# Operationalizing Agentic AI: A Strategic Framework for Lifecycle Management, Documentation, and Dependency Orchestration

## 1. The Delivery Crisis in Agentic Systems

### 1.1 The Paradigm Shift: From Deterministic Code to Probabilistic Behavior

The contemporary software delivery landscape is witnessing a fracture between traditional deterministic methodologies and the emergent requirements of Agentic AI. For the Project or Delivery Manager (PM/DM) tasked with orchestrating agentic systems, this shift is not merely technical but existential. In traditional software development, the contract between input and output is absolute: if a specific set of parameters is passed to a function, the result is predictable, repeatable, and binary—it either works or it fails. The "Definition of Done" is an assertable state.

Agentic systems, however, operate on a probabilistic substrate. Powered by Large Language Models (LLMs), these agents do not simply execute instructions; they reason, plan, and act within a bounded stochastic environment.1 They act as autonomous entities that perceive intent and select tools to achieve a goal. For a delivery team that builds the system but not the host infrastructure, and relies on external teams for data, this introduces a profound layer of volatility. The agent is a probabilistic reasoning engine sitting on top of a deterministic infrastructure, fueled by data streams that are often managed by siloed organizational units.

The friction described in the user's operational context—non-standard documentation, unclear lifecycle milestones, and estimation failures during discovery—is symptomatic of a broader industry lag. We are attempting to manage stochastic "digital employees" with tools designed for static "software artifacts." When an estimation is made during discovery without a rigorous checklist, it fails to account for the "Ambiguity Tax" inherent in GenAI. A task that appears linguistically simple (e.g., "Summarize the risks in this contract") can be technically monstrous if the retrieval mechanism (RAG) is flawed or the underlying data schema is unstable.3

### 1.2 The Federated Challenge: Managing Without Ownership

The specific architectural constraint identified—where the delivery team owns the agentic behavior but relies on external host infrastructure and third-party data teams—creates a "Federated Delivery" challenge. This is a high-dependency environment where the "Mean Time to Resolution" (MTTR) for any issue is often dictated by the responsiveness of the Data Team or the Infrastructure Team, not the Delivery Team itself.4

In this context, the Delivery Manager acts less like a construction foreman and more like a diplomat and systems integrator. The "struggle" with dependencies is often a struggle of visibility and contract. If the Data Team changes a column name in a database without notifying the Agent team, a traditional application might throw a 500 Error. An Agentic system, however, might simply hallucinate a plausible-sounding but factually incorrect answer because it "inferred" the wrong data from the changed schema. This "silent failure" mode necessitates a completely different approach to documentation and lifecycle management, shifting from "API Documentation" to "Data Contracts" and "System Cards".5

Furthermore, the role of the Business as the Subject Matter Expert (SME) and Product Owner introduces a specific psychological complexity. SMEs are accustomed to deterministic software where bugs are fixed permanently. In Agentic AI, "bugs" are often distributional—meaning the agent works 90% of the time. The Delivery Manager must therefore educate the SME to move from "User Acceptance Testing" (UAT) to "Behavioral Evaluation," a process that looks more like employee performance review than software QA.7

### 1.3 Report Objectives and Scope

This report serves as a comprehensive operational blueprint for Delivery Managers navigating this specific terrain. It synthesizes industry best practices from Booz Allen, Microsoft, and AWS, along with specialized insights on Data Contracts and Agentic Architecture, to provide the three core deliverables requested:

1. **The Flow:** A redefined Agentic Software Development Lifecycle (SDLC) that accounts for the iterative, cognitive nature of agent building.
2. **Discovery Checklist:** A granular estimation and feasibility framework to prevent scope creep before a single line of code is written.
3. **Confluence Documentation Structure:** A standardized hierarchy that professionalizes the knowledge management of AI agents, bridging the gap between technical complexity and business intent.

