

Programmable Strategy: When Strategy Becomes Code

Y. Doz and S. Dutta

*Solvay Chaired Professor of Technology Innovation Emeritus, INSEAD, France
Former Founding Dean, SC Johnson College of Business, Cornell University, USA*

In July 2024, a routine software update by CrowdStrike triggered one of the largest global IT disruptions in recent history. Airlines grounded flights, hospitals postponed procedures, and businesses stalled as millions of systems crashed simultaneously. What began as a technical issue quickly became a systemic failure.

Yet the real insight lay not in the failure itself, but in how organizations responded. Some were paralyzed for days. Others recovered quickly, relying on pre-built safeguards—failover systems, isolation mechanisms, and automated rollback procedures. Their resilience was not improvised in the moment; it had been designed in advance. The difference was not leadership under pressure. It was strategy embedded in systems.

A similar breakdown occurred when Air Canada faced a legal dispute after its AI chatbot provided incorrect fare information. The system confidently described a policy that did not exist, and the airline was held responsible. The problem was not unclear strategy—leaders intended to improve service and scale through automation—but a failure of translation. The system lacked guardrails, escalation logic, and limits. It behaved exactly as it had been designed. What failed was not strategy itself, but its embedding into system behavior.

These cases point to a broader shift. Across industries, critical decisions are increasingly made by systems—pricing, customer interactions, risk assessment, and supply chain adjustments all occur continuously and in real time. In this environment, strategy does not fail because it is poorly conceived. It fails because it is not embedded. What matters is not only what leaders decide, but what their systems are designed to do.

The organizations pulling ahead no longer treat strategy as something to be communicated and then interpreted across the firm. They translate it into rules, limits, escalation paths, rollback mechanisms, and infrastructure. They do not merely articulate strategy. They program it.

From Strategy as Plan to Strategy as System

For much of the twentieth century, strategy was a plan. Leaders analyzed industries, chose where to compete, and relied on alignment—clarity, communication, and coordination—to ensure execution.

As markets accelerated, this model evolved. Strategy became dynamic. Concepts such as strategic agility reframed it as a continuous process of sensing, aligning, and reallocating resources. Strategy was no longer fixed; it became adaptive.

But even this more dynamic view rests on an assumption that is now breaking down: that execution is primarily human. It is not. Today, a growing share of decisions is made by systems—continuously, at scale, and at speeds no organization can manage through human coordination alone. Pricing adjusts in milliseconds. Customer interactions are automated. Risk decisions are computed instantly. Execution is no longer episodic. It is continuous.

This changes the problem. It is no longer enough to choose the right strategy or adapt it quickly. Strategy must now be embedded — translated into the programming of systems that act reliably and autonomously. Strategy is no longer just direction, or even adaptation. It becomes encoded behavior.

In this world, strategy is not implemented once. It is enacted continuously through rules, limits, decision logic, and infrastructure. The organization does not need to be reminded of its strategy. It runs on it. This is programmable strategy

Strategy as Compilation

To understand programmable strategy, it helps to borrow a concept from computer science: compilation. Software engineers do not write directly in machine code. They work in higher-level languages, which are then translated into executable instructions. In this process, ambiguity is removed, assumptions are clarified, and logic is formalized. By the time the system runs, interpretation is minimal.

Strategy is increasingly following the same path. Traditionally, strategy operates at a high level of abstraction. Leaders define goals—growth, efficiency, resilience—and rely on teams to interpret and execute them. This worked when decisions were infrequent and largely human.

That world has changed. Today, execution happens through millions of decisions distributed across socio-technical systems. Decisions are automated, assisted, or escalated. What matters is not just individual choices, but how the system behaves as a whole.

