TOPCODE SPECIAL REPORT

The Operational Debt You Can't See

How product development methodology determines IT operations budgets for years to come — and why validation timing changes everything.

Based on 15 years of operational assessments across healthcare, financial services, manufacturing, and technology sectors.

topcode

Published by TopCode | 2026

The Hidden Crisis In Digital Product Development

Across industries, organizations are making multi-million-dollar decisions about digital products without understanding a critical fact: the methodology chosen during product development determines IT operational costs for years. Not months. Years.

Industry data paints a stark picture. Research consistently shows that approximately 45% of software features built are never used, and up to 80% deliver low or no value to end users. Yet every one of those features carries a permanent operational cost — support tickets, monitoring overhead, infrastructure consumption, and accumulated technical debt. The development cost is visible and bounded. The operational cost is invisible until organizations find themselves asking: why does this cost so much to run?

INDUSTRY REALITY CHECK

The numbers behind digital product operations:

- In our operational assessments across 40+ digital products, we consistently observe that substantial portions of built features receive minimal adoption, creating ongoing operational overhead regardless of usage.
- Industry research commonly suggests that 40-60% of software features are rarely or never used (though methodologies and exact figures vary). 80% of features deliver low or no business value.
- Organizations allocate 60-80% of their IT budgets to maintaining existing systems (industry estimates)
- Based on our 15 years of operational assessments, we've documented that 55-70% of operational costs stem from features with less than 10% adoption—regardless of what industry averages claim.
- Poor requirements gathering is the leading cause in 39% of software project failures.

These are not isolated incidents. They are systemic patterns rooted in how products are conceived and built.

In over fifteen years of conducting operational assessments for digital products, TopCode has observed the same pattern repeating across financial services, manufacturing, SaaS, retail, healthcare, and professional services. The conversation always begins the same way: a frustrated IT director, a CTO reviewing unexpectedly high operating costs, or a CFO questioning why a system that costs a significant amount to build costs even more to run.

The answer, almost without exception, traces back to a single decision made months or years earlier — one that rarely gets the scrutiny it deserves.

The Dangerous Misconseption

A persistent and costly belief dominates digital product development: that the best way to determine what to build is to ask stakeholders, document their requirements, and develop comprehensive solutions.

On the surface, it sounds reasonable. Talk to customers. Gather requirements. Get stakeholder buy-in. Build to specification. Yet this approach contains a fundamental flaw that most organizations never examine.

Opinions and behavior are different things.

When organizations ask customers whether they would like a specific feature, the answer is almost always yes. When organizations then build that feature with sophisticated logic, extensive integrations, and comprehensive functionality, adoption tells a different story. The opinion said yes. Behavior says no.

This is not a failure of the feature itself. It is a failure of validation. And that validation failure creates substantial, ongoing operational overhead that compounds year over year.

CLIENT INSIGHT: The Customer Portal That Cost More to Run Than to Build

One mid-size organization TopCode assessed had invested substantially in a customer portal. The executive team spent months gathering requirements from sales, customer success, product management, and end customers. The requirements document was extensive. A reputable agency was engaged.

Halfway through development, the timeline slipped. Two-thirds through, the budget increased. Near the end, scope was reduced. The portal launched — buggy, but functional.
Everyone celebrated. Launch meant success.

Two years later, the IT Director sat across from us: “We spent a considerable amount building this thing. Why does it cost even more to run?”

Usage analytics told the story. A meaningful portion of features showed minimal adoption — some under 5%. But those low-usage features were not operationally free. They still generated support tickets. They still required monitoring. They still created incident complexity.

> *One feature — a sophisticated workflow built at significant cost — was barely used. Yet it created a disproportionate share of incident volume because its logic interacted with multiple other systems.*

This pattern is not unique to one company or industry. It is systemic. The dangerous misconception is not that requirements gathering is unimportant — it is that requirements gathering, without behavioral validation, leads to operational obligations that organizations cannot easily escape.