The following analysis integrates these elements into a cohesive narrative, designed to transform the chaotic "experimental" phase of AI adoption into a rigorous, engineering-led delivery discipline.

## 2. The Agentic Delivery Lifecycle: "The Flow"

To operationalize the delivery of agentic systems, we must abandon the linear fiction of Waterfall and the feature-obsessed velocity of Scrum. Agents require a lifecycle that is circular, focusing on "Cognitive Iteration" rather than just "Feature Completion." The proposed flow, derived from the "GenAI SDLC" models proposed by Microsoft and AWS, is divided into five distinct stages. This flow specifically addresses the constraint of external infrastructure by placing heavy emphasis on the "Scaffolding" phase to insulate the agent from platform volatility.4

### 2.1 Phase 1: Intent Design and Feasibility (The Discovery Phase)

The primary failure mode in Agentic projects is not technical inability but "Intent Ambiguity." In traditional SDLC, this phase gathers requirements. In Agentic SDLC, this phase defines the "Cognitive Boundary." The goal is to determine if the business request requires an *Agent* (which can plan and act) or merely an *Automation* (which follows a script).

* **Ambiguity Mapping:** The team must work with the SME to map the decision-making process. If the SME cannot articulate the rule they use to make a decision ("I just know it when I see it"), the Agent will fail. The Discovery phase must quantify this ambiguity.
* **Dependency Audit:** Before estimation, the Delivery Manager must audit the external "Data Team's" readiness. Is the data accessible via API? Is it structured? Is there a Data Contract in place? (See Section 4 for deep analysis of Data Contracts).
* **Feasibility Gating:** This phase ends with a "Go/No-Go" decision based on the Checklist (detailed in Section 3), preventing the "estimation without checklist" error identified in the user query.

### 2.2 Phase 2: Architectural Scaffolding (The Integration Layer)

Since the team does not own the host infrastructure, Phase 2 focuses on building a robust client-side architecture—the "Scaffolding"—that connects the Agent to the Host and the Data.

* **Tool Definition (MCP):** The team defines the "Tools" the agent will use. This uses the Model Context Protocol (MCP) to standardize how the agent connects to external data sources.10 Instead of hard-coding SQL queries (which creates tight coupling), the team exposes "Tools" like fetch\_customer\_history(id) or search\_policy\_docs(query).
* **Guardrail Implementation:** Safety must be architectural, not prompt-based. The team implements a "Gateway" layer that sits between the User and the Agent. This layer handles PII scrubbing (masking sensitive data) and topic filtering *before* the prompt is sent to the external infrastructure.1
* **Mock Server Setup:** To mitigate the reliance on the "Data Team," the delivery team builds Mock APIs during this phase. This allows the Agent team to develop logic even if the Data Team's API is down or in development.12

### 2.3 Phase 3: Cognitive Development (The Iteration Loop)

This is the core "Build" phase, but it differs significantly from coding. It is a cycle of **Prompt Engineering**, **RAG Optimization**, and **Tool Binding**.

* **Prompt Architecture:** The developers craft the "System Prompt"—the persona and rules of the agent. This involves "Chain of Thought" (CoT) engineering, instructing the agent to "Think, Plan, Act" in sequence.13
* **Context Window Management:** Because the host infrastructure likely has token limits, the team must optimize *what* data is retrieved. This involves "Reranking" search results to ensure only the most relevant context is fed to the agent, reducing costs and latency.3
* **Tool Selection Tuning:** The primary bug in agents is "Hallucinated Tool Use"—trying to call a tool that doesn't exist or using the wrong parameters. This phase involves rigorous unit testing of the agent's *intent classification*.14

### 2.4 Phase 4: Behavioral Assurance (The Evaluation Phase)

Testing shifts from "Unit Testing" (does the code run?) to "Behavioral Evaluation" (did the agent act responsibly?).