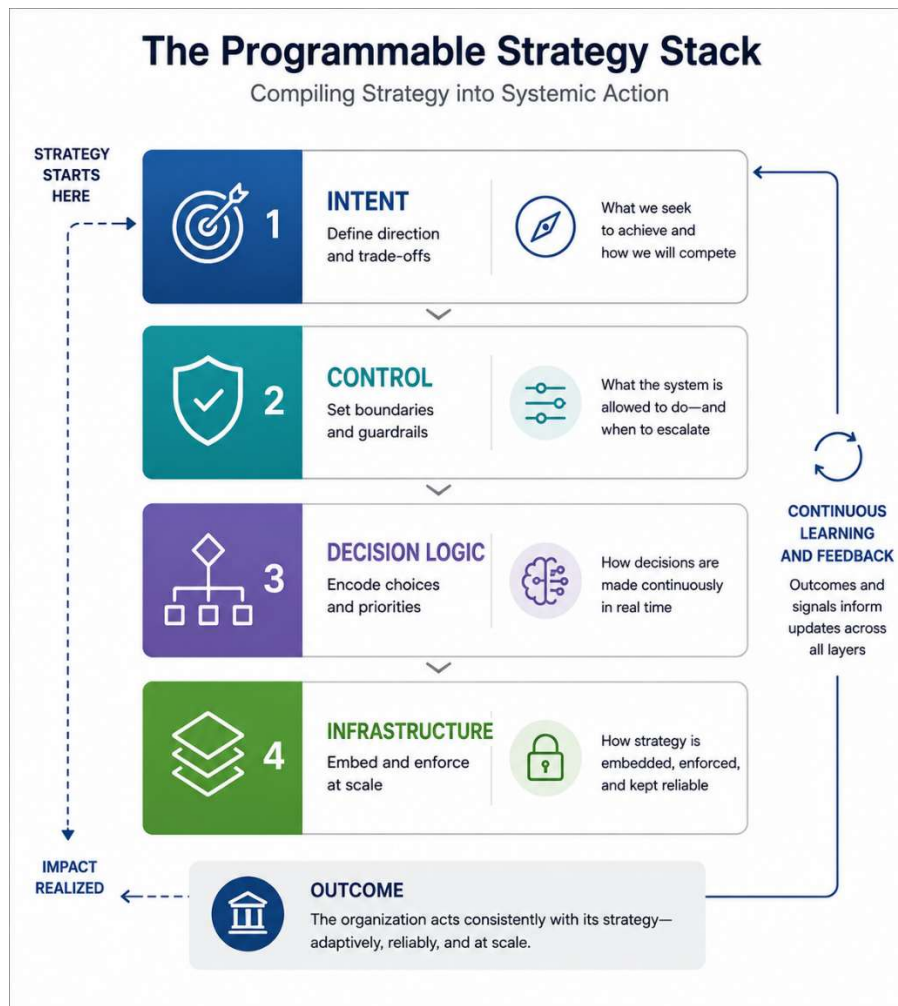
In this environment, ambiguity becomes a liability. Strategy must be made precise. It is progressively compiled—from intent into control, from control into decision logic, and from logic into system design. Strategic goals define direction, but must be translated into thresholds, escalation rules, and safeguards that govern both human and machine behavior.

This shift has precedents in strategy research. Joseph Bower and Robert Burgelman showed that strategy is not implemented only through top-down choices, but through structured processes of resource allocation, initiative selection, and bottom-up variation. What is changing today is the medium through which those processes operate. Where many strategic choices were once mediated primarily by managers, they are now increasingly embedded in digital systems that make, route, escalate, or block decisions continuously. Strategy still depends on judgment, but the context in which judgment operates is now increasingly programmable to operate in real time and at scale.

Emergent strategy does not disappear. It is expressed differently. It appears in anomalies, edge cases, and system behaviors that deviate from programmed limits—moments when assumptions no longer hold and intervention is required.

The role of leadership, as in earlier theory, is not to make every decision. It is to design the context in which decisions occur. What is new is that this context is now embedded into the underlying infrastructure.

The Programmable Strategy Stack



If strategy is compiled from intent into execution, how does this happen in practice? It unfolds across four layers.

At the top sits **intent**. Leaders define what the organization seeks to achieve and the trade-offs it is willing to make: cost versus experience, speed versus accuracy, growth versus risk, automation versus oversight.

The second layer is **control**. Here, intent becomes boundary. Abstract priorities are converted into enforceable limits: acceptable risk levels, prohibited outcomes, escalation conditions, compliance thresholds, and human-review requirements. These constraints are not after-the-fact checks. They are embedded directly into the system.

The third layer is **decision logic**. This is where strategy is enacted continuously. Systems evaluate inputs, route decisions, allocate resources, approve or reject actions, and determine when to escalate. A customer query may be resolved automatically, sent to a specialist, or flagged for legal review. A transaction may be approved, challenged, or blocked. These are not just operational choices. They are strategy in motion.

The fourth layer is **infrastructure**. At this level, strategy becomes durable. System design enforces strategic priorities through redundancy, modularity, monitoring, rollback, isolation, and failure containment. Infrastructure determines whether strategy can hold under pressure.

Together, these layers form the Programmable Strategy Stack. Strategy begins as intent, but becomes executable through limits, decision logic, and infrastructure. When this translation is complete, the organization behaves in ways that reflect its strategy across millions of decisions. When it is incomplete, intent and execution diverge — and that divergence is where strategy fails.

