



Engineering the Next Architecture for Human-Agent Collaboration

Ayşe Kok Arslan

Oxford Alumni- Silicon Valley Chapter, United States

DOI: <https://doi.org/10.68050/JAMS.2026.801>

Received: 02 October 2026; Accepted: 10 October 2026; Published: 17 October 2026

ABSTRACT

This paper examines how an engineering-led approach to information architecture, sprint planning, and quick-start flows can materially improve developer productivity and reliability outcomes. In the initial state, a fragmented and weakly structured knowledge base required AI developers to infer platform behavior from tribal knowledge, ad hoc Slack threads, and outdated implementation guides, increasing onboarding time and error rates. This paper describes a re-design in which documentation is treated as a system with an explicit architecture—comprising Quick Start, Introduction, Environment Setup, First Use Case, Core Capabilities, Observability, and Special Topics—mirroring established systems-engineering practices for modularization and interface definition. By applying classic systems engineering principles to knowledge flows, the team transformed documentation from a passive artifact into an actively engineered component of the developer platform, yielding more predictable first-success paths, reduced support load, and a foundation for future automation and AI-assisted guidance.

Keywords: Agentic AI, Information Architecture, Human-AI Interface, Systems Engineering, Developer Experience, AI Workflows.

1. INTRODUCTION

Designing agentic AI workflows and their documentation through an engineering mindset requires treating “docs” as an executable part of the system, not a marketing afterthought.

The landscape of Artificial Intelligence has undergone a rapid transformation with the advancement of Large Language Models (LLMs) and Vision-Language Models (VLMs). Historically, human-AI interaction followed a linear pattern where a human provided a prompt and the model generated a response. However, a new paradigm has emerged: Agentic AI.

Agentic AI involves software programs that use LLMs as reasoning engines to construct prompts, call external tools, interpret data, and perform follow-up actions without direct human intervention.

The next chapter explores the theories underpinning the challenges of communication and collaboration between different agents.

2. THEORETICAL BACKGROUND

A core theoretical contribution of modern AI engineering is the recasting of documentation from static collateral to executable infrastructure. Traditionally, documentation was a “marketing afterthought,” but for production-grade agentic systems, it must be treated as a first-class engineered system.

The literature suggests that the boundary between system design and documentation design in AI platforms is porous. A production-grade agentic workflow is not merely a collection of scripts but a first-class engineered system where the documentation functions as the control surface. By combining a structured engineering lifecycle, deterministic execution patterns, multi-model reasoning for governance, and containerized deployment, organizations can transition from fragile prototypes to robust, extensible, and trustworthy agentic AI systems.



While prototypes are often easily built in notebooks, scaling agentic AI into production environments introduces significant engineering complexities.

- **Design Challenges:** Developers must determine how to decompose complex tasks into individual agents and choose between different integration protocols, such as the Model Context Protocol (MCP) or direct tool calls.
- **Implementation Risks:** Managing multi-agent communication and maintaining prompt modularity often leads to LLM drift or unpredictable execution paths.
- **Operational Requirements:** Systems must be reliable, governed, and observable, requiring robust handling of concurrency, retries, and cost efficiency.
- **Deployment Hurdles:** Moving from a script to a system involves containerization, versioning, and continuous delivery of both code and prompts.

Without a disciplined engineering approach, these systems often become brittle and opaque "tribal knowledge" constructs that are difficult to debug or scale.

Individual LLMs are prone to hallucinations, reasoning inconsistencies, and bias. To address this, the literature proposes a Model Consortium Architecture—an ensemble of specialized LLMs (e.g., Gemini, GPT, Claude, Llama) that generate independent outputs.

When multiple specialized agents collaborate—handling roles such as searching, filtering, reasoning, and publishing—they form agentic AI workflows. These workflows represent a fundamental shift toward autonomous systems capable of reasoning, planning, and executing multi-step tasks. Unlike monolithic prompts, these systems are modular and extensible pipelines that bridge web retrieval, content generation, and software operations.

Moreover, engineering principles such as the Single-Responsibility Principle are critical for agentic workflows. Overloading a single agent with multiple tools or responsibilities increases prompt complexity and the likelihood of incorrect execution.

