SAMPLE ARCHITECTURE RISK REVIEW

# What I Look For Before Recommending a Rewrite

Illustrative review for a growing AI / SaaS team whose feature velocity has outpaced its production architecture.

REVIEW OBJECTIVE

**Identify the few technical risks that can create disproportionate business impact - and separate urgent stabilization from expensive redesign.**

## ILLUSTRATIVE SYSTEM CONTEXT

A 30-person AI / SaaS company. Product usage is growing. Engineers ship quickly with AI coding tools. Services, data flows and ownership boundaries have evolved organically. Incidents and deployment anxiety are beginning to rise.

| | | |
|---|---|---|
| HIGH | **Change Blast Radius** | Small changes can affect too many services or critical paths. |
| HIGH | **Data Coupling** | Shared databases and unclear ownership make failures hard to isolate. |
| HIGH | **Observability Gaps** | The team can see symptoms, but not reliably explain causality. |
| MED-HIGH | **Async Failure Modes** | Queues, retries and background work can hide partial failure. |
| MEDIUM | **AI Production Boundaries** | Model behavior, state and evaluation are not treated as production dependencies. |

PRINCIPLE

**The goal is not to maximize architectural purity. It is to reduce the probability and cost of the failures that matter most.**

SAMPLE FINDINGS

# Risk Map -> Decision -> Action

A useful review should turn architecture concerns into decisions the team can actually execute.

| RISK | WHY IT MATTERS | RECOMMENDED ACTION |
| --- | --- | --- |
| **Change blast radius** | Deployments touch too many shared paths, so ordinary feature work carries production risk. | Map critical paths; isolate high-consequence dependencies before broad refactoring. |
| **Database ownership** | Multiple services can mutate the same data without clear contracts or recovery boundaries. | Assign ownership; define write paths; move toward explicit service/data boundaries. |
| **Retry / queue behavior** | Retries can amplify load or create duplicate side effects during partial failure. | Define idempotency, retry budgets, dead-letter handling and failure visibility. |
| **Observability** | Metrics show that something is wrong but cannot explain user impact or causal chain. | Instrument critical journeys, correlation IDs, SLOs and failure-mode-specific signals. |
| **AI runtime** | Model calls introduce nondeterminism, latency, cost and evaluation gaps. | Treat prompts/models/state/evals as versioned production dependencies with fallback paths. |

EXAMPLE DECISION

**Do not rewrite the platform. First isolate the two highest-risk paths, improve failure visibility, and make ownership explicit. Reassess broader decomposition after operational risk falls.**

SAMPLE ENGAGEMENT OUTPUT

# What the CTO Should Have at the End

The deliverable is a decision system: what is risky, what to change, what not to change, and in what order.

## CORE DELIVERABLES

| | | |
|---|---|---|
| 01 | **Executive risk summary** | The 3-5 issues most likely to create business-impacting failure. |
| 02 | **Architecture risk map** | Critical paths, failure domains, ownership gaps and hard-to-reverse decisions. |
| 03 | **Decision log** | Recommended architecture choices, alternatives considered and explicit trade-offs. |
| 04 | **Remediation sequence** | A practical order of operations that does not freeze product delivery. |
| 05 | **What not to rebuild** | Areas where redesign cost exceeds the risk it actually removes. |

## EXAMPLE EXECUTION SEQUENCE

NOW

### Contain

Reduce blast radius and make the critical path observable.

NEXT

### Clarify

Define ownership, contracts, recovery behavior and decision boundaries.

LATER

### Evolve

Refactor only where the risk reduction justifies migration cost.

TYPICAL STARTING ENGAGEMENT

**Architecture Risk & Production Readiness Review - focused on one important system, decision or failure mode, followed by a prioritized action plan.**

*Illustrative sample - not based on a specific client engagement.*