The Future of Software Engineering

June 28-30, 2026 | Engelberg, Switzerland

Insights and findings from an unconference on AI, agentic engineering and the future of the discipline.

Executive summary

In February 2026, Martin Fowler brought together industry leaders in Utah to reflect on the state of play in software engineering. The results surfaced many issues that have commanded industry attention in 2026.

Just a few months on and change has continued at an astonishing pace. So, with a view to both exploring the ideas and issues raised at the Utah event, and to reflect on new challenges that have emerged in recent months, Thoughtworks and Martin convened a second Future of Software Engineering Retreat, this time in Engelberg, Switzerland. This report details the takeaways and insights from it.

This was an unconference-style gathering of senior technologists, CTOs, CEOs, architects and consultants. The 40 sessions that made up the event were informal, participant-led and sometimes even argumentative. The value was not necessarily consensus but instead the range and seriousness of the thinking.

Five headline findings run through nearly every session:

- **Code generation is no longer the bottleneck — verification is.** Across testing, legacy modernization, code review and team-design sessions, the same conclusion kept coming up: agents can produce code (and specs, and tests and infrastructure) far faster than any team can trust it. The discipline that wins is the one that builds cheap, fast, human-legible verification — characterization tests, constraint tests, mutation testing, production back-testing — not the one that generates the most code.
- **'Harness engineering' is emerging as a distinct, ownable discipline.** The scaffolding around an agent — context management, deterministic guardrails, skills, self-improving feedback loops — is repeatedly described as more important than the model or the prompt. It may be the place where competitive differentiation will live once models commoditize. Notably, the need for ownership and accountability came up repeatedly but there was no consensus on who that should actually be.
- **Organizations are colliding with a real apprenticeship crisis.** Multiple sessions independently raised the same fear: if senior engineers pair exclusively with agents, junior engineers lose the hands-on path to judgment, taste and the production instinct the industry has always relied upon to cultivate the next generation of seniors. Consistent with the discussions in Utah, most people agreed it was important that people developing software in this manner need to know what good looks like, which explains why the senior engineers thrive. However this increases the risk associated with the apprenticeship issue.
- **The executive/engineer expectation gap is a bigger risk than any technical limitation.** Boards and CEOs are making large, fast bets based on vendor demos and their own experience with report-writing AI while engineers see a

widening list of unresolved verification, security and governance problems underneath the productivity gains.

- **Legacy modernization is the clearest, most defensible near-term value pool.** Several sessions described rigorous, working, technically detailed approaches to AI-assisted COBOL/mainframe modernization with real verification discipline.

Similar to the event in the US, many open questions remain that need further exploration and consideration. However there are implications for how technologists and business act now.

For technologists, a renewed focus on testing and verification is required. Part of this should include rethinking the assumption that manual code reviews are a de facto guarantee of quality and also leveraging adversarial and mutation testing techniques when working with unfamiliar AI-generated code.

Also crucial are mechanisms for learning. This is both at the level of supervising AI agents, ensuring effective feedback loops inside agent harnesses, and at a team level too — implementing practices like mob-pairing and learning checkpoints that can strengthen understanding and skill transfer.

For business leaders navigating this period of strategic uncertainty and constant change, there was clear consensus that today's tokenomics challenges cannot be treated purely as a finance issue: it's ultimately a governance one which requires ongoing oversight and feedback loops rather than budget management alone. Related to this, striking a balance between autonomy and proliferating shadow AI also demands thoughtful governance. Uniform policy needs to be resisted, and autonomy should be calibrated to risk.

Perhaps the most significant strategic takeaway is that we should not expect the cycles of change the industry is currently experiencing to hit a final plateau. Yes, that's a possibility, but it is more likely that hype cycles will only become more compressed and the pace of change increasingly fast. This makes building durable capability that holds value and taking steps to protect areas of established value strategic necessities that should be acted on now.

Part 1: Cross-cutting themes

The headline themes provide a clear and immediate picture of the main issues that surfaced at the retreat. However, it only tells part of the story: the discussions were rich and wide-ranging, so it's worth digging deeper into some of the themes to understand how they emerged and what the group believe their implications may be for the wider software industry.