Assigning a single, well-defined tool to each agent creates predictable roles and simplifies prompting. For example, a workflow that separates "content extraction" from "content publishing" ensures that agents do not skip steps or invoke tools in the wrong order.

A robust theoretical pattern involves splitting conceptual tasks. For multimodal generation, one agent might build a structured JSON "plan," while a non-agent function handles the API "execution". This prevents the LLM from hallucinating status messages or file paths for actions it has not actually performed.

A recurring theme in the literature is the tension between standardized protocols and system stability. The Model Context Protocol (MCP) provides a standardized mechanism for agents to communicate with external services, but it often introduces layers of abstraction that reduce determinism.

In production environments, reliance on MCP can lead to "flickering" failures and ambiguous tool-selection behaviors.

- **Tool Calls over MCP:** For critical steps, replacing MCP with direct tool calls improves stability and reduces the "cognitive load" on the agent by eliminating the need to interpret protocol metadata.
- **Pure Functions over Tool Reasoning:** For operations not requiring language-driven reasoning (e.g., database writes, API posts), pure functions are superior. These functions are deterministic, faster, and cheaper because they bypass the LLM reasoning loop entirely.

The most critical component of an RAI-aligned workflow is the Reasoning Agent. Instead of creating content from scratch, this agent performs:

- **Conflict Resolution:** Identifying shared facts and dropping speculative items.



- Factual Alignment: Grounding the final output strictly in the consensus-backed data provided by the consortium.
- Bias Mitigation: Neutralizing the idiosyncrasies of any single model.

In an engineered system, Responsible AI moves from "aspirational statements" to non-functional requirements enforced at specific gates, such as model registration or production deployment.

The final theoretical pillar is the operationalization of these systems through modern software engineering practices.

- Externalized Prompt Management: Prompts should be treated as code artifacts, stored in external repositories (e.g., GitHub), and loaded at runtime. This decouples agent behavior from application code, allowing non-technical stakeholders to refine constraints without requiring full redeployments.
- Containerization and Orchestration: Production workflows and their adapters (like MCP servers) should be Dockerized and orchestrated via Kubernetes. This ensures portability, scalability, and self-healing mechanisms.
- The KISS Principle: Complexity is a threat to reliability. Agentic systems should favor flat, function-driven designs over deeply nested abstractions. Simple workflows are not only easier to debug but are also more compatible with AI-assisted development tools like GitHub Copilot or OpenAI Codex.

3. METHODOLOGY

The recommended information architecture (Appendix A) for “Key Journey Phases for Developers” starts by reframing documentation around a lifecycle—Onboarding, API Usage, Model Registration, Debugging, Production Deployment, Evaluation, and Responsible AI—rather than isolated pages. Each phase maps to concrete artifacts: Quick Start and Terminology for Stage 1, API References and SDK Guidance for Stage 2, Model Repository docs for Stage 3, and so on.

An engineering mindset treats each doc section as a module with inputs, outputs, and invariants: prerequisites in, repeatable outcomes out. Quickstart becomes the equivalent of a reference implementation of the whole stack, while deeper sections (e.g., RAG, observability, Responsible AI) act as orthogonal sub systems that can be independently evolved. This mirrors microservice design: clear contracts, minimal coupling, and explicit dependencies.

Journey Phase	Description	Documentation
Stage 1 - Onboarding	Help users get from "zero-to-agent": environment setup, prerequisites, and persona-specific onboarding.	Quick Start Guide Terminology
Stage 2 - API Usage	Teach developers how to connect, configure, and consume APIs/SDKs with correct parameters.	API References SDK Guidance
Stage 3 - Model Registration & Model Repository	Demonstrate end-to-end model registration, usage patterns, and full-stack Model repository.	Model Registration Model Repository
Stage 4 - Debugging & Troubleshooting	Centralize error handling, logs, and remediation steps with FAQs and migration playbook.	Troubleshooting Migration Playbook
Stage 5 - Production Deployment & Monitoring	Define production best-practices: deployment, monitoring, auditing, and cost controls.	Production Checklist
Stage 6- Evaluation	Establish continuous observability for AI Agents.	Evaluation
Stage 7 - Responsible AI	Establish policies, standards, and responsible AI documentation.	RAI at Intuit RAI Use Case RAI Experimentation RAI LLM Onboarding RAI Finetuning

This lifecycle alignment is characteristic of an engineering approach: the doc set is indexed by where a developer is in the pipeline, not by internal org charts. The Quick Start guide, for example, walks a developer from DevPortal project creation through agent provisioning, IDPS onboarding, local run, and endpoint testing; each step is testable and stateful, with explicit success conditions (e.g., “healthfull” endpoint returns expected payload, LLM Execution status returns no error). In effect, the Quick Start becomes a formalized “integration test” of the platform, with documentation as the human readable harness.



