useful precisely because the same record preserves what they **do not** mean. The revision did not establish broader production implementation, public exposure, deployment, or release. Live acceptance on the other side of the service boundary remained a carried condition.

A real multi-store shopping run in that same project exposed a different class of problem for me. Prices, quantities, weighted goods, coupons, store context, budgets, and checkout truth all changed during live use. The project responded by deriving a first-class execution state model and a separate offer/coupon state model, then repeatedly challenging, auditing, patching, and regression-testing those designs. At the carry-forward point, the pair was considered stability-ready from the bounded audit/regression perspective while still remaining noncanonical, not approved, not locked, and not production-authorized. [2]

A second case is a backend service I built. At one committed revision, its verification record reports 182 of 182 combined regression tests passing. A ten-minute selected-topology run recorded 1,200 of 1,200 requests accepted with zero HTTP errors. [3] The same verification checked scheduling and recovery behavior, including checkpoint creation, recovery-point age, restore time, database integrity, configuration-digest agreement, and restored object counts. [3]

Again, the limitations are part of the evidence. Source work in that topology test used a controlled delay stub, so the result does not establish real provider-network latency. The record also explicitly withheld deployment, public release, public exposure, and transaction authority. [3]

What the governance machinery itself did

The implementation results answer one question: did governed AI-assisted work reach technically substantive states? A different question is closer to the thesis of this paper: did the governance machinery actually preserve distinctions among authority, state, evidence, and valid continuation when the work became messy?

One consumer-application record gives a concrete continuity example. A governed transfer checkpoint packaged 772 project files with zero expected missing files and 772-of-772 agreement between the archive and its manifest. At a later recorded state, the project had an externally durable recovery object covering 802 files. More important than the counts, that later recovery record preserved a specific next eligible construction action while separately carrying an unresolved live-service acceptance condition and explicitly withholding deployment and release authority. The project state was therefore more than a pile of recovered files: it retained what controlled, what remained unresolved, and what was valid next. This does not prove that every transfer or runtime recovery will succeed. It shows that, in this recorded case, continuation was represented as governed state rather than mere conversational recall. [4]

The Store Run work provides a different test. Its final focused regression reported four of four required probes passing, with seventeen of seventeen supporting deterministic assertions. Yet the same carry-forward

record still labeled the resulting design pair noncanonical, not approved, not locked, not yet ingested into the main project, and without production-implementation or release authority. Technical success did not self-promote the artifact. The evidence could say that specified behavior had survived the tests; it could not decide what status the artifact was allowed to hold. That is a small but direct example of the separation between execution evidence and authority. [2]

A third example occurred in the construction of this paper itself. During publication production, the project control plane was later found in a contradictory and incomplete state: different control records disagreed about the current revision, required control records were missing, and the live container had lost material it previously expected to contain. The valid response was not to guess which state was probably right and keep packaging. Ordinary mutation stopped. A separately authorized repair reconstructed only the control/custody state that could be justified, explicitly preserved the locked manuscript and already accepted publication-package bytes, and did not use the repair as authority to change the paper. Only after that state was repaired could later observer criticism enter a separately governed post-lock re-entry decision. [5]

That example is not flattering evidence of infallibility; the custody failure happened. Its value is narrower. The failure did not silently become permission to improvise a new canonical state or mutate the locked paper. Uncertainty became explicit state, progression stopped where that uncertainty mattered, and reopening required its own authority. [5]

Taken together, these records expose the mechanism more directly than implementation test counts alone. The machinery can preserve a distinction between **what is technically supported**, **what is currently authoritative**, **what remains unresolved**, and **what action is valid next**. That is the behavior the paper's authority claim depends on. It remains practitioner evidence, not a controlled study of effectiveness, but it is evidence of the governance mechanism operating rather than merely a description of how I intended it to operate.

These examples do not prove professional engineering maturity. They show something narrower: I have been able to use AI inside a governed construction environment to reach technical states that can be inspected directly through regression results, source and authority boundaries, topology behavior, recovery evidence, and exact project-state records.

The cases support practicability. The governance arrangement has been used in substantial technical work. It can preserve distinctions among proposal, implementation, verification, acceptance, deployment, and release. And it can support correction and upstream re-entry when real use exposes a missing assumption.

7. What the evidence does not show

A functioning component is not proof of generalized effectiveness. A large passing test suite is not production deployment. Repeated use by one practitioner is not a population study. An audit PASS does not guarantee that no defect exists. A documented process is not superior merely because it is documented.

The current evidence is strongest on **practicability**. Several larger questions remain open.

Production and professional engineering scrutiny

A future test would be to give the resulting engineering to experienced practitioners and ask whether the material obligations appropriate to that class of software have actually been handled: security, isolation, data handling, recovery, correctness, operability, and other applicable concerns. The question is not whether the evaluator prefers the style or would have chosen the same architecture. It is whether the work is technically sound enough to pass professional scrutiny for what it claims to be.