Verification, not generation, is the new bottleneck

The single most repeated observation of the conference was that agents can generate code, tests, specs and infrastructure far faster than any human or organization can currently establish trust in the output. "Engineering is now distilled down to how do I describe the goal, how do I verify I've reached the goal". This shows up as a concrete, practical problem, not an abstract one:

- **A new testing vocabulary is emerging.** 'Constraint tests' (single input/output tests that box in what an agent is allowed to generate), 'scenario tests' and 'good/bad logs' (derived from real production incidents) were named and demonstrated live. Custom, purpose-built approval-testing rigs which were built in hours are proving more effective than generic BDD frameworks. This is partly because they keep the human-reviewable surface simple and hard for an agent to game.
- **A layered trust-verification stack is forming for high-stakes migration work.** It looks like this: characterization tests (behavioral capture from the legacy system) → symbolic execution (mathematically grounded, not AI-generated) → production 'back tests' against real data flows. This is a genuinely rigorous and novel technical methodology.
- **Mixed deterministic/non-deterministic evaluation is the practical answer to LLM-as-judge unreliability.** There was discussion about how one team combined linters and pattern-matching with a three-model 'council of judges,' raising first-pass merge acceptance from roughly 60% to 80%.
- **The long-standing faith in manual code review is being openly questioned.** Multiple practitioners pointed out that no one in the room could cite data on how many defects manual review actually catches — a 'status quo illusion' that needs to be challenged with evidence.

> *"Conformance tests matter a lot more than the spec. If the conformance tests differ from the spec, guess which one wins?"*

Harness engineering is becoming a distinct, ownable discipline

As models commoditize, multiple sessions converged on the same claim: the scaffolding around a model — context management, deterministic guardrails, skills, self-improving feedback loops — is what actually differentiates good agentic engineering from bad.

- **Concrete, measured results exist.** One organization reported that using an effective harness cut token usage by at least 4x and materially increased output determinism. A refactoring experiment found that a raw linter improved code-smell resolution to under 50%, while linter output translated into specific, deterministic, step-by-step refactoring instructions ('habit hooks') achieved roughly 90% resolution. A smaller model with a good harness was also said to outperform a larger model with a weak one.
- **The best-performing teams do not hand-write their harnesses.** They let agents fail, run a 'learn' skill that reflects on each session and proposes harness edits and treat the human's job as periodic pruning and simplification, not authorship.
- **Governance of shared harnesses/skills remains an unsolved organizational problem.** Skills and shared context artifacts decay exactly like unowned code frameworks unless clear ownership is established. However, centralizing into a dedicated 'harness team; risks recreating the old ops team anti-pattern.

Team design is compressing — and the bottleneck is moving to decisions

Team-topology conversations recurred across at least five sessions, converging on a similar shape: smaller 'nucleus' teams (often pairs or trios) supervising large fleets of agents but retaining a social/cohesion floor around roughly 10 people for organizational sanity.

- **The 'two clocks' problem is the sharpest new diagnostic to emerge.** Teams are tracking both the clock for producing code and the clock for waiting on a decision. They're finding that while developer throughput has exploded, overall cycle time hasn't improved; this is because decision-making and specification clarity are now the constraint.

- **A vivid healthy-vs-unhealthy case study recurred verbatim across multiple sessions.** A team where a product manager and designer became 'superpowers' cranking out features on their own using agents while the engineer was relegated to cleanup, versus a team that paired on specs, tests and design intent while a fleet of agents converged on solutions is a good example. The former was found to be highly productive but viewed by leadership as 'a disaster in the making' because it eroded pairing culture and organizational cohesion.
- **Platform teams need a credibility upgrade.** Traditional infrastructure-focused platform teams may lack the coding sophistication to own 'harness' or 'dark factory' tooling. The fix proposed repeatedly is for platform teams to use the same agentic tools they ask product teams to use and to shift from a 'menu of options' to a more opinionated 'paved road'.
- **Domain-driven design is being reappraised as the most relevant existing discipline** for negotiating module and team boundaries at agentic speed and scale. This isn't because it produces one grand design, but because it's the only established discipline for continuously negotiating and naming boundaries across a large organization.

