AI Evaluation in the SDLC

Acceptance Testing, Test Summary Reports,
and Governed Experimentation for AI Features

*A Joint Brief from the Director of Engineering
and the Director of Product*

Audience: Product Managers, Business Analysts, and AI Builders

Why This Document Exists

You are now writing code that reaches production. That is a real and meaningful shift in how our organization builds software. With that shift comes something we want to address directly: **the business owns the risk of what goes to production.** Not engineering. Not the AI tool. Not the model provider. The business. Every feature that reaches a customer is a promise made by this organization—and we are all accountable for delivering on that promise.

This brief establishes how we evaluate AI features as part of acceptance testing, how we document that evaluation through Test Summary Reports, and how we use that documentation to enable—not slow down—the kind of fast, governed experimentation that produces AI products worth using. Products that, in the language of Jobs to Be Done theory, are genuinely *hired* by users because they accomplish something meaningful—and are trusted by the organization to not create compliance, audit, or regulatory exposure.

> *The goal is not to add process. The goal is to make the process we already need so standardized and lightweight that governance becomes invisible infrastructure—like guardrails on a highway. You don't think about them. You just drive faster because they're there.*

AI Evaluation as Acceptance Testing

What AI Evaluation Is

AI evaluation is the discipline-specific extension of acceptance testing for features powered by AI models. In traditional software, acceptance testing asks: “Does this feature do what the requirements said it should?” For AI features, we need to ask additional questions that traditional acceptance testing was never designed to answer:

Does the AI produce outputs that are appropriate, safe, and aligned with our business intent—not just technically functional?

Does it fail gracefully when it encounters inputs it wasn’t designed for, or does it fail dangerously by producing confident but incorrect outputs?

Can we explain why it produced a given output well enough to satisfy an auditor, a regulator, or a customer who asks?

Will its behavior remain stable and appropriate over time, or will it drift as conditions change?

These are not theoretical concerns. They are the specific risks that differentiate AI features from traditional software—and they are the reason AI evaluation must be a first-class part of our acceptance testing process, not an afterthought.

What AI Evaluation Is Not

It is not a separate, parallel process that runs alongside the SDLC. It is acceptance testing, extended to cover AI-specific risks. It uses the same TSR structure, the same go/no-go gates, and the same governance framework as every other change to production.

It is not something only engineers can do. In fact, the most critical evaluators of AI features are the people closest to the business problem and the end user. You understand what “good” looks like for the customer. You know when an AI output would be embarrassing, misleading, or harmful in context. Engineering can tell you whether the system is technically sound. You can tell us whether it’s actually delivering on the job it was hired to do.

It is not a one-time gate. AI features require ongoing evaluation through production monitoring. The model’s behavior can change as the world changes around it—new data patterns, shifting user behavior, upstream model updates. Acceptance testing for AI is the beginning of evaluation, not the end.

The Error Analysis Improvement Cycle

The core test strategy for AI features follows a structured cycle that is specific to AI and should be documented in every TSR. This cycle is how we move from “the AI sort of works” to “the AI reliably delivers on the job to be done.”

Step 1: Error Analysis to Identify Failure Modes

Before we can improve an AI feature, we need to systematically understand *how* it fails. Error analysis is the practice of collecting and categorizing AI outputs that did not meet the acceptance criteria, then identifying patterns in those failures. This is not “does it work or not”—it is “in what specific ways does it fail, and what are the underlying causes?”

Common failure modes for AI features include:

Failure Mode	What It Looks Like	Root Cause Category
Hallucination	AI generates plausible but fabricated information	Model knowledge gaps; insufficient grounding
Scope Violation	AI provides advice or actions outside its authorized domain	Prompt boundaries too loose; missing guardrails
Tone / Register Mismatch	AI response is too casual, too formal, or inappropriate for context	Prompt engineering; system prompt misalignment
Edge Case Failure	AI breaks on unusual but valid inputs (special characters, long text, empty fields)	Insufficient input validation; training data gaps
Inconsistency	AI gives different answers to the same question across sessions	Non-determinism; temperature settings; context window effects
Bias / Fairness Violation	AI outputs systematically disadvantage a group	Training data bias; prompt framing; evaluation blind spots
Performance Degradation	AI responses slow or fail under load	Infrastructure sizing; model serving bottlenecks; rate limiting
Drift	AI behavior changes over time without code changes	Foundation model updates; data distribution shifts; upstream changes

Step 2: Prompt and Architecture Changes

Once failure modes are identified, the team implements targeted changes. This is where AI development differs most from traditional software: many fixes involve **prompt engineering** (changing the instructions to the model), **retrieval architecture changes** (improving what context the model receives), or **guardrail additions** (adding filters, validators, or fallback paths)—rather than traditional code changes.