What Topcode Has Discovered: The Two Paths

Through hundreds of operational assessments, a clear pattern has emerged. Every digital product initiative begins at a fork — two fundamentally different paths that determine not just the build timeline and cost, but the operational reality for years to come.

Path A: Build First, Validate Later

Requirements are gathered. Stakeholder buy-in is secured. Specifications are written. An agency is engaged or an internal team is assembled. Development proceeds for months. The product launches. Only then does the organization discover whether anyone actually uses it.

Most organizations follow this path. Whether operating under Waterfall or Agile methodology, the fundamental issue remains the same: building before obtaining behavioral proof of demand.

Path B: Validate First, Build What's Proven

Core assumptions are identified. Experiments are designed to test them. The organization seeks commitments, not opinions. Demand is proven before supply is built. Only validated features proceed to development. The product launches with confidence that adoption will follow.

Few organizations take this path. It feels slower initially. It requires saying "not yet" to stakeholders. It demands discipline. But the operational outcomes are dramatically different.

The Four Mechanisms of Operational Debt

1. The Incident Multiplier. In build-first approaches, TopCode consistently observes high and persistent incident volumes. When root causes are traced, the patterns are clear: a significant portion of incidents stem from features with low usage, another substantial portion from integration complexity between features, and still more from over-architected solutions that create performance bottlenecks. Actual software bugs are often the minority cause.

2. The Architecture Tax. When requirements are gathered from multiple stakeholders without validation, nobody knows which features will see actual usage. The architecture must therefore support everything. The result is a web of services calling core services, interacting with multiple data sources, each with their own synchronization requirements. When one component is slow, everything is slow. But diagnosing the source of the problem takes time — every time. Resolution times reflect architectural complexity, not team competence.

3. The Infrastructure Bill. During planning, stakeholders estimate optimistic usage scenarios. Infrastructure is provisioned accordingly. Launch happens at a fraction of projected usage. Months later, usage remains well below optimistic projections. But the infrastructure remains provisioned for a scale that was never validated. Organizations cannot easily optimize infrastructure that was provisioned for uncertain demand.

4. The Staffing Trap. TopCode has assessed operations where staffing appears disproportionate to the user base. Multiple support engineers for modest user populations. Multiple DevOps engineers keeping systems stable. Product owners in constant firefighting mode. The people on these teams are typically competent and hardworking. They are maintaining complexity that someone months or years earlier decided to build. Staffing requirements are largely determined when requirements are finalized. The bill simply arrives later.

WHAT TOPCODE HAS LEARNED: The Compounding Effect

Operational debt does not remain static — it compounds.

Year 1: The organization manages technical debt from unvalidated requirements.
Year 2: Year 1 debt plus new debt from rushed fixes created while firefighting.
Year 3: Years 1–2 debt plus architectural workarounds implemented instead of root-cause fixes.

TopCode has observed operational costs increase year over year in build-first approaches — not because teams deteriorate, but because the debt compounds. Meanwhile, validation-first builds often see operational costs stabilize or decrease. Same mechanism, opposite direction.

By Year 3, the comparison is no longer between similar systems. It is between a well-maintained asset and an accumulating liability.

What Actually Works: The Validation-First Approach

The organizations that achieve operational sustainability follow a consistent pattern. These are not theoretical recommendations — they are observable practices drawn from real engagements where outcomes were measurably different.

Phase 1: Define and Score What Matters

Rather than compiling exhaustive requirement lists, validation-first organizations start by answering four questions: Who are we building for? What specific problem are we solving? How will we know if it is worth building? What is the minimum that would prove demand? Features are identified and ruthlessly prioritized to core functionality.

Phase 2: Prove It Works

Clickable prototypes are built. Validation experiments are run with target users. The key distinction: these organizations require commitments, not opinions. They track which features users actually engage with — not which features users say they would engage with.