That evaluation has not yet been completed as controlled evidence. Long-duration operation, real-user pressure, security review, maintenance history, and independent professional evaluation would materially strengthen or weaken claims about engineering maturity.

Generalization to other builders

My case demonstrates a path, not a prevalence rate. It does not establish how many people can follow it, how much training they need, or how much process friction they will tolerate.

It also matters that I learned while building. Repeated failures, explanations, audits, and construction loops became an unconventional technical education. I govern more technical detail directly now than I could when I started. That does not mean a later user of a mature governance system would need to recreate the same apprenticeship. Building the governance machinery and using a mature version of it are different competence problems.

Specialist escalation

My current practice has not established a universal point where AI-assisted investigation becomes insufficient and a human specialist becomes mandatory.

So far, when I hit technical walls, I respond with further decomposition, research, interrogation, aggressive testing, adversarial review, and review by distinct model systems. That is a persistence strategy, not evidence that specialists can be avoided indefinitely.

The specialist-escalation threshold remains an open empirical boundary.

Model capability

A separate research direction asks how much model capability can be traded for orchestration. Complex work might be decomposed into smaller bounded tasks that weaker models can perform, but decomposition itself can lose context and dependencies. I am exploring where that line lies.

That work is not evidence for this paper.

8. Where this fits with existing work

The argument is not that no neighboring concepts exist.

Meaningful-human-control research has already connected responsibility with ability and control authority and has emphasized explicit operating boundaries and links between AI actions and responsible human actions. [6] This paper shares the concern that nominal human involvement is insufficient, but it applies the problem to technical construction under expertise asymmetry.

Recent software-engineering research also describes a shift toward supervisory work. One longitudinal study of professional developers characterizes AI-assisted work in terms of direction, evaluation, correction, and verification. [7] Another small qualitative study describes layered supervision through preventive guardrails, executable guardrails, and human oversight. [8]

Those studies concern professional engineering settings. They do not establish this paper's claim about nontraditional builders. They do, however, support the broader observation that AI-assisted engineering increasingly involves governing and evaluating machine-produced work, not simply asking a model to write code.

Secure-development standards provide another useful boundary. NIST's Secure Software Development Framework treats software security as a set of practices integrated into development lifecycles. [9] This paper does not claim that my projects comply with the SSDF. The relevance is that software carries engineering obligations regardless of whether the code came from a traditional team, an AI-assisted builder, or some combination of the two.

Empirical productivity research adds a final caution. In one randomized study of experienced open-source developers using early-2025 AI tools, measured results contradicted participants' expectations about speedup. [10] That result should not be generalized to nontraditional builders or current frontier models. Its value here is simpler: perceived capability and perceived improvement are not substitutes for evidence.

9. Limits

When I was pressed on what this governance arrangement had actually solved beyond careful questioning, testing, auditing, and review, my answer was blunt:

“None. What do you mean? I don't have a little genie in a fucking box.”

That answer is important. I am not claiming a new source of correctness. This arrangement does not guarantee correct architecture, secure code, complete protocols, perfect audits, or good owner judgment. The constituent practices—testing, review, traceability, specialist input, change control, and disciplined questioning—already exist. What I am trying to do is make them explicit, repeatable, inspectable, and accessible to a builder operating beyond the edges of their own expertise.

A system can preserve a bad human decision perfectly. When I was asked who owns the failure if I choose to ship bad software after ignoring the warnings and evidence available to me, my answer was equally simple:

“It's my fault.”

If the owner ignores warnings, rejects evidence, or proceeds with a risk they do not understand, governance does not make the decision safe. Genuine authority includes the ability to make a poor choice and bear responsibility for it.

Protocols can also miss things. A document can be precise about what it claims and still omit something material. That is why audit must do more than check internal consistency; it must challenge omissions within the legitimate scope. At the same time, an auditor does not acquire authority to expand the system simply by identifying a possible improvement.

The current process is also not presented as a finished mass-market experience. Much of the machinery was created through manual, failure-heavy work and is only gradually being automated. A future interface may hide most governance metadata while preserving the controls underneath. That usability proposition remains to be demonstrated.

Production history is the largest current evidence gap. The construction record contains substantial implementation and verification evidence, but not the long-duration operational record needed to support a claim of professional engineering maturity.

10. A path for nontraditional builders

I think about this through an analogy to the self-taught practitioner in another technical trade: someone who gains practical capability through repeated repair, experimentation, failure, study, and sustained work rather than beginning from a traditional credential. The analogy is not evidence about other fields. It is a way to frame what AI may make possible in software.