* **SME Adversarial Testing:** The Business Owner acts as a "Red Teamer," trying to trick the agent into violating its rules. This is where the SME validates the "Guardrails" defined in Phase 2.15
* **Synthetic Evaluation (LLM-as-a-Judge):** The team runs thousands of automated test cases where a separate, powerful LLM (the "Judge") grades the Agent's responses against a rubric of Accuracy, Safety, and Tone.16
* **Dependency Stress Testing:** The agent is tested against the live Data Team APIs to ensure it handles latency and error states gracefully (e.g., "The Data API is down, so the Agent politely apologizes instead of crashing").

### 2.5 Phase 5: Continuous Orchestration (The Operations Phase)

Deployment is not the end. Agentic systems suffer from "Drift"—as the world changes, the agent's static knowledge becomes obsolete.

* **Feedback Loops:** The system must capture User Feedback (Thumbs Up/Down) and route "Thumbs Down" examples back to the SME for review.1
* **Data Contract Monitoring:** The system continuously monitors the external Data Team's API. If the schema changes (breaking the contract), an alert is fired immediately, flagging the dependency failure rather than an agent failure.5

## 3. The Discovery Phase Checklist & Estimation Framework

The user explicitly highlighted that "initial estimates are made during the discovery phase without a checklist." This is the root cause of project overrun in AI. Estimating an AI project based on "gut feel" is disastrous because complexity in AI is non-linear. A task that involves *reasoning* is exponentially harder than a task that involves *retrieval*.

To solve this, we propose a comprehensive **Discovery Phase Checklist**. This checklist acts as a "Definition of Ready." If the team cannot check these boxes, the project remains in Discovery, preventing premature commitment to delivery dates.

### 3.1 The "Definition of Ready" Checklist

The following checklist integrates business alignment, data readiness, and technical feasibility, drawn from best practices in AI project initiation.17

#### **Section A: Business Alignment & SME Availability**

* **[ ] Problem Determinism Check:** Is the desired outcome probabilistic?
  + *Context:* If the SME expects the system to follow a strict, unvarying 50-step decision tree, an Agent is the wrong tool. Use traditional automation. Agents are for ambiguous tasks.2
* **[ ] SME "Teacher" Commitment:** Has the SME committed 4-8 hours/week for the first month?
  + *Context:* Agents are not "requirements driven"; they are "feedback driven." They need a teacher to review logs and correct behavior. If the SME is unavailable, the agent will never converge on the correct behavior.7
* **[ ] Error Tolerance Definition:** Has the business defined an Acceptable Failure Rate?
  + *Context:* A requirement of "100% accuracy" is impossible in GenAI. The business must agree to a threshold (e.g., "90% accuracy is acceptable if the remaining 10% are routed to a human").4
* **[ ] ROI & Token Budget:** Is there a budget for ongoing inference costs (tokens)?
  + *Context:* Unlike software which has a fixed server cost, agents cost money per question. High-volume, complex reasoning tasks can destroy ROI.18

#### **Section B: Data & Dependency Readiness (The "Other Team" Factor)**

* **[ ] API Accessibility & Documentation:** Are the Data Team's APIs fully documented (OpenAPI/Swagger) and accessible to the Agent's environment?
  + *Context:* "We have the data" is not enough. The agent needs *programmatic* access. If the data is in a CSV on someone's desktop, the project is not ready.19
* **[ ] Semantic Clarity (Metadata):** Do the database columns have clear, descriptive names?
  + *Context:* LLMs use column names to understand data. A column named col\_x\_99 is useless. It must be named customer\_lifetime\_value\_usd for the agent to use it correctly.3
* **[ ] Data Contract Contact:** Is there a named Point of Contact (POC) on the Data Team?
  + *Context:* Who do you call when the API breaks? If this isn't defined, the delivery team will be blocked indefinitely during outages.20
* **[ ] PII/Compliance Scan:** Does the data contain Personally Identifiable Information (PII)?
  + *Context:* If yes, you must estimate the effort to build a "Masking Layer" (middleware) to redact this data before it hits the external LLM infrastructure.1

