SAMPLE REPORT · ANONYMIZED COMPOSITE ENGAGEMENT

Architecture debt is materially constraining the organization.

Based entirely on the 32 answers a respondent provides. No systems, code, or documentation were inspected.

68

OF 100 · HIGHER = MORE DEBT

HIGH

BUSINESS RISK
Material

MODERNIZATION URGENCY
Near-term

HIGHEST-RISK DIMENSION
Delivery & DevOps

PRIMARY CONSTRAINT
Time to market

EXECUTIVE SUMMARY

This organization's Architecture Debt Index of 68 places it in the **High** tier: architecture debt is no longer an engineering inconvenience but a measurable business constraint. The pattern in the answers is consistent, a capable team (Security & Compliance scored 38, your strongest dimension) operating inside a delivery process the architecture actively resists.

The binding constraint is **Delivery & DevOps (85)**. The answers describe releases coordinated as a single unit, rollback paths that are untested or unavailable, and a blast radius that becomes known only during implementation. In practice this converts every change, however small, into a scheduling negotiation, which is why the felt symptom is time to market rather than outages.

The good news in the data: this is a process-and-boundaries problem, not a skills problem. The recommended starting point is narrow: establish a safe rollback path and decouple the release train for the one or two components that change most often, before any broader modernization is scoped.

Eight dimensions, scored where it hurts.

SCORE = DEBT · 0 GOOD · 100 BAD · FOUR QUESTIONS PER DIMENSION

Architecture & Modularity · SCORE — **72**

Component boundaries are not containing change: the answers describe coordinated multi-system changes with a blast radius discovered during implementation. Expect this to amplify every other dimension's cost until ownership boundaries are made explicit.

Scalability & Performance · SCORE — **65**

Growth currently requires manual intervention and produces disproportionate infrastructure cost. Not the binding constraint today, but it inherits risk from modularity: scaling a tightly coupled system means scaling all of it.

Reliability & Resilience · SCORE — **70**

Failures propagate: unrelated capabilities degrade when a single component fails, and recovery relies on people rather than design. Combined with the rollback flag, incidents are longer and riskier than they need to be.

Security & Compliance · STRONGEST — **38**

The strongest dimension. Identity, secrets, and data controls are broadly in place, evidence that the organization can execute discipline when the architecture permits it. Preserve this while changing delivery.

Delivery & DevOps · HIGHEST RISK — **85**

The binding constraint. Platform-wide release trains, untested rollback, and coordination-heavy deployments convert every change into schedule risk. This is where the 90-day priorities start.

Observability & Operations · SCORE — **74**

Diagnosis is slow: teams reconstruct incidents rather than observe them. Improved telemetry is a prerequisite for the delivery changes, you cannot decouple releases you cannot watch.

Data & Integration Architecture · SCORE — **68**

Shared data ownership couples teams that should move independently. API boundaries exist but leak: schema changes still require cross-team coordination.

Maintainability & Modernization · SCORE — **71**

The architecture resists economic evolution, knowledge concentration and historical context stand in for documentation. The rewrite reflex in the answers is a symptom: incremental paths look harder than they are because boundaries are unclear.

Four structural risks, independent of the score.

Named by individual answers rather than by the aggregate. Each one is a specific, checkable claim about how the system behaves.

No Safe Rollback Path

DELIVERY RISK

Releases cannot be reliably reversed, so every deployment carries the full cost of being wrong. This inflates release ceremony, batch size, and fear, each of which further slows delivery.

Triggered by: Q19, "Rollback is untested or effectively impossible once a release is out."

Monolithic Release Train

TIME TO MARKET

Independent changes wait on a platform-wide release cycle, coupling every team's schedule to the slowest. Cycle time becomes a function of coordination, not code.

Triggered by: Q6, "The platform effectively has to be released as a single unit."

Knowledge Concentration

CONTINUITY RISK

System ownership depends on tribal knowledge held by specific people. Every architecture decision, and every incident, has a bus-factor problem attached.

Triggered by: Q11, "Ownership depends heavily on tribal knowledge and historical context."

Unpredictable Blast Radius

CHANGE RISK

The full impact of a change is discovered during implementation rather than planning, which makes estimates unreliable and pushes teams toward big-bang releases.

Triggered by: Q1, "The full blast radius is difficult to predict until changes are underway."

QUESTIONS LEADERSHIP SHOULD BE ASKING

Bring these to your engineering organization. The answers, and how confidently they arrive, are themselves diagnostic.

1. If we shipped a bad release right now, how long until customers are back on the previous version, and when did we last test that?
2. Which two components change most often, and what stops them from releasing independently today?
3. How much of last quarter's roadmap slippage was coordination cost rather than engineering effort?
4. Who are the three people whose departure would strand a system, and what is written down?
5. What would we have to believe for a rewrite to be cheaper than incremental decoupling? Has anyone priced both?

Suggested 90-day priorities.

Sequenced to reduce the binding constraint first. Each is small enough to run alongside delivery commitments.

01 **Establish a tested rollback path** · WEEKS 1–4

For the single most frequently deployed component: automate the rollback, test it in production conditions, and make it the release gate. This converts deployment from a bet into a decision.

02 **Decouple one component from the release train** · WEEKS 3–8

Pick the component with the highest change frequency and give it an independent deployment pipeline with contract tests against its consumers. One proof, then repeat.

03 **Instrument before you decouple further** · WEEKS 6–10

Add release-scoped telemetry, error rates, latency, and business signals per deployment, so independent releases are observable. This addresses the Observability score (74) where it matters first.

04 **Map ownership boundaries in writing** · WEEKS 8–12

A one-page ownership map per component: what it owns, what it depends on, who decides. Directly attacks knowledge concentration and makes the next decoupling candidates obvious.

NEXT STEP · SENIOR-LED · 90 MINUTES · FREE

The Architecture Debt Review

This report identifies where to look. The review determines why the constraint exists, what it costs, and which changes are worth making, a 90-minute working session with a senior architect, ending in a one-page findings memo. Four slots a month. For respondents, the report doubles as the intake, the session starts from it.

egiconsulting.com/architecture-review · contact@egiconsulting.com · (972) 767-5930

Method: the Architecture Debt Index is a 32-question scenario-based self-assessment across eight dimensions, scored 0–100 where higher indicates more debt. This sample reflects a composite of real engagements, anonymized; scores, flags, and recommendations show the format, not any single client. The index is a screening instrument, not an audit; no systems, code, or documentation are inspected. Take the assessment at egiconsulting.com/resources/architecture-debt-index.