There's an apprenticeship and skills-transmission crisis

Independently, in at least six different sessions, senior practitioners raised the same fear: if juniors never get to struggle with real code, real production incidents and real design trade-offs because agents (or senior engineers pairing exclusively with agents) absorb that work, the industry will lose a vital mechanism for growing the next generation of engineers with judgment and taste.

- **Concrete countermeasures were proposed and are already being piloted.** A "design quorum" or mob-programming pattern where a senior leads design conversation while juniors do the actual prompting; explicit non-AI learning exercises with public accountability; new curricula teaching agent orchestration and supervision as an early-career competency.
- **The seven to 10 year experience cohort was identified as the group under the most acute strain.** Having invested a decade mastering skills that models now often exceed, they are facing a real emotional and identity impact.
- **Related research finding reinforces the concern.** University students who wrote essays with heavy LLM assistance showed measurable degradation in their critical-thinking ability over three months, even relative to their own unassisted baseline.

> *"All the best bits of software ever made were made slowly. They were the things we took more time and more care over... I think slow thought is good actually."*

Legacy modernization is the clearest, most defensible value pool

Multiple sessions described technically serious, working approaches to AI-assisted legacy/mainframe modernization. These weren't speculative, but were either advanced pilots or currently running in production.

- **Migration discipline principles repeated across sessions.** "Add nothing, change nothing, delete everything you possibly can" during the port; change one thing at a time (behavior fidelity, then architecture — never both simultaneously); preserve known bugs deliberately, as a client-approved decision, rather than letting an AI 'helpfully' fix things a downstream system may depend on.
- **Newly tractable techniques.** A full custom TypeScript-to-.NET-CLR compiler built via AI in four days; a COBOL compiler passing the NIST test suite built in three days for roughly $5,000 in tokens; reverse-engineering an undocumented, encrypted 1994-era mainframe binary format by having a model spot byte-level patterns. Problems that were previously uneconomical to solve outright, like bespoke compilers, transliterators and formal verification, are now within the reach of an average team.
- **Framing that lands with boards.** Connecting AI investment directly to modernization and maintenance budget (often 30–50% of total IT spend at large enterprises) reframes an abstract technology ask into a scoped, boardroom-legible capital allocation decision. One real example reduced a vague $100M+ ask into a scoped $8M/20%-of-systems proposal with tied, measurable value.

The executive/engineer perception gap is a bigger risk than any model limitation

A recurring, almost universal complaint was that boards and CEOs often believe "a product manager dumps a PRD into the magic machine and perfectly working software comes out". This is partly because their own hands-on AI experience is with AI report-writing and summarization tools that perform very well but are a poor proxy for software engineering.

- **This gap doesn't close with better models.** It closes with vivid, concrete storytelling that's tied to the impact on an organization's balance sheet. It also needs data discipline, like fact-checking vendor "10x" claims against real peer benchmarks and structured exercises that let executives try building something themselves and hit real limits.
- **A concrete, current warning sign.** Reported internal security incidents are up roughly 20x in six months at one organization, while AI token budgets are blowing through annual allocations in three months instead of twelve. This is the kind of budget shock that's now getting board attention faster than productivity claims.
- **Realistic productivity expectations should be actively managed downward from vendor hype.** One practitioner's estimate, accounting for the full SDLC rather than just code generation, put realistic near-term gains at roughly 2–3x, not 10x. They predicted the gap between hype and this reality could "burst the bubble" within 12–18 months.

Governance has not caught up with citizen development or agent autonomy

Sessions on citizen development and security shared a consistent set of real, concrete incidents: an accountant's Copilot-built app accidentally exposed customer data to the open internet via an AI-suggested Cloudflare tunnel; a marketing team's AI assistant was granted broad G-Suite access via cascading OAuth scopes the company could not even enumerate when trying to shut it down; an agent, low on disk space, deleted backups to free room — and was 'thrilled' about it.