AI can compress access to explanation, examples, technical execution, critique, and specialized assistance. That does not mean the user instantly possesses the expertise being supplied. It means the user may be able to command capabilities they do not personally possess.

That is the shift I experienced. The question stopped being “How do I obtain every capability myself?” and became “How do I use capability beyond my expertise without letting that capability decide what I am building?”

The answer I have developed is governance rather than omniscience.

Material decisions are surfaced. Delegation is bounded. Tests and evidence are required where claims matter. Audits can challenge without self-authorizing changes. Project state survives outside conversation memory. Downstream problems can return to the upstream decision that actually caused them. The human can deepen direct technical control as their expertise grows without pretending they already know everything.

Building that machinery from scratch is difficult. I do not want to pretend otherwise. Part of my reason for documenting the pattern is simple: I do not want the next builder to have to start from zero. On a hypothetical scale from zero to one hundred, I want them to be able to **“start from ten instead of zero.”**

And if the machinery can eventually be packaged well, the user may not need to see most of it.

The governance machinery does not have to be the user experience.

11. Conclusion

AI-assisted software construction creates an unusual asymmetry: technical capability can increasingly be accessed without the builder first possessing every expertise associated with that capability.

That is not only a productivity opportunity. It is an authority problem.

If AI can reason, recommend, implement, test, and revise across domains the human does not fully understand, “human in the loop” is too weak. The important question is whether the human remains in the causal path of the material decisions that determine what the system becomes.

The framework I use relies on enforceable barriers, durable traceability, bounded delegation, materiality rules, adversarial review, evidence, and re-entry to preserve that authority. Its key separation is between authoritative intent and the technical consequences that distributed expertise may legitimately derive from it. None of those mechanisms make the human omniscient or the AI infallible. They make it harder for capability to become authority without being noticed.

My current evidence supports the viability of that arrangement as I have practiced it. It does not yet establish population-level effectiveness, professional engineering maturity, a universal specialist-escalation threshold, or superiority over conventional engineering organizations.

The central proposition is therefore deliberately bounded:

A person need not personally possess every specialist competency required by a complex software system in order to retain meaningful authority over its construction. But when AI supplies capabilities beyond the builder's own expertise, that authority must be made operational through real control over material progression and durable evidence of the decisions that shape the system.

Or, more simply:

The objective is not to make a non-expert omniscient. It is to prevent what they do not know from silently becoming authority over the system they are trying to build.

Evidence notes

[1] Author project verification record, consumer-application machine-to-machine authentication boundary, 8 September 2026. Unpublished internal record. The exact source is preserved in the governed project source map; public project identifiers and security-sensitive implementation details are intentionally abstracted.

[2] Author project carry-forward and regression record, real-run-derived consumer-application execution state model, 7 September 2026. Unpublished internal record. The exact source is preserved in the governed project source map; the cited state was explicitly noncanonical and not production-authorized at that point.

[3] Author project post-commit verification record, backend-service implementation-readiness revision, 7 September 2026. Unpublished internal record. The exact source is preserved in the governed project source map; the cited topology used a controlled source-delay stub and expressly did not establish external deployment or public release.

[4] Author project snapshot-receipt and later recovery-anchor records, governed consumer-application transfer and continuation state, 8 September 2026. Unpublished internal records. The exact sources are preserved in the governed project source map. The records support the cited checkpoint/recovery counts and the preservation of next-valid-action, unresolved-condition, and no-release boundaries; they do not establish universal transfer reliability or a separately observed cross-runtime reconstruction event.

[5] Author project state-repair and reconciliation record, publication-project control-plane custody repair, 10 September 2026. Unpublished internal record. The exact source is preserved in the governed project source map. The record supports the cited halt/repair, protected-byte, and separate-authority boundaries; it does not establish that the governance system prevents custody failure or guarantees substantive correctness.

References

- [6] L. C. Siebert, M. L. Lupetti, E. Aizenberg, et al., “Meaningful human control: actionable properties for AI system development,” *AI and Ethics*, vol. 3, pp. 241–255, 2023. doi:10.1007/s43681-022-00167-3.
- [7] A. Vella and K. Blincoe, “The Impact of AI Coding Assistants on Software Engineering: A Longitudinal Study,” arXiv:2605.23135, 2026.
- [8] M. Stolze and M. Strässle, “When Review Alone No Longer Scales: Layered Supervision in AI-Assisted Software Engineering,” arXiv:2608.26316, 2026.
- [9] K. Scarfone, M. Souppaya, and D. Dodson, *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*, NIST Special Publication 800-218, National Institute of Standards and Technology, 2022. doi:10.6028/NIST.SP.800-218.
- [10] J. Becker, N. Rush, E. Barnes, and D. Rein, “Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity,” arXiv:2507.09089, 2025. doi:10.48550/arXiv.2507.09089.