#### **Section C: Technical Complexity Indicators**

* **[ ] Reasoning "Hop" Count:** How many logical steps does the agent need to take?
  + *Context:* "Find the file" = 1 Hop. "Find the file, read it, look up the user in the DB, and compare" = 3 Hops. Failure probability increases exponentially with each hop.
* **[ ] Tool Count:** How many external tools does the agent need?
  + *Context:* < 3 tools is stable. > 5 tools leads to "Tool Confusion" where the agent calls the wrong API. This requires advanced prompt engineering.13
* **[ ] Latency Requirement:** Does the user need a sub-2-second response?
  + *Context:* Agentic loops (Plan -> Act -> Observe) are inherently slow. If speed is critical, the architecture must change (e.g., caching, smaller models), impacting the estimate.21

### 3.2 The Estimation Matrix: Converting Complexity to Story Points

Traditional Story Points often fail because they measure *effort*, not *uncertainty*. We recommend using **"Complexity Buckets"** derived from the checklist results to assign a multiplier to the base development time.22

| **Complexity Bucket** | **Criteria Details** | **Risk Multiplier** | **Estimation Implication** |
| --- | --- | --- | --- |
| **Bucket 1: Retrieval (Low)** | Single Data Source, Read-Only, No Multi-step reasoning. | **1.2x** | Standard Dev Time + 20% buffer for prompt tweaking. |
| **Bucket 2: Action (Medium)** | 2-3 Tools, Single logical step (e.g., "Reset Password"), Internal APIs only. | **2.0x** | Requires Integration Testing and basic Guardrails. |
| **Bucket 3: Workflow (High)** | Multi-step reasoning (Plan -> Act), 3+ Tools, External Data Dependency. | **4.0x** | Requires extensive "Red Teaming" and RAG optimization. |
| **Bucket 4: Autonomous (Critical)** | Open-ended goals ("Analyze trends"), Write-Access to DB, Financial impact. | **8.0x** | The majority of time will be spent on Safety/Simulations, not coding. |

*Strategic Insight:* By forcing the Business/SME to categorize their request into one of these buckets *during Discovery*, the Delivery Manager can defensibly argue for the necessary time. If the SME asks for a "Bucket 4" agent but wants it in a "Bucket 1" timeline, the checklist provides the objective evidence to push back.

## 4. Documentation Architecture: The Confluence Standard

The user identified that "documentation is not standardized." In Agentic AI, documentation is the bridge between the probabilistic behavior of the system and the deterministic expectations of the business. It must serve as a "Knowledge Graph" of the system.

We propose a **Confluence Space Hierarchy** that mirrors the architecture of the agent itself. This structure ensures that every dependency, prompt, and risk is indexed and ownership is clear.24

### 4.1 Top-Level Confluence Hierarchy

**: Agentic AI Platform Delivery**

1. **01. Program Overview & Governance**
   * *Mission Statement*
   * *The "Agent Manifesto" (Ethical guidelines & global guardrails)*
   * *Team Structure (Delivery vs. Data vs. Infra)*
2. **02. Agent Registry (The "System Cards")**
   * *Live Agents*
   * *In-Development Agents*
   * *Retired Agents*
3. **03. Knowledge & Data Contracts** (The Dependency Map)
   * *Data Contract Repository*
   * *External Tool Specifications (MCP Definitions)*
   * *Knowledge Base Inventory (PDFs, Wikis)*
4. **04. Prompt Engineering Library**
   * *System Prompts (Versioned)*
   * *Persona Definitions*
   * *Few-Shot Example Libraries*
5. **05. Evaluation & Quality Assurance**
   * *Golden Datasets (Links)*
   * *SME Feedback Logs*
   * *Red Team Reports*
6. **06. Operations & Runbooks**
   * *Incident Response Protocols*
   * *Monitoring Dashboards (Links)*

