The Team That No Longer Fits

This piece treats evaluation as substrate-dependent, and therefore scarce and human. That treatment holds where the relevant knowledge remains tacit. Where it has been externalised into tests, conventions, and recorded history, parts of the same evaluation work become mechanical, a case developed further in The Bottleneck Moved.

1. The team that no longer fits

The standard engineering team at small-to-medium organisations has had roughly the same shape for two decades. The specific roles vary. Some teams have business analysts, others have product managers doing similar work, some have both. Some have dedicated architects, others have technical leads who absorb architectural responsibility part-time. But the underlying pattern is consistent. Requirements analysis, architectural design, code production, and verification are distributed across specialised roles, with handoffs between them. The most common configuration is a business analyst or product manager, one or more software developers, and a tester or QA function, often with a technical lead or architect attached to the team or shared across teams.

This structure is not arbitrary. It encodes a set of assumptions about what was scarce and what needed to be coordinated. Domain understanding was scarce, so it was concentrated in a specialised role. Architectural judgement was scarce, so it was concentrated in another. Code production was expensive in time, so multiple developers were needed to produce enough of it. Testing was a distinct skill and a distinct phase, so it was given to people who specialised in it. The team structure was an answer to a question about scarcity: where should we put the people, given what is expensive to do?

The question has changed. AI tooling has dramatically reduced the cost of code production, which means the role that was created to address production scarcity is no longer addressing the bottleneck. But the team structure has not yet caught up. Most organisations are still running teams whose composition assumes that production is the constraint. This piece argues that the structure should change, because what has become scarce is no longer what the team was designed around.

2. What production scarcity used to mean

In the pre-AI mode, producing working code was the activity that consumed most engineering time. A feature took weeks not because the decisions were complex but because the typing, debugging, and iterating took weeks. The cognitive work of understanding the domain, deciding on architecture, and verifying behaviour happened around the production work, but the production work set the pace.

This had three consequences for how teams were structured.

It made production capacity the resource that organisations bought. Hiring more developers meant shipping more software, more or less linearly. Engineering managers were judged on headcount and velocity, and both metrics assumed that throughput was the scarce thing.

It made specialisation efficient. If production took most of the time, the work of analysis, architecture, and testing could be spread across fewer specialists who fed the producers with specifications, designs, and verifications. Specialisation produced efficiency because the production layer was where the bottleneck lived.

It made coordination tolerable. The handoffs between analyst and developer, architect and developer, developer and tester, all carried coordination cost, but the coordination cost was small relative to the production cost. Spending a day on a handoff to save weeks of misdirected production was a good trade.

None of these consequences still hold. Production capacity is no longer the resource that organisations buy, because production is no longer the bottleneck. Specialisation around production no longer produces the same efficiency, because the layer it was protecting is no longer expensive. And coordination cost is no longer small relative to production cost, because production cost has collapsed while coordination cost has not.

3. What has become scarce

When the cost of producing software drops, the bottleneck moves. What it moves to is the cognitive work that production used to obscure. This work was always there. Pre-AI engineers did it alongside the typing, often without naming it as a separate activity. Now that typing has been compressed, this work is what remains.

The clearest description of what has become scarce is *evaluation*: the ability to look at code, behaviour, or a proposed design and judge whether it is correct, well-formed, and appropriate to the system it will live in. Evaluation is not the same as writing tests, though tests are part of it. It includes recognising when AI output is subtly wrong, knowing when an architectural choice will create problems two years later, understanding which parts of a system can absorb change and which cannot, and identifying when a specification has missed something the implementer will need to know.

Evaluation depends on substrate, the accumulated detail-level understanding that comes from sustained engagement with code. An engineer with substrate can evaluate quickly because they recognise the patterns. An engineer without substrate has to reason from first principles each time, which is slower and less reliable. Substrate is not transferable. It is accumulated, by individuals, over years.

This is the structural shift the rest of the argument follows from. Production is cheap. Substrate is scarce. Evaluation depends on substrate. Therefore evaluation has become the bottleneck, not because there is no evaluation happening, but because the value of good evaluation has risen sharply while the supply of engineers who can do it well has not.

