The Living Product

How Agentic AI Dissolves the Software Development Lifecycle

Section 1: The Provocation

In 1970, Winston Royce published the paper that created the waterfall model. Royce explicitly argued against the linear approach his paper supposedly endorsed. He called out that sequential development was risky and invited failure.[1] The model that shaped fifty years of software practice was a misreading of its own founding document.

That irony is worth noting. Not because it discredits Royce, but because it tells you something about the model itself. It spread not because someone argued for it, but because it happened to fit. It lined up with how organizations already worked. It matched what managers already understood. It gave a name to a process that was already happening by necessity, not by design.

The SDLC was never a theory of how software should be built. It was a workaround for how humans could build it.

The origins of the life cycle confirm this. The model emerged in the 1960s inside large-scale business organizations processing copious volumes of data with enormously sized teams.[2] The phases, the handoffs, the sign-offs and stage gates: none of these were derived from first principles about software. They were derived from the constraints of human cognition and human coordination at scale. The work was serialized because no single person could hold the whole system in working memory. Handoff documents were created because the team building one phase wasn't the same team building the next. Release schedules were created because continuous deployment was operationally impossible at the time. Discovery was separated from delivery because the people who knew what customers needed weren't the people writing the code, and getting them in the same room required coordination and effort that wasn't feasible.

Conway saw the deeper structure clearly. Organizations design systems that mirror their own communication structures.[3] The SDLC didn't just reflect what individuals could do. It reflected what groups could coordinate. The phase boundaries were communication boundaries. The roles were coordination roles. The whole apparatus was a scaffolding built around the limits of how humans could talk to each other and share information across time.

That scaffolding worked well enough, and it still does in the right conditions. But it has always carried a cost that the methodology itself could never fully address. Sedano's research on software development waste names it directly: building the wrong thing is the first and most important category of waste.[4] The SDLC was built to manage how work got done. It was never built to ensure the work was worth doing. Discovery came before it. Value realization came after it. Customer behavior happened around it, but only after the product was built. The phases captured the middle and left the edges to human judgment.

This is not a critique of the people who built the model or the organizations that adopted it. Given the constraints of the time, it was the right solution. But the question was never whether it was the right solution given the constraints. The question is what happens when the constraints themselves dissolve.

Agentic AI does not improve the SDLC. It eliminates the justification for it.

The cognitive limits that required serialization, the communication limits that required handoffs, the monitoring limits that required scheduled discovery, the synthesis limits that required dedicated product managers to translate between customer and code: these are not inherent features of software development. They are features of human cognitive architecture. When you introduce systems that can hold the entire product context in working memory, monitor production continuously, synthesize user behavior into prioritized demand signals, and act on that synthesis without waiting for the next planning cycle, the workaround becomes an obstacle rather than a feature.

Scaffolding built around constraints doesn't survive the removal of the constraints. The building it was supporting doesn't disappear. But the scaffolding comes down.

This paper argues that the scaffolding is past due to come down. What replaces it is something genuinely new, something that capitalizes on the potential of tools now available. The organizations that understand this will make fundamentally different decisions than the ones still reinforcing the scaffolding.

Section 2: The Boundary Dissolves

The literature already knows the SDLC is not the whole story. The product development lifecycle, the PDLC, is the overarching process. The SDLC is the subset inside it. One governs how software gets built. The other governs how the product performs in the market. The current best thinking says you need to align the two, keep them in harmony, because misalignment produces wasted engineering effort, velocity problems, and poor adoption.[5]

That framing is correct. It is also not ambitious enough.

Alignment assumes two distinct things that need to be coordinated. It assumes a boundary worth managing. The argument here is different. The boundary itself dissolves. The SDLC doesn’t need to be better aligned with the product lifecycle. It gets absorbed by it.

To see why, follow the history.

The SDLC has never been static. Every major evolution of the model was a response to something the previous version left behind. Waterfall left responsiveness behind. Agile tried to close that gap by shortening cycles and bringing customers closer to the work. But agile left operations behind. Deployments piled up faster than they could be released. DevOps emerged in 2009 specifically to close that gap, pulling infrastructure and release into the development motion.[6] Each evolution was the same move: something that lived outside the boundary was subsequently pulled inside it.