### 4.2 Deep Dive: The "System Card" Template (Page Type 2)

The most critical document is the **System Card**. This replaces the traditional "Software Requirement Specification" (SRS). It is a living document that describes the *behavior* and *limitations* of the agent. This document must be signed off by the SME.6

**Template Structure:**

| **Section** | **Content Description** | **Purpose** |
| --- | --- | --- |
| **Agent Identity** | Name, Role, Model Version (e.g., GPT-4o), Technical Owner. | Establishes accountability. |
| **Intended Capabilities** | What questions can it answer? What actions can it take? | Sets the "Happy Path" expectations. |
| **Explicit Limitations** | "Cannot answer questions about data older than 24h." "Cannot give financial advice." | **Crucial:** Manages SME expectations about what the agent *cannot* do. |
| **Tool Access & Permissions** | List of all APIs (Tools) the agent can call. (e.g., Read: UserDB, Write: TicketSystem). | Security audit trail. |
| **Risk Profile** | Known hallucination triggers, bias considerations, PII handling strategy. | Transparency for governance/compliance. |
| **Evaluation Metrics** | Target Accuracy (e.g., 90%), Current Accuracy (e.g., 85%), Latency SLA. | quantitative measurement of success. |

### 4.3 Deep Dive: The "Data Contract" Page (Page Type 3)

To address the "reliance on other teams," every data dependency must have a **Data Contract Page**. This acts as the Service Level Agreement (SLA) between the Delivery Team and the Data Team.5

**Template Structure:**

* **Dependency Name:** (e.g., Customer\_Profile\_API)
* **Provider Team:** (e.g., Enterprise Data Services - Team B)
* **Point of Contact:** (Name/Email of the Data Engineer)
* **Schema Definition:** A JSON or YAML snippet defining the *exact* fields the agent expects.
  + *Constraint:* "Field status must be one of: [Active, Inactive, Pending]. Changes to this list break the agent's logic."
* **Freshness SLA:** "Data is updated every hour."
* **Change Protocol:** "Provider must notify Delivery Team 10 days before any schema change."
* **Mock Server Endpoint:** Link to the internal Mock API used for testing when the real API is down.

## 5. Managing External Dependencies: The Data Contract Strategy

The user noted a "reliance on other teams who manage the data" as a primary struggle. In the world of LLMs, **Data is Context**, and **Context is Control**. If the data quality fluctuates, the agent's intelligence fluctuates.

### 5.1 The Concept of Data Contracts in AI

A Data Contract is not just a document; it is an enforced agreement on the *shape* and *semantics* of data.20 In a traditional app, if a database field changes from user\_id to id, the app crashes (a loud failure). In an agent, the LLM might hallucinate a user\_id to keep the conversation going (a silent failure).

To mitigate this, the Delivery Manager must implement a **"Contract-First" Development Flow**:

1. **Define the Contract:** Before development starts, the Delivery Team and Data Team agree on the Schema (using standard formats like Open Data Contract Standard - ODCS).
2. **Enforce via CI/CD:** The Delivery Team sets up a nightly integration test that queries the Data Team's API against the contract. If the schema has drifted, the pipeline fails, and an alert is sent to *both* teams.
3. **Semantic Locking:** The contract must specify field descriptions. If revenue changes from "Gross" to "Net," the contract is broken, because the Agent's reasoning ("Is this customer profitable?") will now be flawed.

### 5.2 Decoupling via Mocking

Since the Delivery Team does not own the infrastructure or the data, they are vulnerable to external outages. The report recommends a **"Mock-Driven" Architecture**.12

* **The Mock Server:** The Delivery Team builds a lightweight server that mimics the Data Team's API.
* **Development Isolation:** Developers write prompts and logic against the Mock Server. This ensures that the Agent's "reasoning logic" is tested independently of the Data Team's "infrastructure stability."
* **Resiliency:** This allows the Delivery Team to continue sprinting even if the Data Team is blocking them, solving a major delivery bottleneck.

