Managing AI Software Development [1]

Yogi Schulz

Managing AI software development differs materially from conventional, procedural custom application delivery due to probabilistic behavior, data dependencies, and continuous learning dynamics. Effective project management requires adapting planning, governance, engineering discipline, and operational practices to the characteristics of AI applications.

For an AI software development project to succeed, the project manager ensures that the project team prominently adds the following concepts in the project charter and the project management plan, and that they receive ongoing attention as the project progresses.

Treat data as a critical asset

Traditional software development is more design and procedure-focused and often insufficiently data-focused. AI systems are fundamentally data-driven. AI project success is often constrained more by data quality than code quality or architecture.

Treat data as a critical asset with:

- Data governance: Establish clear ownership, lineage tracking, and access controls.
- Data quality management: Implement data validation pipelines to detect schema drift, missing values, and bias.
- Data versioning: Use tools to version datasets alongside model versions to ensure reproducibility of results.
- Data labeling: Define consistent annotation guidelines for your annotators and measure the level of inter-annotator agreement.

Incomplete or haphazard data management leads to more hallucinations, undermining trust in the results. Without at least disciplined and perhaps intense data management, downstream AI model performance becomes unpredictable and difficult to debug.

High data quality ensures project success and acceptance of the AI system.

---

[1] How to cite this article: Schulz, Y. (2026). Managing AI Software Development, *PM World Journal*, Vol. XV, Issue V, May.

Adopt MLOps as the operating model

Traditional development operations (DevOps) practices, such as automated builds and tests, continuous testing and feedback, and continuous integration, are insufficient for AI systems. Machine language operations (MLOps[2]) extend development practices to include model lifecycle management.

Adopt these MLOps components beyond DevOps:

- Pipeline automation: Build continuous integration and continuous delivery/deployment (CI/CD) pipelines that include data ingestion, training, evaluation, and deployment.
- Model registry: Maintain a centralized repository of approved models with a metadata catalogue that includes performance metrics, training data, and hyperparameters.
- Reproducibility: Ensure that experiments can be rerun deterministically using versioned code, data, and configurations.
- Continuous model training: Automate retraining workflows triggered by data drift or performance degradation.

Missing MLOps components increase development effort through rework, lengthening development cycles.

This MLOps environment automates the build, test, and deployment stages. The benefits include teams reducing manual errors, speeding up development cycles, and improving code quality.

Focus on repeated business value delivery

Traditional software development tends to be organized around too much foundational work before end users and project sponsors see functionality they can actually use. This approach risks losing political support for the project well before completion. AI software development faces an even greater risk of losing political support because it requires more foundational work.

[2] MLOps is a set of practices, culture, and tools aimed at automating and streamlining the entire lifecycle of software development that includes machine learning models.

A better approach to AI software development is to plan for many software releases. Each release will include a small amount of:

- Foundational work.
- Functionality that end users and project sponsors can actually use.

AI project teams should plan every AI software release to include these ideas:

- Start with simpler AI models and progress to more complex architectures in successive releases.
- Tune the AI models based on the experience gained during the evaluation of results in the previous release.
- Add a new data source and cleansed data to every release.
- Ensure each release delivers measurable business value.
- Terminate work on releases that do not demonstrate the planned return on investment (ROI) and move on to the next release.
- Avoid overengineering a release.

Incorporate security and privacy

AI systems often process sensitive data, so data security and privacy cannot be an afterthought, as they are too often in traditional software development. AI project teams need to incorporate security and privacy in their AI system design.

Security and privacy design components include:

- Data anonymization: Apply techniques such as tokenization or differential privacy. Anonymization may be more extensive in the test region than in production.
- Access control: Restrict AI model and data access based on end-user roles.
- Model security: Protect against adversarial attacks and model extraction risks.
- Suspicious events: Log and report suspicious events.

Additional considerations:

- The scope of security features must extend across both data and model layers.
- An excellent security and privacy design is insufficient. Security and privacy must be monitored during AI application operation and tuned as adversarial attacks evolve.
- Access control must be monitored to reduce over-provisioning that tends to occur over time.

Inadequate security and privacy measures lead to data breaches and unpleasant noncompliance findings, resulting in reputational damage and fines.

Manage AI model risk and bias

AI software development introduces new categories of operational and reputational project risks that conventional software development has never had to address.