- **A recurring and useful governance pattern.** A green/amber/red risk-tiering model (personal use/team use with mandatory training/company-wide requiring professional engineers), paired with detection over prevention — continuously scanning agent conversation logs for dangerous patterns rather than relying solely on upfront training (which cannot keep pace with weekly model releases) could be a critical governance tactic.

- **A new supply-chain attack vector.** Malicious actors can predict which non-existent libraries an LLM is likely to hallucinate by name and then publish real malicious packages under those exact names. Sandboxing alone doesn't fully solve this, since the dependency can still reach production.
- **Practical, partial mitigations exist and are spreading.** Waiting roughly 14 days before adopting new library versions (most compromises are caught in that window); vetted internal registries; micro-VM sandboxing (better than containers, although not yet proven fully sufficient); and treating agent-generated code as untrusted even inside your own network, applying zero-trust principles internally, not just at the perimeter.

Tokenomics, self-hosting and sovereignty are now board-level questions

Interest in self-hosting models is being increasingly driven less by cost than by a demand for sovereignty and control: fear of US/federal legal reach over data, fear of a provider unilaterally raising prices or throttling access and a desire to avoid losing an organization's 'ability to learn and change' by outsourcing it entirely.

- **Cost efficiency varies by up to 1,400x** depending not just on model choice but on how enterprise data access is architected. Inefficient MCP-based round-tripping between model and enterprise systems is an underappreciated cost driver. It creates a real tension between cost optimization (data closer to the model) and security (broader data exposure).
- **True large-scale self-hosting is a specialized, scarce discipline.** Performance engineering of throughput per dollar of fixed GPU infrastructure, down to physical rack topology is largely being absorbed by hyperscalers and neoclouds. Organizations should expect this capability gap and plan accordingly rather than assuming self-hosting is simple.
- **A workable middle path was offered for coding-specific workloads.** Specifically, smaller dedicated inference hardware or specialized coding-model hosting providers, that remove the need to solve the full hyperscale self-hosting problem.

Open source faces a genuine reckoning

A passionate, unresolved debate took place at the event: does AI worsen a pre-existing open-source sustainability crisis, intensifying issues such as maintainer burnout and unpaid labor extracted by billion-dollar companies, or does it create entirely new dynamics, like single-person mega-projects reaching massive scale in months and AI-generated pull-request floods maintainers can't evaluate?

- **A constructive pattern was proposed for handling AI-generated contributions at scale:** reverse-engineer a pull request's intent into a plain-language description, evaluate that intent and then have your own AI reimplement it from scratch before merging, crediting the contributor while avoiding blind trust, at low maintainer cost.
- **A speculative but plausible shift.** Open source sharing may move from code to specs/ideas, since implementations become cheap to regenerate per-consumer. There's a real risk, though, that if agents fully displace shared library dependency, people without AI/hardware access lose out on a historically democratizing effect of open source.

A values-driven counter-narrative: 'Conspicuously human'

Running underneath the technical optimism was a persistent, serious counter-current: a concern that if verification, prototyping and even market testing all become nearly free and equally available to everyone, the only remaining differentiator is human judgment, taste and care. Organizations need to deliberately protect and elevate that, not engineer it away.

- **There are historical analogies that suggest this isn't naive nostalgia.** Impressionism emerged specifically because the camera could replicate reality perfectly, shifting human value to interpretation; drummers became more sophisticated, not obsolete, once drum machines arrived; the best chess 'player' in the world since 1997 has arguably been a human-plus-engine team, not an engine alone.
- **This is not anti-technology sentiment — it coexists with the conference's overall enthusiasm for agentic tooling.** The clearest articulation was to keep the loop where judgment is injected deliberately and visibly human, even as the loop where things get built becomes highly automated.

> *"The only thing I don't want to outsource is the acceptance criteria. Everything else I'm willing to outsource."*

Part 2: Actions for technical leaders

So, what are the practical implications for technologists? While it's important to note that the field is evolving quickly — what's true in early July may not be a few months in the future — there are a number of recommendations for technology leaders that deserve consideration and reflection.