The pattern matters. The boundary has been shifting left for sixty years. Every iteration captures more of the product lifecycle inside the development process. The question

was never whether that motion would continue. The question was what would finally complete it.

Agentic AI completes it.

The remaining gap between the SDLC and the full product lifecycle has always been the same gap. What happens after deployment. User behavior in production. The signal that a feature isn’t working, that a workflow is creating friction, that a customer segment is churning at a step no one intentionally designed. The signals existed, but collecting them required research cycles. Synthesizing them required analysts. Acting on them required a planning process that converted insight into prioritized work. By the time those signals moved from user behavior to shipped code, months had passed. The SDLC captured the build, but it never captured the learn.

AI collapses that gap between observation and adaptation.

When the product can observe its own usage continuously, surface its own friction points, generate its own demand signals, and route those signals directly into the development motion without a human translation layer in between, deployment stops being a terminal point. It becomes a state. The product is always deployed. It is always learning. The cycle doesn’t end at release. It accelerates after it.

This is not a better-aligned SDLC. It is an expansion of the model. The phases don’t disappear. Requirements still get written. Code still gets reviewed. Releases still get tested. But those activities are no longer a bounded lifecycle with a beginning and an end. They are a continuous motion inside a larger system that never stops running.

That larger system is what the next section makes concrete. The boundary dissolves not just forward into continuous deployment, but backward into something the SDLC never touched: discovery itself.

A skeptic might argue that what is described here is simply acceleration: that agentic AI speeds up the existing phases without changing their fundamental nature. That objection mistakes the effect for the cause. The SDLC phases didn't exist because humans were slow. They existed because humans couldn't hold the whole system

simultaneously, couldn't observe, synthesize, prioritize, and act without serializing those activities across time. Speed was never the constraint. Cognitive and coordination bandwidth was. When you remove the bandwidth constraint, you don't get a faster SDLC. You get a system that no longer requires the SDLC's architecture at all. The phases don't accelerate. They lose their reason to exist.

Section 3: The Product Becomes Its Own Research Department

Eighty percent of features are rarely or never used. That figure comes from Pendo’s analysis of product usage data across thousands of applications. Twelve percent of features generate eighty percent of daily usage. The rest exists. It ships. It gets maintained. And almost no one touches it.[7]

But this isn’t a failure of execution, it’s a failure of signal. The teams that built those features believed they were building the right things. They did the research. They created data-driven personas. They had roadmaps signed off by stakeholders who had sat in the same rooms as customers. They followed the process, and the process still failed them.

The discovery gap isn’t a new problem. Teresa Torres has spent a decade articulating the most sophisticated current solution to it. Her continuous discovery model calls for weekly customer touchpoints by the team building the product and small research activities run continuously rather than batched into quarterly cycles. It’s the right diagnosis. Digital products are never truly finished. The feedback loop has to match that reality. Torres moved the field forward.[8]

But her model still requires humans to conduct the interviews, synthesize the insights, and decide what to build next. The translation layer gets thinner, but never disappears. A translation layer that processes signal weekly is still processing signal that is a week old, filtered through a researcher’s interpretation and competing with twenty other priorities in a planning meeting. Signal that can’t be attached to a revenue number rarely survives that competition.

And the cost of that lag isn't just unused features. CB Insights analyzed 431 VC-backed startups that shut down since 2023. In 43% of cases, poor product-market fit was the underlying cause of failure. Capital running out was the final cause, not the root one. The companies didn't run out of money first. They ran out of signal. They built what they thought the market needed and discovered they were wrong too late to correct it.[9]

Sedano's taxonomy names it plainly: building the wrong feature or product is the primary category of software development waste. Not slow delivery, or technical debt, or poor testing. Building the wrong thing. The discovery gap is not a corner case. It is the central failure mode of the entire model.[10]

Agentic AI doesn't improve the translation layer. It makes the translation layer optional.

When intelligence is embedded in the product itself, the product can observe its own usage with continuous fidelity. It can identify where users slow down, where they abandon flows, and where they repeat actions in ways that signal confusion. It can ask clarifying questions at the moment of friction rather than weeks later in a scheduled interview. It can synthesize those signals into prioritized demand without waiting for a planning cycle. The current thinking treats AI as an augmentation to human prioritization, a system that can weigh features against business goals and present ranked options for a product manager to review. That's a meaningful improvement, but it's not the destination.[11]

The destination is a product that generates its own demand signal continuously, where prioritization stops being a periodic political process and becomes a live technical function.