Every change, whether to a prompt, a retrieval pipeline, a guardrail, or traditional code, goes through the same SDLC. The TSR documents what changed, why, and how the change was validated. This is critical because prompt changes can have non-obvious cascading effects that are invisible without structured testing.

Step 3: Automated Acceptance Tests for AI Features

This is where the cycle becomes sustainable and scalable. The failure modes discovered in Step 1, and the fixes implemented in Step 2, become the basis for **automated acceptance tests** that run on every subsequent change. These tests are the permanent institutional memory of what we've learned about how this AI feature can fail.

Automated AI acceptance tests look different from traditional tests. Because AI outputs are non-deterministic, tests validate *behavioral boundaries* rather than exact outputs. Examples:

What We Test	How We Assert	Why It Matters
Output stays in scope	Response does not contain financial advice, medical guidance, or other prohibited domains	Prevents scope violations that create liability
Factual grounding	Claims in the response can be traced to retrieved source documents	Detects hallucination before users encounter it
Tone appropriateness	Response sentiment and formality fall within defined thresholds for the use case	Protects brand and user trust
Graceful degradation	When retrieval returns no results, the AI acknowledges uncertainty rather than fabricating	Prevents confident wrong answers
Input robustness	Adversarial, empty, and edge-case inputs produce safe, appropriate responses	Hardens the system against misuse and unexpected inputs
Consistency	Same question asked 10 times produces substantively consistent answers	Builds user trust; surfaces non-determinism issues
Performance SLA	P95 response time under load stays below defined threshold	Ensures production viability under real conditions

> *The error analysis → fix → automate cycle is how AI features get better over time without requiring the team to re-evaluate everything manually. Each cycle adds to the automated test suite, which means each deployment gets faster and more confident—not slower.*

The Test Summary Report for AI Features

The TSR is the single artifact that ties everything together. It is how the team communicates to the business: here is what we built, here is how we tested it, here is what we found, and here is our recommendation on whether to promote it to production. For AI features, the TSR includes sections that are specific to the risks AI introduces.

TSR Structure: Tailored to Risk

Not every change needs the same level of documentation. The TSR structure is calibrated to the risk level of the application and the change being made. A prompt refinement on a low-risk internal tool does not need the same TSR as a new AI feature handling customer-facing financial data.

Risk Tier	Examples	TSR Depth	Review Required
Tier 1: Low	Internal tools, productivity aids, non-customer-facing summaries	Lightweight: scope, test results, known limitations, go/no-go	Tech lead sign-off
Tier 2: Medium	Customer-facing but non-decisional (search, recommendations, drafting assistance)	Standard: full error analysis, failure mode inventory, automated test results, monitoring plan	Tech lead + product owner
Tier 3: High	Financial decisions, compliance workflows, customer data processing, regulated domains	Comprehensive: all Standard items + fairness assessment, explainability documentation, adversarial testing results, model provenance, regulatory mapping	Tech lead + product owner + compliance/governance

This tiering is how we avoid undue burden. A Tier 1 TSR can be completed in thirty minutes. A Tier 3 TSR takes more effort—but the effort is proportional to the risk, and the structure means the team knows exactly what to produce rather than guessing. Standardization is what makes this fast: when everyone uses the same template, reviewers know where to look, authors know what to write, and auditors know what to expect.

What a TSR Contains

Every AI feature TSR, regardless of tier, includes these sections:

1. Change Summary

What changed, why, and what job-to-be-done it serves. Written in business language, not technical jargon. The audience is a product owner or business stakeholder who needs to understand the intent and scope without reading the code.

2. Test Strategy

The specific approach taken to evaluate this change. For AI features, this describes the error analysis methodology: what inputs were tested, what failure modes were targeted, and what success criteria were used. This section documents the test *thinking*, not just the test results.

3. Error Analysis Results

A structured accounting of failure modes discovered, categorized by type and severity. For each failure mode: what it looks like, how often it occurs, what the business impact would be, and what mitigation was applied. This section is what makes the TSR defensible—it shows we didn't just test the happy path.

4. Changes Made

What prompt changes, architecture changes, guardrails, or code modifications were implemented in response to the error analysis. Each change links back to a specific failure mode. This traceability is what auditors and regulators need to see: that every change has a documented rationale.

5. Automated Test Results

The output of the automated acceptance test suite. Pass/fail status, coverage metrics, and any new tests added during this cycle. Over time, this section becomes the living record of everything the team has learned about how this feature can fail.

