ML LABS INTELLIGENCE

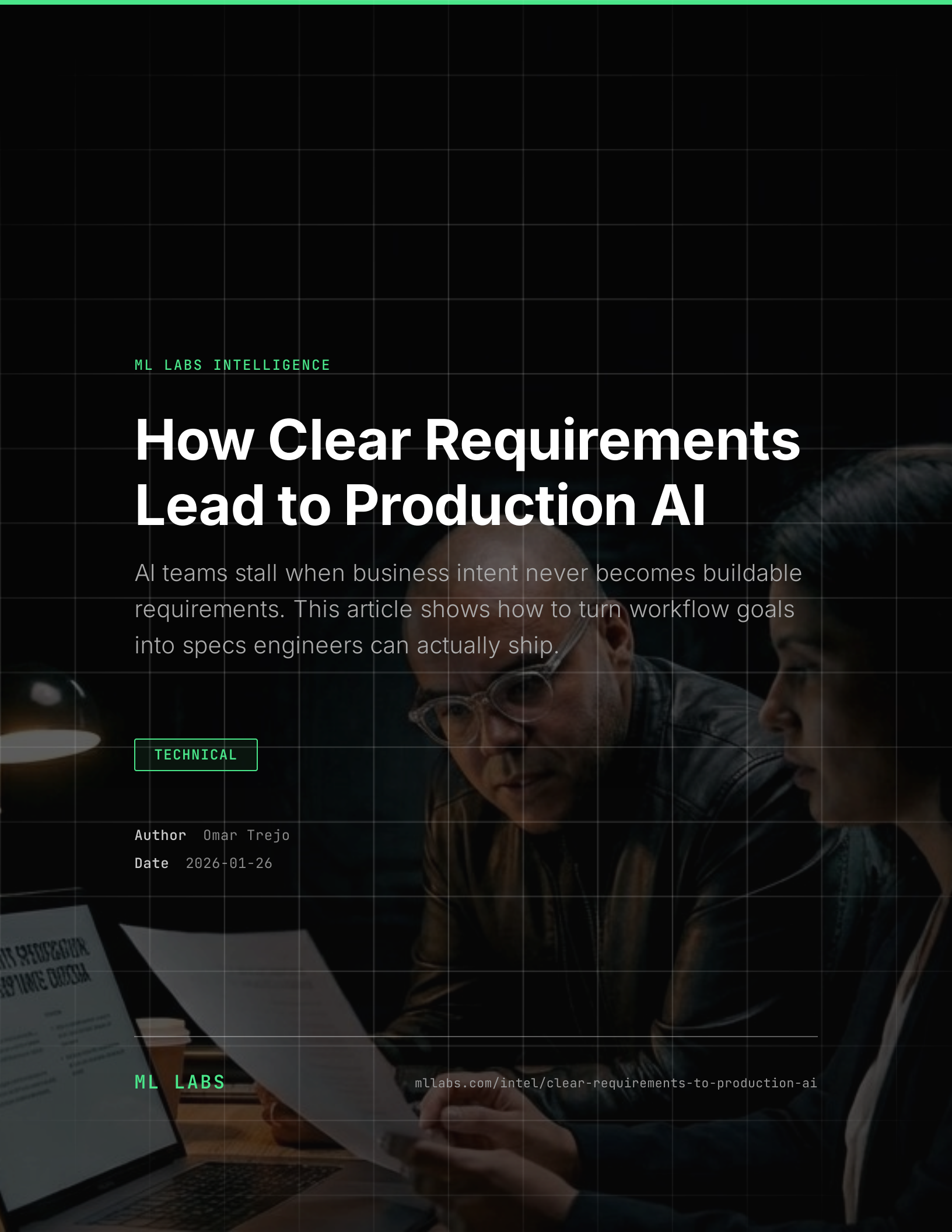
How Clear Requirements Lead to Production AI

AI teams stall when business intent never becomes buildable requirements. This article shows how to turn workflow goals into specs engineers can actually ship.

TECHNICAL

Author Omar Trejo

Date 2026-01-26



You've defined the business problem. Inputs are available, outputs are needed, success metrics are quantified, budget is allocated. Now comes the hard part: translating business requirements into technical specifications that an AI team can actually build against.