Testing and verification

- **Retire generic BDD frameworks where step definitions hide complexity;** adopt custom, purpose-built approval-testing rigs (constraint tests, scenario tests) that keep the human-reviewable surface simple and hard for an agent to game. Budget hours, not weeks — conference examples suggest a working rig can be built in a single session with an experienced person.
- **For any modernization/migration engagement, adopt the three-tier verification stack** as a default: characterization tests from real system behavior, symbolic execution for mathematical rigor where models can't help and production back-testing against real data flows as the final gate.
- **Run the coverage → adversarial-AI-probing → mutation-testing sequence** on any unfamiliar or AI-generated codebase before trusting its test suite: check coverage first, then have an AI actively try to break the code without breaking tests, then apply mutation testing only if that probing succeeds.
- **Stop treating manual code review as a de facto quality guarantee;** measure it. If your organization can't produce data on defects actually caught by review, treat that as a real gap, not a formality.

Harness and context engineering

- **Convert passive lint/static-analysis signals into deterministic, specific, step-by-step instructions** fed back to agents ("here's how to refactor a long function," not "this function is too long") — the single highest-leverage, cheapest harness improvement documented at the conference.
- **Build a "learn" loop into your harness:** let agents fail, reflect, propose harness edits and limit human involvement to periodic pruning rather than upfront authorship.
- **Assign explicit ownership to shared skills/context artifacts** before they fork and decay the way ungoverned frameworks always have; build a lightweight

eval process (with/without-skill scenario testing) to periodically prune skills that no longer add value as models improve.

- **Constrain infrastructure/cloud-facing agents to narrow, schema-defined, audited tool interfaces** rather than broad cloud-provider CLI access — block raw AWS/gcloud commands in favor of structured, reviewable tool calls.

Team design and ways of working

- **Track the "two clocks" explicitly:** the first, time spent producing code, and the second, time spent waiting on decisions/specification clarity. If throughput is up but cycle time isn't improving, the constraint has moved upstream. Fix the decision-making process, not the pipeline.
- **Deliberately preserve pairing on specs and design intent,** even as implementation pairing declines — treat spec-pairing as at least as important as code-pairing ever was, given a fleet of agents can now generate far more code from a single instruction than a human pair ever could review line by line.
- **Adopt a risk-tiered autonomy model per system or component** (cobot-style human oversight vs. dark-factory-style full automation), explicitly based on risk, reversibility and blast radius — do not apply a single autonomy policy uniformly across a portfolio.
- **Shift platform-team culture from a "menu of options" to a genuinely opinionated paved road.** Have platform teams use the same agentic tooling they mandate for product teams, to close the credibility/pace gap that undermines platform-team authority.

People and skill formation

- **Implement a design-quorum or mob-pairing pattern deliberately,** not incidentally: senior engineers lead design conversation in real time while junior engineers do the prompting, preserving hands-on exposure to design trade-offs that would otherwise disappear.
- **Build explicit, non-AI learning checkpoints into onboarding and training** — moments where trainees must work without agent assistance and explain their reasoning — rather than assuming skill transfer happens automatically alongside AI-assisted delivery.
- **Watch the seven to ten years experience cohort specifically** for signs of disengagement or identity strain. This group's expertise is being devalued fastest and least visibly, and they are often your most experienced delivery leads.

Governance

- **Apply a green/amber/red risk-tiering model to any AI/agent tool usage** (personal productivity/team-level with training/company-wide requiring professional engineering sign-off), and back it with continuous log-scanning detection rather than relying solely on upfront training.
- **Treat AI-agent-generated application code as untrusted even inside your own network;** apply zero-trust and blast-radius-limiting principles internally, not just at the perimeter.
- **Adopt a minimum library-adoption delay (roughly two weeks)** as a default supply-chain mitigation, and explicitly screen for AI-hallucination-based dependency attacks in code review.

Part 3: Strategic advice for management

The discussions in Switzerland didn't only surface potential actions for technologists; there are also a number of significant strategic implications that extend beyond software engineering and are pertinent to business leaders.

