When Software Development Stops Being the Bottleneck

Prabodh Ambale on how AI is compressing and redistributing the product lifecycle.

Omid Razavi · September 11, 2026

I worked with Prabodh Ambale when Customer Service Management was still an emerging product at ServiceNow. Even then, resolving a customer's issue required connections across operational systems, engineering teams, and the people responsible for completing the work.

At our SuccessLab Executive Forum in San Ramon, Prabodh returned to that broader view of engineering. Now GVP of CRM & Industry Workflows Engineering at ServiceNow, he described how AI is changing the entire product lifecycle.

His presentation, "AI is Rewriting the Product Lifecycle," identified four shifts: what teams build, how they build it, how they release it, and how they deploy it with customers. Across those shifts, work becomes faster and responsibilities extend across traditional functional boundaries.

My takeaway is that the redistribution of responsibility may matter as much as the increase in speed. As engineering produces more, organizations must strengthen the judgment, coordination, and customer understanding needed to turn that output into value.

What teams build

Prabodh's first shift concerns the product experience. In his framework, voice, chat, and messaging become primary ways for users to express what they need. Agents carry out straightforward work, while the fuller application interface remains available for more complex tasks.

This changes where product design begins. Teams must consider the outcome a user wants and the actions required to achieve it, alongside the screens and forms that support the work. Prabodh extends AI assistance across administration, design, and runtime, rather than limiting it to a conversational feature added to an existing application.

The architecture matters too. His slides describe a microservices stack with substantial use of open-source components that AI tools can work with effectively. They also identify product metadata as something to restructure so those tools can generate and update capabilities more efficiently.

For me, this adds another consideration to architecture decisions. Alongside stability, security, scalability, and maintainability, engineering leaders need to evaluate how well their chosen stack supports AI-assisted development. Decisions made years ago can influence how effectively a team works with agents today.

How teams build

Prabodh's second shift compresses the work from planning through deployment. His slides illustrate planning moving from sprints to days, building from weeks to days, delivery from days to hours, and deployment from days to minutes. These describe the direction of his framework, rather than a universal performance benchmark.

He attributes much of that compression to harness engineering: connecting agents with the context, tools, tests, and controls needed to coordinate a sequence of work. In his model, agents participate in discovery, prototyping, technical design, code generation, testing, review, and merging.

Product-domain context supports the agents throughout that process. Tests, reviews, and documentation develop alongside the change. Engineers retain responsibility for the decisions that allow work to proceed, with safeguards for rollout and rollback.

Prabodh also places cost within engineering governance. His slides treat unexpected increases in token consumption as product regressions. That makes the economics of running an agent part of the quality of the product being built.

My concern is what happens when production capacity grows faster than decision capacity. Someone must still validate the output, resolve dependencies, and decide whether a change is ready for customers. Faster generation can move the queue into review and governance unless those activities evolve too.

How teams release

Prabodh's third shift moves releases toward smaller, more frequent increments. His slides show a progression from semiannual releases through quarterly and monthly cycles to weekly delivery, with daily releases also part of the broader direction.

That cadence requires changes in planning. Teams take more responsibility for their release paths, while groups with shared dependencies come together for defined periods to deliver joint priorities. Design partners and early adopters participate before wider release, with customer success informing readiness for general availability.

His slides set an ambitious standard for upgrades: backward compatibility by default and no upgrade effort required from customers. I read that as a design goal that makes the faster cadence usable, rather than evidence that every customer deployment already meets it.

During the forum discussion, an audience member raised the other side of this equation: customers have their own testing, training, procurement, and release schedules. Prabodh emphasized preserving their choice about when to adopt changes.

Those two ideas belong together. Suppliers can make updates smaller and easier to absorb, while customers retain control over their operating environments. My view is that release speed becomes commercially valuable when customers can adopt the improvements with confidence.

How engineering works with customers

Prabodh's fourth shift brings engineering directly into deployment. His slides describe product and engineering teams working alongside implementation partners and professional services from discovery and scoping through configuration, testing, and go-live.

Forward deployed engineering, in this model, carries responsibility for realized customer outcomes. The same approach to harness engineering supports use-case discovery and execution in the field. Deployment governance tracks agent identities and authorization alongside usage, spending, and value measures such as time and cost savings.

The feedback path is central to his framework: what engineers learn with customers shapes subsequent product development.

I see that as the test of whether forward deployed engineering strengthens the business over time. An engagement may solve an immediate customer problem through specialized work. Its broader value depends on whether the learning improves the shared platform and makes future deployments more effective.

That distinction has consequences for margins, implementation quality, and the role of engineering. Leaders need to decide what remains specific to one customer, what becomes reusable, and who owns that decision.

The judgment that faster development requires

Prabodh ends with the skills and mindset required to operate this way. Engineers increasingly review generated work, manage teams of agents, repair the tools and instructions those agents depend on, and take responsibility for deployment outcomes. He also acknowledges that people's capacity to hold context does not expand at the same rate as development velocity.

His slides identify entry-level engineers as facing a particular challenge: they are expected to judge AI output while still developing their own technical judgment. He presents reskilling and upskilling as immediate needs and describes this part of the transformation as only partially resolved.

That raises an apprenticeship question for me. Junior engineers have traditionally developed judgment through bounded assignments, mistakes, and feedback. As agents take on more of that work, organizations need deliberate ways for people to build system knowledge and learn to recognize failures. Near-term productivity should support the development of future senior engineers.

Prabodh's presentation brought me back to our early work in Customer Service Management. Strong engineering depended on understanding the full path to resolution: the capability, the surrounding systems, and the people responsible for the customer's result.

My conclusion is that AI increases how much engineering can produce while making the decisions around that production more consequential. The emerging constraints are whether teams can judge the output, organizations can coordinate the change, and customers can realize the value. The engineer's contribution increasingly extends from the build itself to what happens after it reaches the customer.

Based on Prabodh Ambale's presentation, "AI is Rewriting the Product Lifecycle," and Omid Razavi's account of the SuccessLab Executive Forum discussion. Analysis and first-person reflections are Omid's.

More perspectives and conversations: successlab.us