6. Production Monitoring Plan

What will be monitored post-deployment, what thresholds will trigger alerts, and what the rollback criteria are. For AI features, this includes model drift detection, output quality sampling, and defined escalation paths if the AI's behavior degrades. This is not optional—it is part of acceptance.

7. Go/No-Go Recommendation

The team's explicit recommendation with supporting rationale. If the recommendation is "go," it includes known residual risks and the monitoring plan that addresses them. If "no-go," it includes what additional work is needed and estimated timelines.

Example: TSR for a Production Modification

To make this concrete, here is what a Tier 2 TSR looks like for a modification to an AI feature already in production. This is a real-world scenario: a customer-facing document summarization feature that has been receiving user complaints about accuracy.

TEST SUMMARY REPORT

Feature: Document Summarization Assistant
Change Type: Production modification — Prompt refinement + retrieval pipeline adjustment
Risk Tier: Tier 2 (Customer-facing, non-decisional)
Author: J. Rivera (AI Builder, Product Team) + M. Chen (Tech Lead)
Reviewers: Tech Lead sign-off + Product Owner approval

1. CHANGE SUMMARY

Users reported that document summaries occasionally included details not present in the source document (hallucination) and sometimes missed key contractual terms. This modification addresses both failure modes through prompt refinement to enforce source-grounded summarization and a retrieval pipeline adjustment to improve chunk relevance scoring. The job to be done: users hire this feature to quickly understand the key obligations in a document without reading the entire thing. It must be trustworthy enough that they can act on the summary.

2. TEST STRATEGY

Error analysis methodology: Collected 47 user-flagged inaccurate summaries from the past 30 days. Categorized each by failure mode. Ran the same documents through the modified prompt on a sample of 200 documents spanning contract types (NDAs, SOWs, MSAs, amendments). Success criteria: zero hallucinated facts in test corpus; key terms extraction rate above 92%; response time under 4 seconds at P95.

3. ERROR ANALYSIS RESULTS

Failure modes identified: (a) Hallucination — 12 of 47 flagged summaries contained fabricated clauses, root cause traced to the model filling gaps when retrieved chunks were too short. Severity: High. (b) Key term omission — 28 of 47 flagged summaries missed termination clauses or liability caps, root cause traced to chunk retrieval ranking deprioritizing sections near document end. Severity: Medium. (c) Tone inconsistency — 7 of 47 flagged summaries used informal language for formal legal documents. Severity: Low. Mitigation applied to all three.

4. CHANGES MADE

(a) Prompt change: Added explicit instruction to only reference information present in provided context; added instruction to state ‘Not addressed in document’ when information is absent rather than inferring. Links to Hallucination failure mode. (b) Retrieval pipeline: Increased chunk overlap from 50 to 150 tokens and added positional weighting to boost chunks from document beginning and end sections. Links to Key Term Omission failure mode. (c) Prompt change: Added tone instruction specifying formal register for legal documents, matched to document type metadata. Links to Tone Inconsistency failure mode.

5. AUTOMATED TEST RESULTS

Full suite: 34 existing tests + 8 new tests added this cycle. Results: 42/42 passing. New tests added: 3 hallucination-specific (verify no claims without source attribution), 3 key-term extraction (verify termination, liability, and payment terms captured for each contract type), 2 tone-register (verify formal language for legal documents). Coverage: All three failure modes now have automated regression tests. Consistency check: Same 10-document set run 5 times, substantive consistency score: 96%.

6. PRODUCTION MONITORING PLAN

Metrics monitored: User-flagged inaccuracy rate (baseline: 8.2%, target: below 3%), key term extraction spot-check accuracy (weekly sample of 50 summaries reviewed by legal ops), response time P95, error rate. Alert thresholds: Inaccuracy flags exceed 5% in any 7-day window triggers review. Response time exceeds 6s at P95 triggers infrastructure review. Rollback criteria: If inaccuracy rate exceeds 8% (current baseline) within 14 days, automatic rollback to previous prompt version. Observation window: 21 days before declaring stable.

7. GO/NO-GO RECOMMENDATION

GO. All automated tests passing. Error analysis demonstrates targeted mitigation of identified failure modes. Residual risks: Non-determinism means isolated inaccuracies remain possible (mitigated by monitoring plan and user feedback mechanism). Key term extraction improvement is validated on common contract types but has not been tested on highly unusual document structures (mitigated by the ‘Not addressed in document’ fallback behavior). Production monitoring plan is in place with defined rollback criteria.

Notice what this TSR accomplishes: A product team member co-authored it. It took approximately 90 minutes to prepare. It is readable by a business stakeholder. It is defensible in an audit. And it documents a clear, repeatable cycle of improvement that makes the next TSR faster, not slower.

