Sustaining the Product Engineer

1. The shared assumption

Three positions dominate the conversation about AI and software development. The first is that AI is a force multiplier for engineers, increasing throughput and freeing them to do more of what they were already doing. The second is that AI is coming for engineering jobs and developers should retire to the woods. The third is that AI is so unreliable that engineers will spend the next decade fixing the bugs it generates. The positions disagree about almost everything but they share an assumption.

The assumption is that the role being affected — the software engineer — is stable. AI is faster, slower, better, worse, threatening, useful; whatever it is, it is happening to a job whose shape is treated as fixed. The question being asked across all three positions is some version of *what does AI do to the engineer*. The question that is not being asked is whether the engineer was ever only what the productivity framing implies — someone whose job is to write code — or whether the role was always larger, and whether AI is changing the visible weight of its parts rather than the role itself.

A different signal is coming from the job market. Companies including Heron Data, PostHog, Microsoft AI, and others are increasingly hiring against a title — Product Engineer — that pre-dates the current AI moment by several years. The title was articulated by Mansour and Lemieux at Atlassian in 2019, well before AI tooling was a serious factor in engineering work. It described an engineer who owns outcomes end-to-end, engages with the why of a problem rather than only the what, and ships without hand-offs. Such engineers existed in 2019. They were rare.

The resurgence of Product Engineer as a title is the market's implicit recognition of something the productivity framing does not surface. AI has not created a new kind of engineering role. It has reduced the cognitive cost of one specific part — the coding — enough that the larger role the coding once obscured can now be the default expectation rather than the exception. The shift is already underway. The question this piece examines is not whether the shift is happening or what it means in the abstract. The question is what the shift requires in order to be sustainable — what the older medium was producing alongside code that the role depended on, and what deliberate practice now has to do in its place.

2. Why the role was rare

The role concept was not new in 2019, but as a hiring structure it was uncommon. Most engineering work was organised around narrower scopes, hand-offs at boundaries, and outcome ownership distributed across product managers, technical leads, and engineers operating within their slice of a feature. Engineers who took on outcome ownership alongside writing code did exist — competent

senior engineers have always operated this way informally. What was uncommon was companies treating this shape of work as the explicit, hireable default.

The reason was structural. An engineer formally accountable for outcome ownership while also writing the code that delivered the outcome was carrying two cognitively expensive activities simultaneously. Outcome ownership required holding architectural context, user context, and business context in mind across the lifecycle of a feature. Writing the code required sustained attention to the unit-level decisions that gave the feature its working form. Either activity was demanding alone. The combination, sustained as the formal expectation rather than as informal initiative, was difficult to staff at scale.

This is why the role appeared as a recognised hiring pattern at companies like Atlassian, Vercel, Linear, and Incident.io but did not generalise across the wider market. The structure was sound. The conditions that would let it become a default expectation rather than a niche pattern were not yet in place.

3. What changed

AI does not reduce the cognitive cost of outcome ownership. Holding architectural context, user context, and business context in mind across the lifecycle of a feature is still demanding, and AI does not do this work for the engineer. What AI reduces is the cost of the other activity — writing the code that delivers the outcome. The work of producing working code, which previously required sustained unit-level attention from the engineer, can now be substantially absorbed by AI tooling under the engineer's direction.

This is not a quantitative shift in productivity. It is a qualitative shift in what the role costs to perform. An engineer who would previously have been overwhelmed by the combined load of outcome ownership and code production can now sustain the first because the second has become cheaper. Companies are hiring against a description that previously fit a small population, because the population for whom the description fits has expanded. AI's role in this shift is not to make engineers do the existing job faster. It is to make a previously rare job structure the new baseline — and the cognitive demands of operating at that baseline are not the same as the demands of the job AI is assumed to be accelerating.

4. The yield of the medium

Code production, before AI, was paced by typing speed. The pacing was not designed and it was not chosen. Engineers could not produce code faster than they could type it, and they could not type it faster than they could think it through. It was an artefact of the medium — an involuntary constraint, and like most involuntary constraints, it produced side-effects nobody had set out to engineer.

The most important side-effect was that engineers acquired knowledge as a by-product of producing code. A developer working through a single unit of code over hours encountered the unit repeatedly, revisited its decisions, and accumulated a detailed understanding of its nuances. The understanding

was not the goal of the work; the working code was the goal. The understanding was a yield — what the work produced alongside the code. A senior engineer was, at root, an engineer who had accumulated enough yield across enough units to hold a coherent architectural picture of larger systems. Seniority was not a different kind of knowledge from juniority; it was the same kind of knowledge, accumulated.

AI removes the pacing. A feature that previously required weeks of unit-level engagement can be produced in days, with the engineer moving through the work in a different rhythm — bursts of generation, periods of waiting, rapid evaluation of candidate outputs, decisions made by elimination rather than by construction. The new rhythm is not slower or faster than the old one in any simple sense; it is structurally different. Where the old mode required the engineer to build a solution incrementally, the new mode requires the engineer to evaluate solutions presented to them. These are different cognitive activities. Construction builds understanding; evaluation builds judgement. They produce different kinds of expertise.

The consequence is that yield, in the form that pre-AI engineering produced it, no longer reliably occurs. The mechanism that generated nuance-level understanding — sustained engagement with single units over time — is the mechanism AI most directly compresses. The mechanism that generated architectural understanding — accumulation of unit-level engagements across many features and years — is the mechanism that depended on the first one. Both are weakened by the same change.

This produces a predictable shift in the kind of knowledge engineers hold. The new mode is biased toward architectural understanding, because evaluation forces engineers to consider how candidate solutions fit a larger picture. The new mode is biased away from nuance understanding, because evaluation does not require engagement with the line-by-line decisions inside a candidate solution. The result is engineers whose architectural judgement may be sharper than their predecessors' — they spend more of their attention there — but whose grasp of the details inside their own systems is shallower.