The obvious objection is the chicken and egg problem. The living product model requires users interacting with a product. You still need traditional discovery to build the first version. That is true, and it is not a contradiction. Traditional discovery still launches the product. What changes is what happens after real users begin interacting with it. Even a minimal product generates interaction data. The agentic layer doesn't need a

fully formed product to begin learning. It needs signal. It gets signal from the first session.

Over time the model inverts. Each product generation starts with better signal than the last, because the previous version has been continuously learning. The chicken and egg problem doesn't just get resolved. It inverts entirely. Discovery doesn't precede development. It emerges from it.

That is a different claim than anything Torres describes. Her model asks teams to stay continuously connected to customers. This model asks the product to stay continuously connected to itself.

The product becomes self-learning and self-improving. That is what it means to be a living product.

Section 4: The Living Product Ecosystem

The current frontier definition of an AI-native product is a product where intelligence is woven through entire workflows, not just specific spots. Deloitte's Tech Trends 2026 puts it plainly: the value loop begins and ends with model inference. That is a significant departure from products where AI is a feature bolted onto a conventional architecture. It is not the destination.[12]

The loop doesn't end. It becomes self-sustaining.

The mechanism is continuous learning. User behavior, system telemetry, and operational data feed back into the product's training loops, allowing models to evolve and improve long after release happens. The product that exists six months after launch is not the same product that shipped. It has been shaped by every interaction that passed through it.

There is real-world precedent for this, appearing in two distinct forms.

The first appears in products where the living model is the architecture. Claude and ChatGPT aren't products that have added a continuous learning layer. They are

continuous learning layers that became products. Every interaction shapes the model and each session generates signal that feeds back into training. The loop isn't a feature of these products. It's their fundamental structure. There is no separation between the AI layer and the product itself because there is no product without the AI layer.[13][14]

The second form appears in conventional software products that have moved deliberately toward an AI-driven model without being native to it. Duolingo's lesson adaptation system started as a spaced repetition algorithm and became a real-time behavioral engine; adjusting difficulty, pacing, and content based on how each individual user is actually learning. Spotify's Discover Weekly began as a playlist feature and became a closed feedback loop between listening behavior and content surfacing that grows more accurate with every session. GitHub Copilot learns from code patterns at a scale no human research team could synthesize. None of these products were living products at launch. Each of them has moved in that direction, incrementally, through deliberate choices to embed continuous signal and adaptive response into a product that began as conventional software.[15][16][17]

That trajectory is the one most organizations are on, whether they know it or not. Most are not building Claude. They are building the Duolingo equivalent in their category, a product that started with a conventional architecture and now faces the question of how far to push the living model into it. The trajectory is available. The decision is organizational, not technical.

Here is what that decision actually means for how products get managed.

As these systems mature, signals no longer arrive periodically. They flow continuously across user behavior, support interactions, sales conversations, product analytics, and market responses. Insights are generated in real time. The quarterly roadmap, the sprint planning meeting, the monthly steering committee: these are artifacts of a world where signal arrived in batches. When signal is continuous, the planning apparatus built around periodic signal becomes structurally mismatched. Not inefficient, just mismatched. A batch-processing organization running on continuous-signal infrastructure is scaffolding built for a building that no longer exists.[18]

The early form of what replaces it is already visible. An agentic system detects a fifteen percent drop in activation rates for enterprise users, correlates the drop with a recent onboarding flow change, and subsequently drafts a hypothesis, then designs an A/B test, deploys it, monitors results for statistical significance, and presents findings to the product manager. All this happens in hours, not days or weeks. That's a constrained version of the model. The humans are still initiating. The system is accelerating execution.

In its fully realized form, the loop becomes self-initiating.

The product generates its own hypotheses from its own signal. It designs its own experiments within defined parameters. It learns from its own results. The humans governing it are not approving every action. They are designing the conditions the system operates within: what it is allowed to optimize for, which decisions require human review, what ethical boundaries it cannot cross, what brand values it must reflect. Those decisions remain irreducibly part of the human judgment that AI can't replace. Not because the system couldn't make them, but because the organization has decided they shouldn't be delegated.