4. The team structure that follows

A team structure designed around evaluation scarcity looks different from one designed around production scarcity.

The number of people needed is smaller, because evaluation does not scale linearly with headcount the way production did. Adding a second developer roughly doubled production capacity. Adding a second evaluator does not double evaluation capacity, because the evaluators have to agree, coordinate, and avoid duplicating each other's attention. Beyond a small number, additional evaluators produce coordination cost faster than they produce evaluation throughput.

The composition is different, because the specialisations that addressed production scarcity no longer carry their previous weight. The role of "developer who writes code" loses its distinctness when most code is produced by AI under engineer direction. The role of "tester who verifies what was specified" loses its distinctness when the same engineer can evaluate behaviour, generate tests, and trace edge cases without a handoff. The roles do not disappear. The activities are still required. But they no longer need dedicated people because the activities no longer consume enough time to justify specialisation.

The same compression applies to the product side, though it is less often discussed. A product manager translating customer needs into specifications was a separate role because the engineers receiving the specifications did not have the time or the breadth to do that work themselves. When the engineer's time is no longer consumed by production, and when AI tooling makes it easier to engage with users, data, and product judgement, the product responsibilities can move into the engineering role as well. This is a bigger consolidation than the engineering-side merges, because it crosses what has historically been a stronger boundary between engineering and product. But the logic is the same. Specialisation was efficient because production was expensive. When production becomes cheap, specialisation becomes coordination cost rather than efficiency.

The consolidation is not free. The engineer who can do all of this is not the same engineer who used to fit into a slot. Carrying outcome ownership, architectural judgement, evaluation, and product responsibility in a single role requires a combination of skills that has historically been rare, and which most existing hiring processes are not designed to identify. Companies that adopt this structure will find that the constraint shifts from "we cannot ship fast enough" to "we cannot find people who can carry the role we now need." This is a different problem, and probably a harder one, than the problem the structure was designed to solve.

What remains is the engineer who can do all of these activities at the level of judgement, with AI tooling absorbing the production work that previously required dedicated headcount. This is the product engineer: an individual who owns outcomes end-to-end, performs evaluation across the full lifecycle of a feature, and uses AI as the production layer rather than as a tool that augments their own production.

The team that fits this structure is small. Two or three product engineers, with one acting as the senior who carries the heaviest substrate and the most architectural responsibility. The work that a

larger team of six to eight people was doing can be done by this smaller team, because the work that used to consume most of the time has been absorbed by tooling, and what remains is the work that benefits from being concentrated in fewer, more capable people.

Companies should be hiring for this team structure rather than for the role-divided team. The hiring impulse to add more developers to ship faster is an artefact of production scarcity that no longer obtains. Adding more developers to a team where evaluation is the bottleneck does not increase throughput. It increases coordination cost.

5. Why this has not happened everywhere yet

The team structure described above is already visible in some organisations. Small companies, startups, and the parts of larger organisations that operate with more autonomy have moved in this direction, often without explicitly framing it as a response to a structural shift. They simply found that fewer, more capable engineers using AI tooling produced more than larger, more specialised teams using the same tooling, and reorganised accordingly.

Most organisations have not. The reason is not that they have failed to notice. The reason is that the reorganisation carries costs that small organisations do not face.

The largest of these costs is tacit knowledge. A team of six to eight people at an enterprise organisation holds substantial tacit knowledge about systems, processes, regulatory constraints, customer behaviours, and inter-team dependencies. Collapsing this team into three product engineers does not just require finding three people who can do the activities. It requires transferring the tacit knowledge from the people who currently hold it, which takes time and often does not fully succeed. The risk of losing tacit knowledge during the transition is real, and the consequences of losing it can be severe, particularly in regulated industries where the knowledge encodes compliance requirements that are not fully documented anywhere.

A second cost is the accountability surface. At enterprise scale, certain decisions, such as architectural choices that span multiple teams, regulatory commitments, and security postures, have consequences that exceed what a single product engineer can responsibly carry. Some specialisations persist at enterprise scale not because they are irreducible in principle but because the accountability they carry is too large to absorb into a generalist role without creating risk the organisation cannot accept.