The agentic AI Quick Start is a case study in workflow engineering. It decomposes “build an agent” into ordered, verifiable stages:

1. Create/register a non Playground service in DevPortal, with constraints on GitHub org naming to correctly set the asset alias.
2. Select the Agent Starter Kit template, binding the service to a Python Agent Starter Kit implementation.
3. Clone the generated GitHub repo, establish a virtual environment, and install dependencies via poetry.
4. Configure IDPS policy IDs and secret paths in configuration classes for base, pre prod, and prod settings, enforcing security and environment separation.
5. Integrate with LLM Execution by choosing a current supported model from the Model Access catalog and updating URLs in configuration files for each environment.
6. Run the agent as a FastAPI service (CLI or Docker) and verify readiness via health endpoints.
7. Invoke the agent endpoint with a fully parameterized curl command, where every header and field is explicitly documented and tied back to platform entities (Experience ID, asset alias, IAM ticket, realm ID).

This pipeline exhibits three engineering traits: explicit contracts (inputs/outputs per step), environment aware configuration, and reproducible state transitions. Agentic behavior is not treated as magic; it is a deterministic outcome of wiring together DevPortal assets, security policies, model routing, and observability endpoints.

3.1 Sprints, Gap Analysis, and Documentation as a Product

The execution plan institutionalizes an engineering mindset by applying backlog, prioritization, and sprints to documentation itself. A gap matrix tags each missing capability—Quick Start, error codes, mobile SDK guidance, Responsible AI docs—with P0–P2 severity and concrete remediation steps. This is indistinguishable from a defect tracker on a critical service: P0 items block core flows and must be burned down before GA.

The 9 week sprint plan then schedules doc work the way one would schedule feature development: Monday planning, mid week execution, Friday demo and retrospective, with Git practices (frequent commits, JIRA updates) enforced for documentation assets. Docs are thus versioned, reviewed, and dog fooded as code; they become part of the releasable artifact of the AI platform rather than static collateral.

Furthermore, the Responsible AI and 3PRM workstream further extends the engineering mindset into governance. The gap matrix explicitly flags “Responsible AI & Compliance Docs” as missing policy mapping, standards, and 3PRM steps, with an action to publish requirements and wire in official process links. This moves Responsible AI from aspirational statements to operational constraints on the workflow.

AI Hub Stage Mapping

● Onboarding (7 docs)

Document	Status	Priority	Estimated Effort	Sprint Planning
Quickstart	● Good	P0	Small	Sprint 1
Project & Use Case Registration	● Not created	P1	Small	Sprint 2
Tutorial: LLM completion using Express V3 APIs	● Not created	P1	Small	Sprint 3
Tutorial: LangGraph	● Not created	P1	Medium	Sprint 5-6
Tutorial: MCP	● Not created	P1	Medium	Sprint 6-7
API Reference	● Good	P1	Small	N/A
Terminology	● Good	P1	Small	N/A



By anchoring to the internal Responsible AI homepage, program process site, GenAI use case checklist, applied data science RAI guidance, and the Third Party Risk Management microsite, the documentation embeds formal interfaces between technical workflows and governance systems. In engineering terms, Responsible AI becomes a non functional requirement enforced at specific gates (e.g., model registration, production deployment) rather than an optional appendix.

