Agile Management for Foundation Model Based Agentic Systems: A Vision

Lucas Romao
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Rio de Janeiro, Brazil
lromao@inf.puc-rio.br

Yasmin Sandes
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Rio de Janeiro, Brazil
yasmin.sandes@aluno.puc-rio.br

Gabriel Mariquito
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Rio de Janeiro, Brazil
gsousa@inf.puc-rio.br

Jose M. C. Boaro
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Rio de Janeiro, Brazil
jboaro@inf.puc-rio.br

Marcos Antonio Alves
Federal University of Jequitinhonha and Mucuri Valleys (UFVJM)
Diamantina, Minas Gerais, Brazil
m.alves@ufvjm.edu.br

Marcos Kalinowski
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
Rio de Janeiro, Rio de Janeiro, Brazil
kalinowski@inf.puc-rio.br

Abstract

[Context] Foundation Model (FM)-based agentic systems extend the reasoning capabilities of FMs to enable autonomous software components. Yet, while these systems open new possibilities for software, managing their development through agile methodologies is not straightforward: agent development is inherently experimental, traditional agile frameworks offer no suitable artifacts for it, and non-technical stakeholders often lack AI literacy. [Goal] To address these challenges, in this vision paper we discuss how agile management can be extended to support the development of FM-based agentic systems. [Method] We ground our vision in lessons learned from agile management of Machine Learning (ML)-enabled systems development, extending it to an agentic perspective. [Results] Our vision comprises guidance organized around practices, artifacts, and roles. In terms of practices, we suggest that teams refine agent development through sprint-bound experimentation, with an initial specification of agent requirements and an offset cadence between AI and Software Engineers. Regarding artifacts, we propose specifying agent-intensive agile requirements through Agent and Data Stories, with Layers of Done versioning, such as an Agent Demo API and a Minimum Viable Agent. On roles, we advocate that Product Owners specify Agent and Data Stories collaboratively with AI Engineers. [Conclusion] We acknowledge that this guidance still require empirical evaluation. Even so, we propose it as a starting point for practitioners and researchers seeking to extend agile management to the development of agentic systems.

Keywords

Agile Management, Foundation Model Based Agents

1 Introduction

In the context of Machine Learning (ML)-enabled systems, intelligence emerges from models trained for narrow, well-delimited tasks, whose behavior is shaped by the training data and refined through iterative cycles of data preparation, training, and evaluation [4]. More recently, Foundation Models (FMs), such as Large Language Models (LLMs), have reshaped this landscape: pretrained on massive corpora designed primarily to natural language tasks, they are adapted to a wide range of tasks through prompting, context engineering, and fine tuning [2], moving much of the engineering effort from building task specific models to specifying, steering, and evaluating general purpose models embedded in software systems.

FM-based agentic systems take this transformation a step further. Rather than responding passively to requests, they extend the reasoning capabilities of FMs into software components that plan, invoke tools, maintain memory, and act autonomously in pursuit of user goals [11]. While these capabilities open new possibilities for software products, they also raise the question of how to manage the development of such systems.

Agile management seems a natural candidate for this task. Rooted in the Agile Manifesto values [3], agile approaches emphasize incremental value delivery, fast response to change, and close collaboration within cross-functional teams, characteristics that established them as the standard for modern software development [5].

In practice, however, points of friction emerge. Agent development is experimental by nature: hallucination control, prompt and context management, tool integration, and the progressive evolution of agent capabilities [11] advance through iterative trial and evaluation rather than predictable implementation tasks, undermining sprint planning and effort estimation. Traditional agile frameworks, in turn, offer no suitable artifacts for this kind of work: User Stories, Definitions of Done, and product backlog provide no place to specify agent goals, autonomy boundaries, guardrails, or evaluation procedures [7]. These difficulties are compounded by the lack of AI literacy among non-technical stakeholders, which widens the coordination gap between AI and software teams, fosters unrealistic expectations about agent behavior, and produces requirements that are hard to translate into agent development tasks. Together, these frictions leave agile practices, artifacts, and roles insufficiently tailored to agentic systems development.