Phase 3: Build to Learn

Only validated features proceed to development. The product is deployed to pilot users. Actual usage is measured — not predicted usage. Infrastructure needs are validated with real load patterns, not optimistic projections.

Phase 4: Scale and Optimize

The product rolls out to the full user base. Infrastructure is right-sized based on validated patterns. Iteration is based on real usage data. Features that show weak adoption are removed, not maintained indefinitely.

CLIENT INSIGHT: The Alternative Path in Practice

One mid-size company TopCode worked with needed a customer portal but approached it differently. In the validation phase, they identified potential features and tested prototypes with target users. Some features showed strong behavioral validation. Others showed interest but no commitment.

Decision: build the validated features, park the others for future validation.
The operational difference was dramatic:

- **Lower incident volume** — fewer features meant fewer things that could break
- **Faster resolution times** — simpler architecture meant clearer troubleshooting paths
- **Smaller operational team** — less surface area to support, monitor, and maintain
- **Lower infrastructure costs** — right-sized for validated demand, not projected optimism
- **Lower vendor dependency** — clean architecture and clear documentation reduced reliance on original builders

The Mechanical Differences

The differences between these approaches are not philosophical — they are mechanical. Build-first approaches tend toward high incident volumes that persist. Validate-first approaches tend toward lower volumes that stabilize quickly. Build-first approaches see extended resolution times. Validate-first approaches see shorter ones. Build-first approaches experience higher change failure rates due to unvalidated feature coupling. Validate-first approaches maintain lower failure rates through properly isolated components.

The logic is straightforward: every feature not built is incident volume prevented. Every validated requirement creates a simpler architecture. Simpler architectures are faster to debug. Organizations cannot have incidents in features they did not build.

Why Organizations Resist — And Why Resistance Is Costly

Despite the evidence, the shift to validation-first development encounters consistent resistance. TopCode has encountered the same objections across industries, and each one reveals a deeper organizational challenge.

"We Know What Users Need — We Asked Them"

This is the most common and most costly assumption. Users reliably report what they think they want. They do not reliably predict what they will actually use. These are fundamentally different data points. Organizations that treat stated preference as behavioral proof consistently over-build and under-deliver on adoption.

"Comprehensive Features Are Better Than Minimal Ones"

Comprehensive features create comprehensive operational overhead — regardless of whether users adopt them. Every feature represents permanent operational cost unless an organization deliberately sunsets it, which most avoid because "someone might be using it." Minimal validated features create minimal operational overhead, then expand based on proven demand.

"We Can Fix Operational Issues After Launch"

This may be the most damaging misconception. Operational cost structure is largely determined during requirements, not during operations. By launch, the organization is committed. Architectural complexity cannot be easily reversed post-launch. Features cannot be removed without rebuild cost. Over-provisioned infrastructure cannot be easily right-sized. Operational reality is locked in when the architecture is approved. The implications simply do not become visible until later.

WHAT TOPCODE HAS LEARNED: The Conversation IT Leaders Don't Have

In most organizations, the budget conversation focuses entirely on build cost:

CFO: "The agency quoted a substantial amount for the build."
CTO: "That's within budget if it solves the problem." Approved.

What is almost never adequately discussed is operational cost. Because requirements have not been validated, nobody can accurately predict incident volume, infrastructure load, or staffing needs.

Operational cost structure is determined when requirements are approved. Organizations just do not see the full picture until they are living it.

The pattern TopCode observes is consistent: service management maturity cannot fix architectural immaturity. IT operations leaders optimize what they are given. But what they are given was determined during requirements — often before operations leaders were involved in the conversation.

Why This Matters More Than Ever

The financial stakes of this pattern are accelerating. With IT budgets under increasing scrutiny, rising infrastructure costs, and growing pressure to deliver more with less, organizations cannot afford to carry the hidden weight of unvalidated operational obligations.