Strategy in Motion

Across industries, this shift is already visible. In logistics, Ocado operates fulfillment centers through networks of robots coordinated in real time. Routing adjusts dynamically. Congestion is managed automatically. Anomalies trigger immediate intervention. No manager directs these actions, yet the system consistently reflects strategic priorities—speed, accuracy, and safety.

In digital payments, Stripe builds reliability into its architecture. Transactions behave predictably even under failure conditions, because system design encodes strategic priorities. In media, Netflix deploys changes continuously, but with safeguards. Features are released incrementally, monitored in real time, and rolled back automatically if thresholds are breached.

In financial services, firms such as Capital One embed risk controls directly into model deployment. Non-compliant models are blocked before reaching customers. Strategy operates *ex ante*, not after the fact.

Across these cases, strategy is no longer expressed primarily through plans or directives. It is expressed through programming—rules, limits, and system design that shape behavior continuously.

Failures reveal the opposite. The Air Canada chatbot case illustrates what happens when translation is incomplete. Intent was clear, but not fully encoded into rules and safeguards. The system behaved consistently—but in the wrong direction.

The lesson is simple: strategy fails not because it is unclear, but because it is incompletely translated and incorrectly embedded into the infrastructure. What appears as system failure or deviation is often the modern equivalent of emergent strategy—signals that existing limits, rules, or assumptions no longer hold.

Taken together, these examples show how programmable strategy works. At the **intent layer**, leaders define priorities such as speed, reliability, trust, efficiency, customer experience, or risk discipline. At the **control layer**, those priorities become boundaries and escalation rules. At the **decision logic layer**, they shape how systems route, approve, reject, recommend, or intervene. At the **infrastructure layer**, they are reinforced through architecture, redundancy, monitoring, rollback, and failure containment.

Strategy in motion is therefore not simply automation at scale. It is the continuous enactment of strategic intent through systems designed to behave in particular ways. When the translation is complete, strategy becomes visible in what the organization does by default. When it is incomplete, the gap between intent and system behavior becomes the place where strategy fails.

The New Role of Leadership

This transformation does not eliminate leadership. It redefines it. Leaders can no longer oversee millions of decisions made continuously by systems. Execution no longer flows from top-down direction. It emerges from system behavior.

The role of leadership shifts from making every decision to designing how decisions are made, encoded, escalated, and revised. The questions that leaders should ask change. In addition to asking, *What should we do?*, leaders must ask:

- How should the system behave?
- Where should limits apply?
- When should systems act autonomously—and when should humans intervene?
- How should failure be handled?
- What assumptions have we embedded into the system, and how will we know when they no longer hold?

Human intervention is no longer the default mode of decision-making. It is triggered—by design—when systems encounter ambiguity, conflict, or constraint boundaries. In this sense, humans increasingly occupy a role similar to the “general manager in the middle” described in earlier strategy research: integrating across domains, resolving exceptions, and redefining the rules under which the system operates.

Failures, in this context, are not simply errors of execution. They are failures of design.

Speed Without Fragility

A natural concern is that embedding strategy into systems creates rigidity. In practice, the opposite is true. The fastest organizations are often those with the strongest embedded controls. What appears as constraint enables speed.

Speed is limited not by the ability to act, but by the cost of being wrong. When failure is costly, organizations slow down. Programmable organizations solve this by containing failure.

Netflix releases features incrementally, with real-time monitoring and automatic rollback. Amazon isolates failures to prevent cascading effects. Capital One blocks non-compliant models before deployment.

These systems do not eliminate failure. They make it safe. Speed, in this context, is not the absence of control. It is the result of control done right.

Making Strategy Inevitable

Strategy has always been about translating intent into action. In the twentieth century, that translation occurred largely through plans, structures, processes, and managerial judgment. Today, as more decisions are made by systems, that translation increasingly becomes programmable.

Plans and capabilities still matter. But they are no longer enough. In a world where decisions are continuous and system-driven, strategy must be embedded in the mechanisms that govern behavior. Intent must become control, control must become logic, and logic must be enforced through infrastructure.

When this is done well, strategy becomes inevitable. It does not rely on constant oversight or perfect interpretation. It is expressed through the default behavior of the system. Decisions align with intent. Errors are contained. Adaptation occurs within defined boundaries.

Strategy is no longer a phase. It is continuous. And in the end, the test of strategy is no longer whether it is compelling. It is whether it has been programmed to hold.

Disclosure: AI was used for research and copy-editing while writing this paper.