From an academic standpoint, the community has so far explored mostly the opposite direction of this relationship: a growing body of work investigates how FMs and FM-based agents can reshape agile management itself, giving rise to the so-called AI-enhanced agile [6, 12]. Guidance on how agile management should be adapted to support the development of agentic systems, however, has not been addressed, a gap echoed by the research agenda of a recent seminar on agents and software engineering (SE) [9].

This gap does not need to be addressed from scratch. Over the past years, the research community has extended agile management

to the development of ML-enabled systems [7, 8, 10]. FM-based agents inherit the experimental and stochastic nature of ML components while introducing concerns about autonomy, orchestration, and open-ended behavior, making these lessons a natural foundation to build upon. In this vision paper, we therefore ground our vision in these lessons and extend them to an agentic perspective, organizing our guidelines around practices, artifacts, and roles.

In short, we suggest that teams refine agents through sprint bound experimentation, supported by a shared initial specification of agent requirements and an offset cadence between agent and conventional software work; we propose Agent and Data Stories for specifying agent-intensive requirements, versioned through Layers of Done (LoD) such as an Agent Demo API and a Minimum Viable Agent (MVA); and we advocate that Product Owners (PO) write these stories collaboratively with AI Engineers. While this guidance still requires empirical evaluation, we offer it as a starting point for practitioners and researchers seeking to extend agile management to agentic systems.

2 Background

2.1 FM-based Agentic Systems

LLMs enable use cases that go beyond earlier task-specific models, including open conversation, summarization, code generation, and reasoning. Their maturation made it possible to extend this reasoning beyond conversational interfaces, giving rise to FM-based agents, that is, software entities that employ an FM as their reasoning core to interpret natural language commands, decompose them into actionable plans, and act upon their environment, while coordinating with other agents or external tools [11].

In practice, the autonomy of these agents rests on a set of architectural capabilities layered around the FM. Tool calling turns textual reasoning into concrete actions, such as querying a database or triggering a business workflow. Skills package reusable instructions and procedures that extend agent competence without retraining the model. Memory mechanisms persist information across interactions, from short-term task context to long-term records of preferences and prior decisions. Notably, these capabilities are configured primarily via natural-language artifacts, such as prompts, tool definitions, and curated context, rather than via model training [11]. As a consequence, the engineering effort focuses on the orchestration layer surrounding the model, a characteristic that has direct implications for how the development of these systems should be managed.

2.2 Agile Management

Agile management is a widely adopted iterative and adaptive approach to software development [5], rooted in the principles of the Agile Manifesto [3]. It emerged to address the uncertainties inherent in software development, focusing on incremental value delivery, rapid adaptation to change, and the empowerment of multidisciplinary teams. Building on this view, agile management can be understood through a sociotechnical perspective in which collaboration among people is coordinated through activities and mediated by shared artifacts. Under this lens, managing software development hinges on continuous alignment among practices, artifacts, and roles. That is, how the work is conducted, which representations capture and communicate its state, and who is responsible for performing it. More than a mere checklist of guidelines, agile management embodies a cultural shift toward values and principles that promote teamwork and adaptability to change [5].

2.3 Lessons Learned from Agile Management for ML-enabled Systems Development

The literature on agile management of ML-enabled systems development [7, 8, 10] yielded lessons that we organize along the three dimensions: practices, artifacts, and roles.

2.3.1 Practices. Four practices are worth mentioning for managing ML-enabled systems. First, anchor ML work in a shared initial vision of ML requirements before building the backlog: stakeholders should gather an understanding on what each intelligent module should do, under which constraints, and with which data, instead of jumping straight into modeling. Second, let business questions bound ML experimentation. Experimentation is an inherent part of the ML lifecycle and should be embraced, yet it must stay bounded by business questions. Even failed experiments are valuable when bounded by business directions. Third, refine models incrementally, scoping each modeling task to a single sprint and planning incremental versions across sprints when a model is too complex for one iteration. Fourth, offset the model and software work cadence, as depicted in Figure 1. Model refinement cycles have unpredictable duration and outcomes, so software work should not depend on a finished model within the same iteration. Instead, the ML team should run ahead, first delivering a Demo API that fixes the integration contracts, then advancing toward a Minimum Viable Model (MVM) and subsequent increments.