The failure profile follows from this asymmetry. Architectural mistakes are visible: they are large, structural, and immediately apparent in design discussion or system behaviour. Nuance bugs are invisible: they are small, embedded in the details of single units, and only surface as production incidents weeks or months after the code was shipped. Engineers operating in the new mode are well-equipped to catch the first kind of failure and poorly-equipped to catch the second. Code reviews and demos provide some compensation — an engineer who cannot walk through their own code reveals the gap — but only when reviews and demos are taken seriously enough to surface it. Where they are not, the code ships, and the bugs are found in production.

The asymmetry across experience levels is real but its shape is more troubling than it first appears. A senior engineer who entered the field in the pre-AI mode has substrate already built — the accumulated yield of years of unit-level engagement — and now operates in a mode that does not replenish it. A junior engineer entering the field now has no substrate and operates in a mode that does not produce one. The senior is in a better immediate position, but only because of an

inheritance from conditions that no longer obtain. Without deliberate compensation, both populations drift toward the same end state — engineers who hold architectural pictures whose foundations they have never built — at different rates. The problem is not juniors versus seniors. The problem is that the mechanism by which the field reproduced expertise across generations has stopped operating, and nothing has yet replaced it.

The Product Engineer role formalises a position that requires what the new mode does not reliably produce. Outcome ownership demands engagement with the why of a problem, which is architectural; it also demands engagement with the consequences of decisions, which lives in the nuance. An engineer who can articulate the architecture but cannot walk the code that implements it owns the outcome only on paper. The role is being hired for; the conditions under which it can be sustainably staffed are not yet in place.

5. Deliberate compensation

The new conditions do not self-correct. The mechanisms that produced yield as a side-effect of code production are not coming back, because the medium that imposed them is not coming back. Engineers operating at the new baseline who do not deliberately compensate for what the medium used to provide will end up holding architectural pictures whose foundations they have never built. Three disciplines, each addressing a specific loss, can serve as candidate responses.

The first is the separation of production from comprehension as distinct activities. Under the old medium, the two were bundled — engineers could not produce code without comprehending it, because typing was slow enough to require ongoing engagement with what was being typed. AI dissolves the bundle. Production can now occur without comprehension, and the engineer who does not deliberately schedule the second activity will end up with code they own only nominally. Comprehension as a deliberate activity means reading, tracing, and questioning code that has been produced — without producing more of it during the same session. This is not the same as code review, which is comprehension of someone else's output for the purpose of approval. It is comprehension of one's own output for the purpose of building the substrate that the medium used to build automatically.

The second is forced articulation through review. The old medium auto-detected gaps in understanding through the friction of writing — an engineer who did not understand what they were doing would get stuck at the next decision point. AI removes this friction. Code that the engineer cannot articulate flows as easily as code they can. The gaps are no longer surfaced by the medium; they have to be surfaced by a deliberate mechanism. Walking another engineer through what was built and why — answering questions, defending decisions, explaining the consequences of changes — is the most reliable substitute. The purpose is not to catch errors. The purpose is to expose gaps that fluent AI-assisted output conceals from the engineer themselves. An engineer who cannot walk their own code reveals the gap. The walk has to be required, and it has to be taken seriously enough by both parties to function as more than a ritual.

The third is the treatment of design documents as active artefacts rather than historical ones. Under the old medium, documentation was produced as a side-effect of comprehension — engineers who understood their code wrote design notes, architectural decisions, and rationale because the understanding was fresh. The documents were often read once and then faded into the backlog. The new conditions invert this. Documents are now read continuously, because they serve as the input that AI tooling consumes when producing code, and as the reference that engineers consult when evaluating that code. A document that is not current is not merely outdated; it is actively misleading the production process. This requires methodical pre-development design work, sustained throughout the lifecycle — not a ceremony performed before coding begins and then abandoned, but a continuous activity that produces and maintains the artefacts the new mode depends on. The design work is no longer separate from the production work. The design work is what makes the production work coherent.

These three disciplines share a common shape. Each names something the medium used to do automatically — bundle production with comprehension, auto-detect gaps in understanding, generate documentation as a side-effect — and each replaces it with a deliberate practice. None of them is novel. All of them have appeared in software engineering literature for decades, usually as virtuous practices that careful teams adopted and others did not. What changes under the new conditions is that they are no longer optional. The medium used to compensate for their absence; the new medium does not. Practices that were once nice-to-have become structurally necessary, because the structure that previously substituted for them is gone.

Whether these specific disciplines are the right responses, or whether better ones will emerge, is an open question. The point is not that this list is canonical. The point is that the responses must be deliberate, must address what the medium specifically removed, and must be sustained — because the conditions that previously made them unnecessary no longer obtain.

6. Closing

The dominant framing of AI in software engineering asks how much faster AI makes engineers at the work they were already doing. The framing assumes the role being affected is stable. It is not. AI has reduced the cognitive cost of one part of the engineering role — the production of code — enough that a previously rare role concept can now be staffed as a default. The Product Engineer title, articulated before AI was a serious factor in engineering work, names what becomes possible when the binding constraint shifts. The market is hiring against this title not because the role is new but because the conditions for it are new.

This piece is not a claim that engineering as it was practised before AI is being lost. It is not a claim that AI is producing worse engineers, or better ones. It is a claim that the medium of engineering work has changed, that the medium was producing side-effects beyond code that the role depended on, and that the disciplines which used to be virtues are now necessities. Whether the specific disciplines named here are the right responses is an open question. That responses are required is not.

The conditions under which the new role can be sustainably staffed are not yet in place. Naming what changed is the precondition for building them. The work of building is what comes next.