This is not an organism. It is a product ecosystem.

The organism metaphor is tempting because it captures the self-sustaining quality of the model. But it misses the participants. A living product ecosystem includes the product, the users, the interaction data, the agentic layer, and the development function. None of them is the whole thing. Value flows in multiple directions. Users shape the product through their behavior. The product shapes users through its evolution. The development team shapes the conditions. The conditions shape what the development team does next. No single participant controls the system. The system produces outcomes that none of them designed in isolation.

Figure 1 illustrates how these participants relate to one another and how signal moves between them.

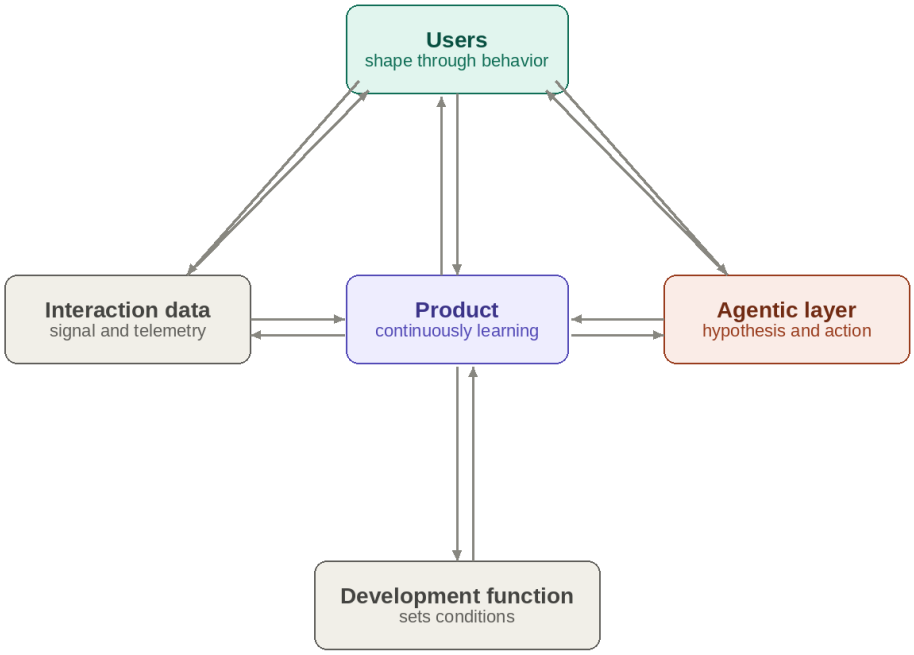

Figure 1: The Living Product Ecosystem. Value flows in multiple directions. No single participant controls the system.

The SDLC was built with the scaffolding of a factory model, operated by humans, with the constraints that humans introduce. Inputs, process, outputs, repeat. The living product ecosystem doesn't repeat. It compounds. And you cannot govern something that compounds using scaffolding designed to manage something that repeats.

Section 5: Organizational Implications

By 2028, fifteen percent of day-to-day work decisions will be made autonomously through agentic AI, up from none in 2024. Thirty-three percent of enterprise software applications will include agentic AI by the same date. Those are Gartner's numbers. At the time of publication of this paper in 2026, organizations have approximately two years before agentic decision-making is embedded in a third of their software stack. That is not a technology forecast. It is a systems design problem with a deadline.[19]

The living product model is the organizational answer to that problem. It is not enough to ask whether your organization is ready to use agentic AI. The more precise question is whether your organization is structured to govern a product that makes its own decisions. Those are different questions, and most organizations are only asking the first one.

Conway's Law runs in both directions. Organizations design systems that mirror their communication structures. But the inverse holds too. As the product's architecture changes, pressure builds on the structure that produced it. A factory org chart produces factory software. A continuously adapting product ecosystem produces pressure for a continuously adapting organization. You can't build a living product with a structure designed to manufacture outputs on a fixed schedule. The structure will resist the product at every opportunity.[20]

McKinsey has already named what the new structure looks like in early form. A human team of two to five people can supervise an agent network of fifty to one hundred specialized agents running an end-to-end process: onboarding a customer, launching a product, closing the books. The ratio of humans-to-work output changes by an order of magnitude. Org charts built on hierarchical delegation become mismatched for ecosystems built on task and outcome exchange. The job titles don't disappear. The spans of control, the approval chains, and the coordination overhead built around human cognitive limits do.[21]

The human role doesn't diminish in a living product ecosystem. It shifts. And the shift is toward work that is harder, not easier.