2.3.2 Artifacts. Three artifacts operationalize these practices: Data Stories, Model Stories, and LoD. Data and Model Stories close the gap left by User Stories, which offer no structure for data acquisition goals, model training objectives, or experimental acceptance criteria [7, 10]: a Data Story specifies the goal, source, processing steps, and readiness criteria of a data pipeline, while a Model Story specifies the learning objective, model type, data dependency, and expected performance threshold. LoDs, analogous to Definitions of Ready and Done, are checklists of model maturity milestones [8]. We suggest using at least two layers: a Demo API and an MVM. The Demo API [10] (LoD 0) respects the agreed input and output contracts and returns plausible outputs before an optimized model exists, unblocking software integration, and the MVM [8] is a performance threshold negotiated with business stakeholders at which the module already delivers measurable value, letting the team iterate in production instead of optimizing indefinitely, with further layers marking later milestones.

2.3.3 Roles. Writing Model and Data Stories is a shared responsibility. A professional combining business expertise, ML literacy, and SW engineering knowledge would be almost a “unicorn”, so the ML engineering team should support the PO in writing these stories, while the PO remains sufficiently literate in ML to lead the conversation and grasp the particulars of the ML lifecycle.

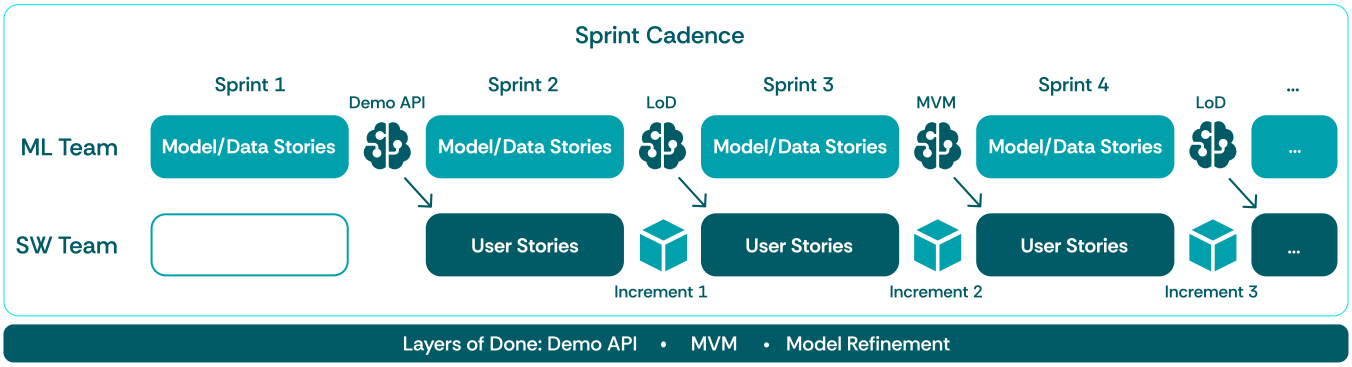

Figure 1: Sprint cadence showing ML and software team collaboration.

3 A Vision on Managing FM-based Agentic Systems Development

Although the lessons learned centered on ML, our current vision shifts toward managing systems that integrate FM-based agents. This shift changes backlog items when viewed from an agile management perspective. Data Stories remain relevant, but they now capture the work of collecting and structuring the context that the agent consumes, rather than the creation of a training dataset. Model Stories lose their direct applicability, since the effort no longer involves training or adapting a model. Traditional User Stories, in turn, offer no structure for expressing agent requirements. This gap motivates the development of a dedicated story type for agentic work, which we call Agent Stories.

In the following, we organize our vision for managing FM-based agentic systems development through an agile lens, structured around practices, artifacts, and roles.