A third cost is regulatory and compliance constraint. In sectors where separation of duties is a regulatory requirement, certain specialisations cannot be collapsed regardless of whether AI tooling makes the underlying work consolidatable. The regulation exists to create independent oversight: a QA function that did not produce the code it is verifying, a security review that is not performed by the engineer whose work is being reviewed, an architectural sign-off from someone other than the implementer. A single engineer who owns outcomes end-to-end cannot provide that oversight on their own work, and no tooling improvement changes this. In banking, healthcare, and other regulated industries, the role-divided team is not just an artefact of production scarcity. It is also a

deliberate mechanism for risk management, and the parts of it that serve that function will persist even as the production-scarcity parts do not.

A fourth cost is organisational inertia. Hiring practices, promotion ladders, performance review processes, team metrics, and budget allocation are all built around the existing team structure. Changing the team structure requires changing all of these, which is a multi-year effort that most organisations have not yet begun.

These costs explain the asymmetry. Small organisations face few of them and can reorganise quickly. Enterprise organisations face all of them and reorganise slowly. The direction of change is the same in both cases. The speed differs by orders of magnitude.

6. What this implies

The piece's recommendation, stated directly: companies hiring engineering teams should reorganise around evaluation scarcity rather than production scarcity. For small and mid-sized organisations, this means hiring product engineers rather than specialised software engineering teams. For enterprise organisations, this means beginning the longer-term work of transferring tacit knowledge into forms that fewer, more capable engineers can hold, while accepting that some specialisations will persist as long as the accountability surface they carry remains large or as long as regulatory frameworks require independent oversight.

The current state of the market, with small companies running ahead and enterprise companies running roughly the same as five years ago, is transitional. As small companies demonstrate what is possible at their scale, as tooling matures, and as the tacit knowledge of enterprise teams is gradually externalised into the documentation, context artefacts, and reference materials that AI-assisted teams require, the enterprise structure will compress toward the small-company structure. The direction is settled. The timing varies.

What this means for individual engineers is that the path to seniority no longer runs through accumulating production capacity. It runs through accumulating substrate, the kind of evaluation capacity that becomes scarce when production becomes cheap. Engineers who orient toward this will find themselves valuable in ways that engineers who continue to optimise for production speed will not.

What this means for hiring managers is that the role-divided team is now a structure that should be defended only when defended deliberately, not adopted by default. Hiring a business analyst, an architect, four developers, and a tester for work that three product engineers could do is no longer the safe choice. It is the expensive choice. The safe choice is the smaller team, but it requires hiring engineers who can carry it, which is a different problem from hiring engineers who can fit into a slot.

There is a problem this recommendation makes worse. The product engineer is an engineer with substrate, and substrate is accumulated through years of sustained engagement with code. If teams hire only for the role that requires substrate, fewer engineers will have the conditions under which

substrate is built. The pipeline that produces the next generation of evaluators is the same pipeline that the new team structure shrinks. This is not a reason to keep the role-divided team. It is a reason to recognise that the recommendation is one half of an answer whose other half, how the field continues to produce engineers capable of carrying the role, remains an open question, and one that organisations adopting the new structure will eventually have to address deliberately.

7. Closing

The dominant framing of AI in software engineering treats it as a productivity tool that lets existing teams ship faster. This piece has argued that the framing misses the structural consequence of the cost shift. When the cost of producing code drops dramatically, the team structures that were designed around production scarcity become inefficient. What was scarce has shifted, and what was efficient has shifted with it.

This piece is not a claim that the role-divided team should be abolished overnight, or that enterprise organisations should restructure tomorrow. The transition is real and the costs are real. Reorganising too quickly creates its own risks. It is a claim that the team structure that fits the new cost structure is smaller, more concentrated, and built around engineers who can carry evaluation across the full lifecycle of an outcome, and that organisations who continue to hire as if production were still the bottleneck are paying for capacity that no longer addresses what is scarce.

The work of reorganising team structures will take years. The work of recognising that the reorganisation is necessary can start immediately.