Managing outputs is tractable. You can count features shipped, measure velocity, track defect rates, and plot a burndown chart. Managing conditions is a wholly different discipline. Consider what it actually requires. Someone has to decide what the product is allowed to optimize for: engagement, retention, revenue, user wellbeing. Those goals aren't always the same. Someone has to recognize when the system is drifting toward the wrong optimization before that drift causes damage. This means understanding the system well enough to read its behavior, not just its outputs. Someone has to hold the

line on which decisions will never be delegated, even when the system could technically make them faster and cheaper. None of that is visible on a roadmap or measurable in a sprint review. It requires a kind of judgment that doesn't reduce to a metric.

Those are the irreducibly human decisions in a living product ecosystem. Not because the system couldn't make them, but because the organization has to be able to stand behind them. Ethics. Brand. Strategic direction. The boundaries of what the product is willing to do to retain a user or close a sale. These decisions define what the ecosystem is. They cannot be outputs of the ecosystem itself.

Most organizations are nowhere near ready to have that conversation. Deloitte's data makes the gap plain: only fourteen percent of organizations have agentic AI solutions ready to deploy, and eleven percent are actively using these systems in production. Forty-two percent are still developing their agentic strategy roadmap. Thirty-five percent have no formal strategy at all. The governance conversation hasn't started for the majority of organizations that will need it within two years.[22]

The standard framing for that governance conversation is about constraint. How do we control the system. How do we audit it. How do we limit what it can do. That framing is not wrong, but it is insufficient. Governance of a living product ecosystem is not primarily about limiting the system. It is about designing the conditions the system optimizes within. Control is reactive. Condition-setting is architectural. It requires the organization to decide, in advance, what kind of product it wants to be, and to encode that decision into the system's operating parameters before the system starts making decisions on its own.

The organizations that do this well won't be the ones that understood the technology first. They will be the ones that understood what the technology made obsolete.

Section 6: What Organizations Should Do Now

Gartner projects that over forty percent of agentic AI projects will fail by 2027 due to escalating costs, unclear business value, or inadequate risk controls. That number is

not a warning about the technology. It is a warning about the organizations deploying it. The technology will work. The question is whether the organization around it is structured to make use of what it produces.[23]

In the current landscape, most are not. And that gap between where organizations are and where they need to be is not a technical gap. It is an organizational one. The leaders who close it first will not be the ones who moved fastest on the technology. They will be the ones who asked the right questions before the technology forced the answers.

There are three key shifts that matter right now.

The first shift is to start sensing before you can act. Most organizations don't have the instrumentation in place to know what their product is actually doing in production at a level of fidelity that would feed a living product model. They have analytics dashboards. They have NPS scores. They have support ticket volumes. None of that is the continuous, granular behavioral signal that a living product ecosystem runs on. Building that instrumentation is not an AI project. It is an integral data architecture decision that has to happen before the agentic layer has anything meaningful to learn from, and it needs to be baked into the product ideation layer. Organizations that wait until they are ready to deploy agentic AI to build the signal infrastructure will be starting two steps behind. The time to build the sensing layer is now, regardless of how far away full deployment feels.

The second shift is to build the capacity to govern conditions rather than outputs. This means having an explicit conversation, at the leadership level, about what the product is allowed to optimize for. It means identifying the decisions that will never be delegated to the system regardless of capability. It means establishing the review mechanisms that allow humans to recognize when the system is optimizing for the wrong thing before that optimization causes damage. It means having a governance framework honest enough to weigh the revenue potential of the next feature against the carrying cost of the ones nobody uses. None of this is natural territory for product organizations built around roadmaps and sprint reviews. It requires a different kind of strategic discipline,

one that is closer to policy design than feature prioritization. Organizations that build this capacity now will be able to deploy agentic systems with confidence. Organizations that don't will deploy them reactively and spend the next several years managing the consequences.