Industry analysts estimate that organizations spend 60–80% of their IT budgets simply maintaining existing systems. Research from Gartner projects that by 2025, companies would be spending 40% of their IT budgets on maintaining technical debt alone. Developers report spending roughly a third of their work week addressing technical debt, time that could otherwise go toward building new value.

The organizations that master validation-first development are not just avoiding waste. They are building a compounding competitive advantage. While competitors cycle through escalating maintenance costs, firefighting, and eventual “modernization initiatives,” validation-first organizations are iterating, improving, and expanding on a stable operational foundation.

INDUSTRY REALITY CHECK

The widening gap between build-first and validate-first organizations:

Build-first trajectory: Significant initial build investment → ongoing operational costs that increase → Year 2 refactoring investment → compound technical debt → eventual “modernization initiative” that may repeat the cycle.

Validate-first trajectory: Smaller initial validation + focused build investment → lower ongoing operational costs → Year 2 savings fund validated enhancements → operational costs stabilize or decrease as the system matures.

By Year 3, the comparison is no longer between two versions of the same system. It is between a well-maintained operational asset and an accumulating operational liability. One improves. The other degrades.

The Pattern That Changes Everything

Once leadership teams see this pattern, it changes how they evaluate every decision. Not “can we build this?” but “should we build this?” Not “what is the development cost?” but “what is the total cost of ownership?” Not “what do stakeholders want?” but “what will users actually adopt?”

It changes how vendors are evaluated. Not “how fast can you build?” but “how do you validate?” Not “what is your feature list?” but “what is your approach to operational sustainability?”

And it changes how success is measured. Not just “did we launch?” but “did operational costs align with expectations?” Not just “did we hit the timeline?” but “did we validate before building?”

Moving From Insight To Action

The evidence is clear: build-first approaches create operational debt that compounds over time, while validation-first approaches create operational efficiency that improves over time. The gap is not in code quality. It is in what was validated before a single line of code was written.

But knowledge without implementation changes nothing.

Questions Every Organization Should Ask

For organizations evaluating a build decision right now — internal project, vendor selection, or agency engagement — the following questions can reveal whether the path ahead leads to operational sustainability or accumulating liability.

Before requirements finalization: Which of these requirements are validated with user behavior, not just opinions? What is the operational cost model if we build all of this versus validating first and building less? What are our incident volume and infrastructure assumptions based on?

During vendor or agency selection: How does the vendor determine what to build? How do they validate requirements before development? What can they share about operational sustainability of systems they have built?

After launch (for those already operational): What percentage of operational cost goes toward features with low adoption? What percentage of incidents involve features built speculatively? What would we not build if we could start over with current knowledge?

HOW TOPCODE CAN HELP

For IT leaders and organizations ready to move from insight to action, TopCode provides the operational assessment and validation methodology that makes the difference between accumulating liability and building sustainable assets.

Operational Assessment: A comprehensive analysis of current operational costs, incident patterns, infrastructure utilization, and feature adoption — revealing the true cost of ownership and identifying immediate optimization opportunities.

Validation-First Strategy: A structured approach to validating requirements before development begins, ensuring that what gets built is what gets used — and that operational costs align with expectations from day one.

Ongoing Operational Optimization: Continuous monitoring and improvement of operational efficiency, helping organizations maintain the competitive advantage that comes from sustainable systems.

> **The multi-million dollar decision is not just the build decision. It is the validation decision. And the organizations that make it consciously — with the right methodology and the right partner — are the ones that build operational assets instead of operational liabilities.**

About the Author

The report was written by Rony Plevnik, a TOPCODE specialist who specializes in operational assessments for digital products. With over 15 years of experience, TopCode has observed and analyzed patterns in how product development methodology affects operational sustainability across healthcare, financial services, manufacturing, technology, and other sectors. The consistent finding: operational sustainability is largely determined during requirements — not during operations — and validation timing significantly impacts long-term cost of ownership.

topcode

TOPCODE d.o.o.
Smoletova ulica 13,
1000 Ljubljana, Slovenia
info@topcode.si