A **systematic mapping study on requirements engineering for AI systems** (Ahmad et al., 2022) found that requirements gaps — not missing requirements, but requirements that were clear in business terms and ambiguous in technical terms — are among the top drivers of AI project failure.

The Translation Problem

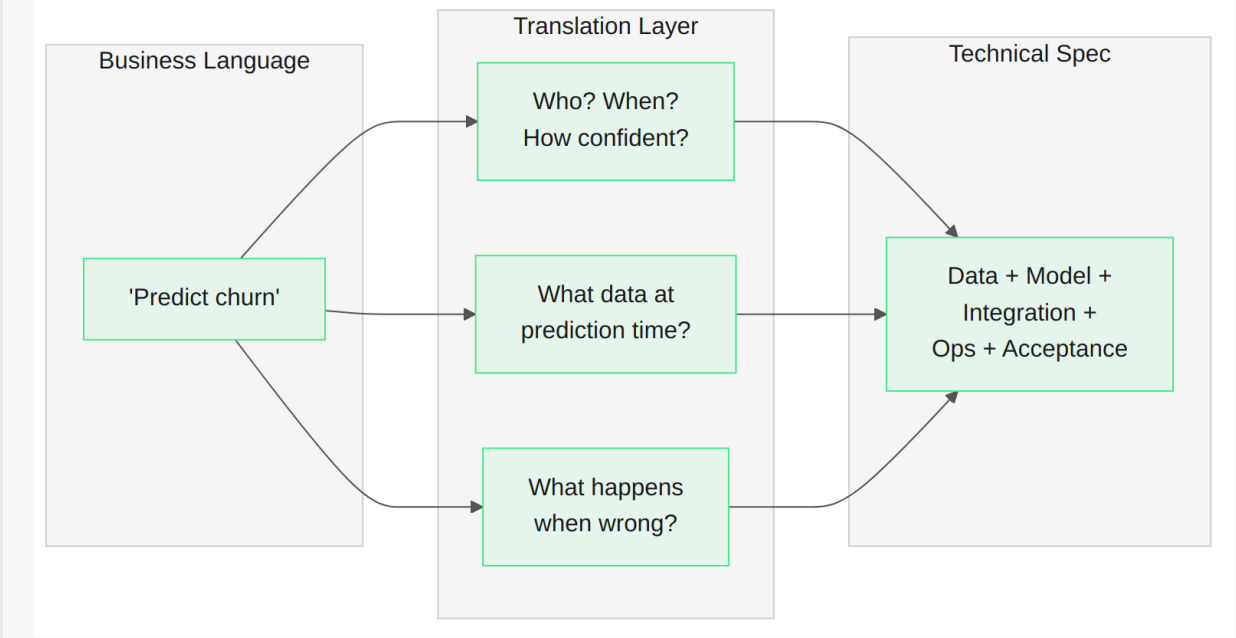
Business stakeholders speak in outcomes: "predict which customers will churn," "automate invoice processing," "detect fraudulent transactions." Each hides dozens of technical decisions that fundamentally change the system's design, cost, and scope. "Predict which customers will churn" raises immediate questions: How far in advance? With what confidence threshold? What data is available at prediction time? What happens with the prediction?

Traditional software requirements assume deterministic behavior: given input X, produce output Y. AI systems are probabilistic: given input X, produce output Y with confidence Z, and sometimes it's wrong. The **An analysis of AI project failures** (RAND, 2024) shows that misaligned requirements — not technical limitations — account for the majority of failed initiatives.

> **The system works as specified, but the specification didn't capture what stakeholders actually needed. That deterministic-probabilistic mismatch is where most AI project disappointment originates.**

The Requirements-to-Deployment Pipeline

This pipeline structures the translation from business requirements through technical specification to deployed system.



Step 1: Formalize Business Requirements

For each AI capability, document the prediction (what the system produces), the operational context (throughput, latency, who consumes outputs), and the error model (relative cost of false positives versus false negatives, autonomous versus human-reviewed).