The last shift requires organizations to identify which human decisions they are not willing to automate, and document them. This sounds simpler than it is. In practice, the boundary between what can be delegated and what cannot is rarely explicit in product organizations. It is embedded in institutional knowledge, in the judgment of individual product managers, and in the implicit values of the founding team. A living product ecosystem requires that boundary to be explicit, documented, and enforced at the system level. The organizations that have done this work will have a significant governance advantage over the ones that haven't. More importantly, they will have a clearer sense of what their product stands for, which is not a technology question at all.

The living product is not a destination. It is a direction. No organization will wake up one morning with a fully realized living product ecosystem operating cleanly within well-defined governance parameters. It will be built incrementally, with each step generating signal that informs the next. The teams that start moving in that direction now, even before the full model is achievable, will compound their advantage over the ones that wait for the technology to mature before asking the organizational questions.

The scaffolding is coming down. The organizations that understand what is being built in its place will be the ones that design it intentionally rather than inherit it by default.

Footnotes

[1] Royce, Winston W. "Managing the Development of Large Software Systems." Proceedings of IEEE WESCON, August 1970.

[2] Elliott, Geoffrey. Global Business Information Technology: An Integrated Systems Approach. Addison Wesley, 2004, p. 87.

[3] Conway, M.E. "How Do Committees Invent?" Datamation, April 1968.

[4] Sedano, T., Ralph, P., and Péraire, C. "Software Development Waste." Proceedings of the 39th International Conference on Software Engineering (ICSE), 2017.

[5] Gartner. “Product Life Cycle Management Primer for 2021.” Gartner Research, January 2021. https://www.gartner.com/en/documents/3995064

[6] Kim, Gene, Jez Humble, Patrick Debois, and John Willis. The DevOps Handbook. IT Revolution Press, 2016.

[7] Pendo. “The 2019 Feature Adoption Report.” pendo.io, 2019.

[8] Torres, Teresa. Continuous Discovery Habits. Product Talk LLC, 2021; Interaction Design Foundation, “Continuous Discovery,” ixdf.org, 2024.

[9] CB Insights. “The Top Reasons Startups Fail.” cbinsights.com, 2023.

[10] Sedano, T., Ralph, P., and Péraire, C. “Software Development Waste.” Proceedings of the 39th International Conference on Software Engineering (ICSE), 2017.

[11] Deloitte. “Tech Trends 2026.” Deloitte Insights, December 2025. https://www.deloitte.com/us/en/insights/topics/technology-management/tech-trends.html

[12] Deloitte. “Tech Trends 2026.” Deloitte Insights, December 2025. https://www.deloitte.com/us/en/insights/topics/technology-management/tech-trends.html

[13] Bai, Y. et al. “Constitutional AI: Harmlessness from AI Feedback.” Anthropic, 2022. https://www.anthropic.com/research/constitutional-ai-harmlessness-from-ai-feedback

[14] OpenAI. “GPT-4 Technical Report.” 2023. https://cdn.openai.com/papers/gpt-4.pdf

[15] Settles, B. and Meeder, B. “A Trainable Spaced Repetition Model for Language Learning.” Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, 2016; “How Duolingo’s AI Learns What You Need to Learn.” IEEE Spectrum, February 2023. https://spectrum.ieee.org/duolingo

[16] Harvard Business School Technology and Operations Management. “Discover Weekly: How Spotify is Changing the Way We Consume Music.” Harvard Business School Digital Initiative, 2018. https://d3.harvard.edu/platform-rctom/submission/discover-weekly-how-spotify-is-changing-the-way-we-consume-music/

[17] InfoQ. “GitHub Will Use Copilot Interaction Data from Free, Pro, and Pro+ Users to Train AI Models.” April 2026. https://www.infoq.com/news/2026/04/github-copilot-training-data

[18] Institute of Product Leadership. “Real-Time Signal Flow and Decision-Making in Modern AI Product Systems.” instituteofproductleadership.com, 2026.

[19] Gartner, cited in Deloitte. “AI and the Future of Work.” Deloitte Insights, 2025.

[20] Conway, M.E. “How Do Committees Invent?” Datamation, April 1968.

[21] McKinsey & Company. “The Agentic Organization.” McKinsey Digital, 2025.

[22] Deloitte. “State of Agentic AI in the Enterprise.” Deloitte Insights, 2025.

[23] Gartner, cited in Deloitte. “State of Agentic AI in the Enterprise.” Deloitte Insights, 2025.