4. RESULTS AND DISCUSSION

Agentic AI systems are themselves orchestration engines over tools, models, and data sources. An engineering mindset for their documentation borrows directly from software architecture (See also Appendix A):

- Clear abstraction layers: top level conceptual docs (MDLC, Generative AI lifecycle, Agentic AI) give developers a mental model; lower layers specify APIs, configuration, and examples.
- Composable patterns: Starter kits, mobile SDK guidance, and cross platform examples are reusable patterns that encode “known good” architectures into code and config, not prose alone.
- Observability by design: evaluation, tracing, logging, and cost controls are treated as first class doc topics (Stage 4–6 in the journey), enabling engineers to reason about runtime behavior and production risk.
- Feedback loops: quarterly doc reviews, NPS surveys, and support channel log mining are defined as continuous improvement mechanisms, analogous to SRE error budget reviews for services.

In such a regime, the agentic workflow and its documentation co evolve: each new capability (e.g., GraphRAG integration or MCP marketplace onboarding) is introduced through a pattern that spans architecture diagrams, reference configuration, and an executable tutorial, all placed within the appropriate journey phase.

5. CONCLUSION

The AI platform doc revamp illustrates how an engineering mindset transforms documentation from static reference into infrastructure that shapes how agentic AI is built, deployed, and governed. By aligning information architecture with lifecycle stages, encoding workflows into Quick Start pipelines, and subjecting documentation to the same sprint discipline and governance as code, the team effectively designs agentic AI workflows as first class engineered systems.

In modern AI platforms, the boundary between system design and documentation design is porous. A well designed agent workflow is not only implemented in services and SDKs; it is also concretely realized in the structure, precision, and operability of its documentation, which functions as the control surface by which humans safely and reliably orchestrate agentic systems.

APPENDIX A- Recommended Information Architecture

Scope: What do our customers want to achieve? e.g., how to build an agent?

Current issue: Our existing knowledge base is fragmented and outdated.

Context: AI Native application development

Top-Level Navigation (Homepage/Main Menu):

1. Introduction

- What is the Machine Learning Model Development Lifecycle (MDLC)?
- What is Generative AI Application Development Lifecycle? (P0)
- What is Agentic AI?



2. Setting Up Your Development Environment

- Tools and Accounts: Overview of required platforms and credentials
- AI Assist Code Generation Tools: (e.g., Cursor, Windsurf)
- Platform-Specific Tooling: (e.g., AWS Q, Azure AI, etc.)

3. Quick Start (General Onboarding)

- Getting Started: Journey from Zero to GenAI Agent (P0)
- Quickstart Guide for AI Development UX (AI Workbench)(P0)

4. Building Your First Agent & Use Case (Agent-Specific Quick Start)

- Develop a Simple Use Case (step-by-step walkthrough)
- Build Your First Chatbot (agent-focused onboarding)
- Agent Starter Kits (templates, starter packs for scale) (P0, P0)

5. Deployment

- Testing & Registry
- Agent Development
- Agent/Tool Protocols
- Agent/Tool Registry (P0)
- Evaluation
- Agent Deployment
- Agent Tracing and Debugging (P0, P0)
- Building and Deploying Applications (P0)
- Integrating with Other Systems (P0 ,P0)

5. Deployment

- Testing & Registry
- Agent Development
- Agent/Tool Protocols
- Agent/Tool Registry (P0)
- Evaluation
- Agent Deployment
- Agent Tracing and Debugging (P0, P0)



- Building and Deploying Applications (P0)
- Integrating with Other Systems (P0 ,P0)

Core Capabilities

(Generative) AI Application Development

- Conversational Experience
- Dialog Management
- Experience Analytics
- Design & Rendering
- Testing & Registry
- Natural Language Understanding
- A List of all APIs for AI development (P0)
- Generic Features
- LLM Service Integration
- Retrieval-Augmented Generation (RAG)
- LLM Fine-Tuning Service (TBD)

Evaluation & Optimization

- Creating Ground Truth (Labeled Data/Annotation)
- Evaluation Methods
- Optimization Techniques
- Prompt Optimization
- Data for Eval
- Prompt Logging
- Tracing data
- Synthetic Data Generation (SDG)
- Labeling Data

AI Observability

- Monitoring (P0)
- Logging & Tracing (P0)
- LLM Availability & Lifecycle Management



- Cost (P0)