Example translation for invoice processing:

- **Accuracy**: "Get invoices right" becomes "extract 12 fields at 95%+ field-level accuracy, measured weekly against human audit"

- **Volume and speed**: "Handle invoices quickly" becomes "500/day sustained, peak 200/hour, under 30 seconds each"
- **Errors**: "Flag problems" becomes "route extractions below 80% confidence to a human review queue with SLA tracking"

Step 2: Data Definition Document

Poorly defined features (Google, 2024) are a significant source of ML system bugs. For each input, create a data definition that specifies source, schema, and quality constraints.

- **Source and access**: where the data comes from, how the AI system reads it, and who owns the pipeline that delivers it
- **Schema and quality**: field names, types, constraints, known issues, training volume, and how often the schema changes
- **Features explicitly defined**: "average monthly spend over last 6 months" is a feature; "customer data" is not

Step 3: Model Design Specification

Specify the class of approach — supervised classification, regression, sequence-to-sequence, retrieval-augmented generation — not the specific model, which is determined through experimentation.

Cover training data selection (which records, time period, filtering criteria), validation strategy (data splits, cross-validation), and evaluation metrics with a clear primary metric. Anchor to baselines: current performance without AI, minimum acceptable for deployment, and target for success.

Step 4: Integration Design

Integration failures are among the most common causes of delay in AI deployments — the surface area of dependencies, auth flows, versioning contracts, and error handling is large.

- **Input and output interfaces**: API spec, batch format, or event stream schema with latency requirements
- **Authentication and versioning**: how the system authenticates and how model versions are tracked
- **Error responses**: what happens on timeout, low confidence, or missing data — and who gets notified when it does

Step 5: Operations Design

Based on **research on scaling AI in organizations** (MIT Sloan, 2019), every production AI system needs monitoring, alerting, retraining, and rollback procedures defined before deployment.

- **Monitoring and alerting**: accuracy, latency, throughput, and error rate tracked daily with automated threshold alerts
- **Retraining triggers**: calendar-based, performance-based, or data-drift-based — with ownership and approval gates defined
- **Rollback procedure**: documented steps to revert to the previous model version within minutes

Step 6: Acceptance Criteria

Acceptance criteria must be measurable and agreed upon by both business and technical stakeholders before development starts.

- **Functional tests**: process representative samples with known correct answers and verify edge case handling
- **Performance tests**: sustain target throughput under simulated production load and verify latency SLA at P95
- **Operational tests**: verify monitoring accuracy, alert correctness, and rollback completion within five minutes

Boundary Conditions

This approach assumes that stakeholder goals can be made concrete and measurable. When business objectives remain genuinely ambiguous — "make the customer experience better" without defined metrics, or competing stakeholders with contradictory success criteria — the translation pipeline produces specifications that encode the wrong targets.

When you encounter persistent ambiguity, pause and resolve outcome ownership first. Get a single decision-maker to define success in measurable terms with explicit tradeoffs (e.g., "we'd rather miss 10% of fraud than flag 5% of legitimate transactions"). Only after that alignment exists does the pipeline produce specifications worth building against.

First Steps

1. **Formalize business requirements.** Run them through the Step 1 template. Resolve every ambiguity with stakeholders before writing code.
2. **Map the data landscape.** Define source, schema, and quality for every input. This alone often reveals hidden gaps.

3. **Define acceptance criteria.** Write them before development starts. If you cannot describe "done" in measurable terms, you are not ready to build.

Practical Solution Pattern

Adopt a six-stage translation pipeline that converts business requirements into production contracts before a single line of code is written: formalized objective, data definition, model design, integration design, operations design, and signed acceptance criteria — in that order, with business and technical stakeholders aligned at each step.

This works because requirements gaps, not technical limitations, are the dominant cause of AI project failure. The pipeline forces ambiguities to surface when they are cheap to resolve rather than after significant development investment. If one workflow is already defined and needs to move from requirements into software, **AI Workflow Integration** is the direct build path.

References

1. Ahmad, K., Abdelrazek, M., Arora, C., Bano, M., & Grundy, J. **A Systematic Mapping Study on Requirements Engineering for AI-Intensive Systems**. arXiv, 2022.
2. RAND Corporation. **Analysis of AI Project Failures**. *RAND Corporation*, 2024.
3. Google. **Rules of Machine Learning: Best Practices for ML Engineering**. *Google Developers*, 2024.
4. Ransbotham, S., et al. **Winning with AI**. *MIT Sloan Management Review*, 2019.