Manage these categories of AI model risk:

- Bias detection: Regularly audit AI models for demographic or systemic bias.
- Explainability: Use interpretable models or explanation techniques such as SHapley Additive exPlanations (SHAP[3]) or Local Interpretable Model-agnostic Explanations (LIME[4]) where required.
- Compliance: Align the AI software design with regulatory frameworks. Typical examples include privacy laws, data retention regulations and emerging AI governance standards.
- Human oversight: Build and operate human-in-the-loop mechanisms for high-stakes decisions.

Uneven or incomplete project risk management is a common source of failure in AI application development projects.

Effective project risk management begins in design and continues through to deployment. Comprehensive project risk management contributes significantly to project success. Operational risk management is an ongoing feature of AI system operation.

Emphasize cross-functional collaboration

AI projects require greater coordination across multiple disciplines than traditional software development.

The cross-functional collaboration that occurs in AI software development includes these disciplines:

---

[3] SHAP values are a way to explain the output of any machine learning model. Each AI model feature is assigned an importance value representing its contribution to the model's output.

[4] LIME aims to provide clear, human-understandable explanations for decisions generated by AI models.

- Data engineering: Design, build and test data pipelines.
- Data science: Design, develop and test the AI models.
- Data stewards: Responsible for achieving and maintaining data quality.
- Software engineering: Design, build and data integration routines and APIs
- Domain experts: Provide business context and validation for designs.

Siloed disciplines significantly increase data and software integration friction and project failure rates.

Best practices for cross-functional collaboration include:

- Shared artifacts: Maintain a single, version-controlled set of documents.
- Iterative delivery: Use agile methodologies with short feedback cycles.
- Common terminology: Align stakeholders on definitions, metrics, and expectations.

Define clear evaluation metrics

Unlike traditional deterministic software development, AI systems require statistical performance assessment.

Define evaluation metrics from these categories:

- Task-specific metrics: Use F1 score[5] for precision and recall, Receiver Operating Characteristic Area Under the Curve (ROC-AUC[6]), or domain-specific KPIs.
- Business alignment: Translate model metrics into business impacts such as cost savings, margin improvement and risk reduction.
- Baseline comparisons: Benchmark model versions against simple models or heuristics to validate incremental value.
- Robust testing: Expand the set of planned software test cases to include edge cases, adversarial inputs, and out-of-distribution scenarios.

Ambiguous or misaligned metrics are a common source of failure in adopting an AI application or a development project.

---

[5] The F1 score is a machine learning evaluation metric that represents the harmonic mean of a model's precision and recall, providing a balanced measure of accuracy for classification models.

[6] ROC AUC measures a binary classifier's ability to distinguish between classes across all thresholds. It represents the probability that a model ranks a random positive example higher than a negative one.

Projects that regularly evaluate metrics spot problems early and take corrective action before the related rework becomes a major source of budget and schedule variance.

Implement robust iterative practices

AI software development is inherently iterative or even experimental. That's rarely the case for traditional software development. The Agile methodology applies to both types of development work.

Robust iterative practices consist of the following:

- Experiment tracking: Log model versions, hyperparameters, datasets, software versions and results systematically.
- Controlled comparisons: Use A/B testing or shadow deployments in production.
- Statistical rigor: Avoid overfitting to validation sets; use cross-validation where appropriate.

When the pressure of a project's schedule or budget forces the team to cut corners on these practices, the AI system's output often disappoints in production.

These practices ensure that improvements delivered in each release are genuine and reproducible.

Conclusion

Managing AI software development requires integrating data discipline, MLOps practices, and risk governance into a cohesive project management plan. Organizations that treat AI as an engineering system—rather than a purely experimental activity—achieve more consistent, scalable outcomes. The distinguishing factor is not model sophistication, but project management and AI operational maturity across the lifecycle.

About the Author

Yogi Schulz

Calgary, Alberta, Canada

Yogi Schulz has over 40 years of experience in Information Technology across various industries. Yogi works extensively in the petroleum industry, selecting and implementing financial, production revenue accounting, land & contracts, and geotechnical systems. He manages projects arising from changes in business requirements, leveraging technology opportunities, and mergers and acquisitions. His specialties include IT strategy, web strategy, and systems project management.

Mr. Schulz regularly speaks to industry groups and writes a regular column for Engineering.com and for TechNewsDay.com. He has written for Microsoft.com and the Calgary Herald. His writing focuses on project management and IT developments of interest to management. Mr. Schulz served on the Board of Directors of the PPDM Association for 20 years, until 2015. He can be contacted at yogischulz@corvelle.com

His new book, co-authored by Jocelyn Schulz Lapointe, is "A Project Sponsor's Warp-Speed Guide: Improving Project Performance."