## 6. The SME Interaction Model: From Product Owner to "Gardener"

The user states that the Business acts as the SME/Product Owner. This role must evolve. Managing an agent is not like managing a feature backlog; it is like gardening—it requires constant pruning, watering (data), and guiding.

### 6.1 Education and Expectation Management

The Delivery Manager's first task is to realign the SME's expectations regarding **Probabilistic Outcomes**.

* **The "Hallucination" Conversation:** The PM must explain that hallucinations are not just "bugs" but a feature of the model's creativity. The goal is to minimize them, not eliminate them (unless using strict constraints).
* **The "Golden Dataset" Requirement:** The SME cannot just say "it doesn't work." They must provide a **Golden Dataset**—a spreadsheet of 50-100 questions and their *perfect* answers. This becomes the ground truth for all testing.14

### 6.2 The User Acceptance Testing (UAT) Revolution

Standard UAT scripts ("Click button A, expect B") do not work for agents because users will phrase questions infinitely differently.

* **Adversarial UAT:** The SME should be encouraged to try and "break" the agent—asking ambiguous questions, using slang, or trying to trick it. This "Red Teaming" is valuable data.15
* **Grading Rubric:** Instead of Pass/Fail, the SME grades responses on a scale (1-5) for Accuracy, Tone, and Safety.
* **Feedback Loops:** The Delivery Manager must establish a weekly "Triage" meeting where the SME reviews the "Low Confidence" logs. This feedback is then fed back into the Prompt Library or Data Contracts.16

## 7. Conclusion

For the Delivery Manager operating in this federated, agentic environment, success requires a shift from "Task Management" to "Dependency Orchestration." The chaotic nature of the current state—unclear milestones, ad-hoc estimation, and fragmented documentation—can be tamed by applying the rigorous structures outlined in this report.

By implementing the **Agentic Lifecycle Flow**, enforcing **Data Contracts** to manage external teams, and professionalizing knowledge management with **System Cards** in Confluence, the delivery team can insulate itself from external volatility. The **Discovery Checklist** serves as the primary defense mechanism, ensuring that the team never commits to a timeline before understanding the true complexity of the probabilistic system they are building. This framework transforms the "Magic" of AI into the manageable "Mechanics" of enterprise engineering.