How Product and Business Analysts Participate

Here is the collaboration model. Each role contributes what they are best positioned to provide, and the TSR structure ensures nothing falls through the cracks.

Activity	You (PM / BA)	Engineering	Together
Define the job to be done	Own: What problem does this solve? What does 'good' look like?	Advise: What's technically feasible? What are the constraints?	Align: Acceptance criteria that are testable and meaningful
Error analysis evaluation	Own: Is this output appropriate for the user and the context?	Own: Why did the model produce this? What technical root cause?	Categorize: Failure mode inventory with business impact ratings
Prompt and architecture changes	Advise: Does the new behavior match intent?	Own: Implement prompt, retrieval, or guardrail changes	Validate: Test the fix against the failure mode that triggered it
Write the TSR	Co-author: Change summary, job-to-be-done framing, go/no-go input	Co-author: Test strategy, technical changes, automated test results	Review: Does this TSR tell a complete, defensible story?
Production monitoring	Own: Define what 'working' looks like from the user's perspective	Own: Instrument monitoring, set up alerts, define rollback triggers	Decide: Based on monitoring data, is this feature delivering on its job?
Communicate to the business	Own: Translate TSR findings into business language for stakeholders	Support: Provide technical context when needed	Present: Joint recommendation backed by documented evidence

Why the Business Should Trust This Process

When a product manager or business analyst brings a TSR to a business stakeholder and recommends deploying an AI feature, the TSR answers every question the business should be asking:

What does this feature do? The change summary, written in business language.

How do we know it works? The error analysis and automated test results.

What can go wrong? The failure mode inventory with documented mitigations.

What happens if it stops working? The production monitoring plan with rollback criteria.

Are we exposed to compliance or regulatory risk? The risk tier determination and, for Tier 3, the explicit regulatory mapping.

Can we prove we did our due diligence? The entire TSR, as a dated, authored, reviewed artifact.

This is what it means for the business to own the risk responsibly. Not by avoiding AI, but by deploying it with documented evidence that the organization understood the risks, tested for them, mitigated them, and has a plan for when reality departs from expectations.

Governance Without Undue Burden

The number one objection we hear is: “This sounds like a lot of documentation.” Here is how we keep it lightweight:

Standardization Is Speed

The TSR template is the same every time. You are not inventing a new document with every deployment. You are filling in a known structure with the specifics of this change. When the structure is standardized, the cognitive load drops dramatically—for the author, the reviewer, and the auditor. The Qodex research on test report best practices confirms this: standard templates make reports faster to write, faster to review, and more consistent across teams.

The Template Does the Thinking for You

Each section of the TSR has a specific purpose. You do not need to decide what to include—the template tells you. For a Tier 1 change, you fill in four sections and you are done. For Tier 2, seven sections. The tiering decision itself is guided by a simple risk matrix. This is by design: the IAA and FINOS frameworks both emphasize that governance overhead should be proportional to risk, and our tier structure implements that principle.

Automation Reduces the Manual Burden Over Time

Every cycle through the error analysis → fix → automate loop adds tests to the automated suite. This means the test results section of the TSR is increasingly generated, not written. The monitoring plan, once established, carries forward. The change summary and error analysis are the only sections that require fresh writing—and those are the sections where human judgment adds the most value.

TSRs Are Living Documents, Not Tombstones

For features with ongoing modifications, the TSR is appended, not rewritten. Each modification adds a dated entry documenting the change, the error analysis that prompted it, and the test results that validated it. Over time, the TSR becomes a concise history of the feature’s evolution—valuable for onboarding new team members, conducting periodic reviews, and responding to audit inquiries. The DoD AIES Guidebook calls these “model cards”—we call them TSRs, but the principle is the same: a living artifact that travels with the feature.

Production Monitoring Generates Its Own Evidence

The monitoring plan defined in the TSR produces dashboards, alerts, and trend data that become the basis for the next TSR’s error analysis. This creates a virtuous cycle: production monitoring generates the data that informs testing, testing generates the automated checks that improve monitoring, and the TSR captures both. The Alation governance framework emphasizes exactly this: governance is a “system for improvement, not just a gate.”

FROM BOTH OF US

We want you building with AI. We want the experiments. We want the features that genuinely solve problems for users and create value for the organization. The SDLC, the TSR, and the governance framework described in this brief are not obstacles to that goal—they are the infrastructure that makes it possible to experiment at the pace that matters while maintaining the trust that the business, the regulators, and our customers require.

The products that truly matter—the ones that users hire to do a real job, the ones that organizations trust with real decisions—are the ones that were built with the discipline to be trustworthy. This framework is how we build that discipline together.

Build boldly. Document clearly. Ship with confidence.