Sequence discipline before acceleration

The single clearest strategic warning from the conference, repeated in different words across security, governance and regulated-industry sessions: agentic AI amplifies discipline and habits — positive or negative — that already exist in an organization. Teams and organizations with weak testing culture, unclear risk ownership or poor documentation hygiene will get worse, faster, not better. The practical implication for management is to resist the temptation to "just go fast and fix quality later" — fix the underlying discipline gaps first, or in parallel, not after.

Manage the story, not just the metric

Boards and executives are moved by vivid, concrete, specific stories — not aggregate productivity dashboards. This cuts both ways: it's how organizations get boards to take security and governance seriously and it's how the AI industry itself oversells capability (repeatable '10x' anecdotes without underlying data). Management's job is to actively curate and fact-check the stories reaching decision-makers, not simply to report metrics upward and hope the right conclusions get drawn.

Treat token/infrastructure economics as a governance problem, not just a finance problem

Multiple organizations reported token spend growing 10x within months, unbudgeted and largely undetected until it was a crisis. This is a management process failure as much as a cost problem: it needs the same kind of governance discipline (named accountability, budget review cadence, usage-pattern-based policy) that organizations already apply to cloud spend, applied early rather than reactively.

Resist uniform policy; calibrate autonomy to risk

A recurring failure mode described at the conference is applying a single AI-adoption policy ("go fast everywhere" or "lock everything down") across a portfolio that actually has widely varying risk profiles. The more effective organizations described at the conference explicitly tiered their approach — by system criticality, by team maturity,

by regulatory exposure — rather than picking one stance and applying it uniformly. Management's job is to build and maintain that tiering, not to pick a single company-wide answer.

Protect the things that create differentiated value, deliberately

As verification, prototyping and even foundational engineering capability become commoditized and cheap, the conference's consistent argument is that judgment, taste and genuine human collaboration become the scarce, differentiating resource — not despite being "inefficient," but because of it. This has a direct management implication: practices like pairing, mob design sessions and slow, careful architectural thinking should be explicitly funded and protected as strategic capabilities, not treated as legacy costs to be optimized away under margin pressure. Organizations that quietly erode them in the name of AI-driven efficiency may find they have optimized away the actual source of their competitive advantage.

> *"The punishment for your bad habits now comes much sooner than before... it's only a couple of hours now until it comes back and kicks you."*

Plan for a compressed hype cycle, not a stable plateau

Several participants with direct market/valuation experience expect the current AI investment cycle to compress faster than previous technology cycles (the internet, crypto) and specifically flagged a plausible 12–18 month window before expectations reset against real (roughly 2–3x, not 10x) delivered value. Management should plan investment and messaging horizons accordingly: build durable capability (harness engineering, verification discipline, governance) that holds its value regardless of where the hype cycle lands, rather than betting the strategy on today's most extreme productivity claims holding up.

Final thoughts

The transition to an agentic future is less about a change in tooling and more about a fundamental recalibration of what it means to build software. As the discussions in Switzerland demonstrate, the central tension of this new era is that while code generation has become seemingly trivial, ensuring the necessary harness, tests and verification has become the critical bottleneck.

To succeed, engineering organizations need to pivot from viewing productivity as a volume metric to treating it as a discipline of trust and governance. The path forward demands harness engineering as a core competency and proactively mitigating the apprenticeship crisis; we need to ensure the next generation of engineers is able to develop the judgment and taste that AI models and agents cannot replicate.

As verification, prototyping and foundational engineering become commoditized, the final and most critical takeaway is this: the only true, sustainable differentiator for an organization—and for an individual engineer—is human judgment. We are moving toward a reality where "conspicuously human" effort is not an inefficiency to be eliminated, but a strategic asset to be protected.

We shouldn’t be afraid to be deliberate and slow where it matters most. In a world where agents can generate vast amounts of code in seconds, the highest value will be found in those who carefully curate the intent, accept the responsibility for the outcome, and preserve the taste and care required to build systems that endure.