#### Works cited

1. Agentic Software Development Decoded - Booz Allen, accessed January 15, 2026, <https://www.boozallen.com/insights/velocity/agentic-software-development-decoded.html>
2. The Complete Guide to Agentic AI (PART #2): Building and Deploying Autonomous Systems for…, accessed January 15, 2026, <https://bishalbose294.medium.com/the-complete-guide-to-agentic-ai-part-2-building-and-deploying-autonomous-systems-for-17103f7b9750>
3. Agentic AI for Full-Cycle Software Development [CTO Guide], accessed January 15, 2026, <https://zencoder.ai/blog/agentic-ai-for-full-cycle-software-development-the-ctos-guide>
4. Evolving software delivery for agentic AI - AWS Prescriptive Guidance, accessed January 15, 2026, <https://docs.aws.amazon.com/prescriptive-guidance/latest/strategy-operationalizing-agentic-ai/software-delivery.html>
5. Data Contracts: The Complete Guide to Data Contract Standards, Tools & Best Practices, accessed January 15, 2026, <https://datacontract.com/>
6. Model Card Template for an AI Developer - FairNow, accessed January 15, 2026, <https://fairnow.ai/model-card-template-ai-developer/>
7. To what level of granularity does an SME typically test in UAT? : r/QualityAssurance - Reddit, accessed January 15, 2026, <https://www.reddit.com/r/QualityAssurance/comments/8xgx37/to_what_level_of_granularity_does_an_sme/>
8. User Acceptance Testing with AI Agents and Agentic AI - XenonStack, accessed January 15, 2026, <https://www.xenonstack.com/insights/user-acceptance-testing-ai-agents>
9. Modernizing the SDLC process with Agentic AI | by Shashikanta Parida | Data Science + AI at Microsoft | Medium, accessed January 15, 2026, <https://medium.com/data-science-at-microsoft/modernizing-the-sdlc-process-with-agentic-ai-8330163bca29>
10. The complete guide to agentic AI basics | by Fahim ul Haq | Jan, 2026, accessed January 15, 2026, <https://medium.com/@fahimulhaq/complete-guide-to-agentic-ai-basics-ec340176d0a8>
11. How to Manage Your AI App Dependencies - YouTube, accessed January 15, 2026, <https://www.youtube.com/watch?v=Ds0t1f_KPMg>
12. How to deal with external API dependencies? : r/AI\_Agents - Reddit, accessed January 15, 2026, <https://www.reddit.com/r/AI_Agents/comments/1k9owwb/how_to_deal_with_external_api_dependencies/>
13. Building Effective AI Agents - Anthropic, accessed January 15, 2026, <https://www.anthropic.com/research/building-effective-agents>
14. The Ultimate Checklist for Rapidly Deploying AI Agents in Production - Maxim AI, accessed January 15, 2026, <https://www.getmaxim.ai/articles/the-ultimate-checklist-for-rapidly-deploying-ai-agents-in-production/>
15. ChatGPT Agent System Card - OpenAI, accessed January 15, 2026, <https://cdn.openai.com/pdf/839e66fc-602c-48bf-81d3-b21eacc3459d/chatgpt_agent_system_card.pdf>
16. LLMs project guide: key considerations - Microsoft Learn, accessed January 15, 2026, <https://learn.microsoft.com/en-us/ai/playbook/technology-guidance/generative-ai/getting-started/llmops-checklist>
17. The discovery phase of a project, and how it helps supercharge your IT initiatives, accessed January 15, 2026, <https://itrexgroup.com/blog/discovery-phase-of-a-project-practical-guide/>
18. The Discovery Phase: Crafting an Effective AI Agent Project ..., accessed January 15, 2026, <https://botscrew.com/blog/discovery-phase-crafting-effective-ai-agent-project-roadmap/>
19. What It Means to Be AI-Ready: Data Access, Interoperability, and the Future of Software (Part 1) - Sam Edelstein, accessed January 15, 2026, <https://samedelstein.medium.com/what-it-means-to-be-ai-ready-data-access-interoperability-and-the-future-of-software-part-1-a9abc8e0d03c>
20. Data Contracts Explained: Key Aspects, Tools, Setup in 2026 - Atlan, accessed January 15, 2026, <https://atlan.com/data-contracts/>
21. Managing Common and AI-Based Application Dependencies During a Migration, accessed January 15, 2026, <https://www.netscout.com/blog/managing-common-and-ai-based-application-dependencies-during>
22. An LLM-based multi-agent framework for agile effort estimation - arXiv, accessed January 15, 2026, <https://arxiv.org/html/2509.14483v1>
23. How to approach story point estimation with advent of AI Dev acceleration tools? - Scrum.org, accessed January 15, 2026, <https://www.scrum.org/forum/scrum-forum/94752/how-approach-story-point-estimation-advent-ai-dev-acceleration-tools>
24. Confluence AI Agents - Akira AI, accessed January 15, 2026, <https://www.akira.ai/ai-agents/confluence-ai-agents>
25. Confluence documentation guide: Best practices and tips, accessed January 15, 2026, <https://www.refined.com/blog/confluence-documentation-best-practices>
26. Google Model Cards, accessed January 15, 2026, <https://modelcards.withgoogle.com/>
27. 3 Data Contract Types and Examples - Gable.ai, accessed January 15, 2026, <https://www.gable.ai/blog/data-contract-example>