Agent Development

- Agent Marketplace for Discovery
- Agentic Asset Creation and GSK onboarding
- Agent Development
- Prompt Engineering & Management
- Agent/Tool Protocols
- Agent/Tool Registry (P0)
- Evaluation
- Agent Deployment
- Agent Tracing and Debugging (P0, P0)
- Online Monitoring

Data Cognition & Integration

- Intelligent Data Cognition Layer
- Data Connection & Integration (P0)

Developer Experience

- Best Practices (Industry Benchmark) (P0)
- Reusable Agents & Tools
- Production Checklist (P0)

Troubleshooting & Support

- Building and Deploying Applications (P0, P0)
- Integrating with Other Systems (P0, P0)
- Data Connection Issues
- Support Channels (P0)
- Feedback

Special Topics

- Responsible AI Compliance (RAI)
- 3PRM Process
- AI Standards (P0)



Reference

- API Reference (P0)
- SDK Documentation (P0)
- Configuration Files (P0)

Resources

- FAQ (P0, P0)
- Glossary (P0)
- Templates & Examples
 - Prompt Template and best practices
- External References

Contribution and Feedback

- How to Contribute (Open Contribution)
- Feedback
- Product Feature Request (PFR)

REFERENCES

1. Chen, M. et al., 2023. Agentic AI: Architectures and Challenges. *Journal of Artificial Intelligence Research*, 72, pp.123-145.
2. Wang, X. & Li, J., 2022. Multi-agent Systems for Complex Task Coordination. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4), pp.1472-1485.
3. Zhang, Y. et al., 2021. Recursive Reasoning in AI Agents: ReAct and Beyond. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(7), pp.6543-6550.
4. Smith, A. & Patel, R., 2020. Memory Architectures in Intelligent Agents. *Cognitive Systems Research*, 61, pp.45-56.
5. Brown, T. et al., 2020. Language Models Are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33, pp.1877-1901.
6. Russell, S. & Norvig, P., 2016. *Artificial Intelligence: A Modern Approach*. 3rd ed. Pearson.
7. Mnih, V. et al., 2015. Human-Level Control Through Deep Reinforcement Learning. *Nature*, 518(7540), pp.529-533.
8. LeCun, Y., Bengio, Y. & Hinton, G., 2015. Deep Learning. *Nature*, 521(7553), pp.436-444.
9. Sutton, R.S. & Barto, A.G., 2018. *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press.
10. Doshi-Velez, F. & Kim, B., 2017. Towards A Rigorous Science of Interpretable Machine Learning. *arXiv preprint arXiv:1702.08608*.
11. Chollet, F., 2019. On the Measure of Intelligence. *arXiv preprint arXiv:1911.01547*.
12. Marcus, G., 2020. The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence. *arXiv preprint arXiv:2002.06177*.
13. Chen, D. et al., 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint arXiv:2201.11903*.
14. Janner, M. et al., 2021. Offline Reinforcement Learning with Implicit Q-Learning. *Advances in Neural Information Processing Systems*, 34, pp.1116-1127.
15. Vinyals, O. et al., 2019. Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning. *Nature*, 575, pp.350-354.



16. Zilberstein, S., 2015. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. Morgan & Claypool Publishers.
17. Lipton, Z.C., 2016. The Mythos of Model Interpretability. arXiv preprint arXiv:1606.03490.
18. Levesque, H.J., Davis, E. & Morgenstern, L., 2012. The Winograd Schema Challenge. Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning, pp.552-561.
19. Goodfellow, I., Bengio, Y. & Courville, A., 2016. Deep Learning. MIT Press.
20. Bommasani, R. et al., 2021. On the Opportunities and Risks of Foundation Models. arXiv preprint arXiv:2108.07258.
21. Sutton, R.S., Precup, D. & Singh, S., 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. Artificial Intelligence, 112(1-2), pp.181-211.
22. Pearl, J., 2009. Causality: Models, Reasoning, and Inference. 2nd ed. Cambridge University Press.
23. Amershi, S. et al., 2019. Guidelines for Human-AI Interaction. Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, pp.1-13.
24. Sutton, R.S., 1998. Reinforcement Learning: An Introduction. IEEE Transactions on Neural Networks, 9(5), pp.1054-1054.
25. Kwiatkowska, M., Norman, G. & Parker, D., 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. Computer Aided Verification, pp.585-591.