3.1 Practices

Align a shared initial vision on agent requirements before building the backlog. As with ML-enabled systems, the team and PO should establish a common understanding before decomposing the work into backlog items or starting development. Agent specification techniques can support this alignment by translating business intent into explicit agreements on goals, action capabilities, and acceptance thresholds, agreements that later give structure to the backlog and development itself. One example of such support is PerSpecAgents [1], a perspective-based approach for specifying FM-based agentic systems.

Evolve agents incrementally through experimentation. Rather than attempting to deliver a fully capable agent at once, teams should refine it through sprint-bound experimentation, progressively expanding tools, skills, memory mechanisms, reasoning strategies, input modality, and workflows as outcomes are validated. AI Engineers can begin with a minimum agent devoted to a single purpose and then layer in capabilities and orchestration logic as the system matures and requirements become better understood. This incremental philosophy fits naturally with an experimental agent development approach, where each sprint comprises an experiment that informs the next.

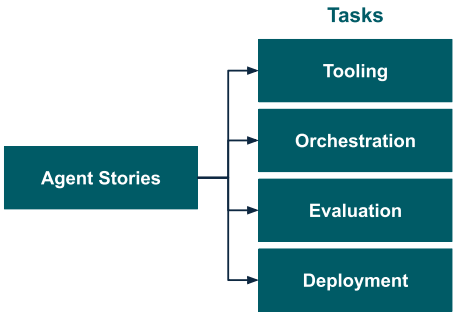

Figure 2: Agent story proposal and related tasks.

Offset the cadence between agent and conventional software work. The cadence presented in Figure 1 also applies in the context of agent development. It follows experimental cycles whose duration is difficult to predict, whereas the surrounding non-agentic features evolve at a more regular pace. Offsetting the two cadences accommodates the experimental nature of agent development without blocking product delivery, while LoD, such as an Agent Demo API and an MVA, allows both teams to progress in parallel.

3.2 Artifacts

Specify agent-intensive agile requirements with Agent Stories and Data Stories. Inspired by the concept of Model and Data Stories of Vaidhyanathan et al. [10], Agent Stories capture the work of tooling, orchestrating, evaluating, and deploying FM-based agents. It should consist of: 1) the agent goal, 2) the possible tasks, 3) the available artifacts (*i.e.* tools, skills, and external spaces), 4) the autonomy level and human interference, and 5) the expected performance threshold on a curated evaluation suite.

An Agent Story could embed four task categories (Figure 2). Tooling defines the agent's capabilities, comprising the operational opportunities (tools), the procedural knowledge it can load (skills), and the context and memory schemas. Orchestration shapes behavior through prompts, planning loops, workflow patterns, guardrails, and error recovery. Evaluation tasks manage the agent acceptance criteria, and Deployment operationalizes the agent serving and system integration.

To illustrate, consider the following example on customer support triage: "As a support triage agent, it should resolve customer refund requests end-to-end [goal] by classifying the request, verifying eligibility against the refund policy, issuing the refund, and drafting the customer reply [tasks]. It should use a ticketing and payment tool and the refund policy skill [artifacts], while requiring explicit human approval for all actions [autonomy level] with a pass rate of at least 90% on the evaluation suite [expected performance]."

Supplementing Agent Stories, Data Stories remain in the backlog with a different focus than ML. Rather than capturing the creation of a training dataset, they should focus on collecting and structuring knowledge sources and context for the agent, such as policy documents, retrieval corpora, and curated examples that feed the evaluation suite. Returning to the customer support triage example, a Data Story could cover gathering the refund policy documents, structuring them into the refund policy skill, and assembling representative refund requests for evaluation. The stories thus complement each other: Agent Stories specify what the agent must do and under which conditions, while Data Stories address agent's required knowledge sources, context, and evaluation suite.

Version agents through LoD. Similar to ML development [8], LoD defines versioning, expressed as checklists that a given agent must satisfy. A first possible milestone emerges before a capable agent even exists. The AI and software engineers agree on the input and output rules for the agentic module and tools, establishing the AI team's first responsibility: delivering an Agent Demo API (LoD 0). Its purpose is to unblock the software team, allowing integration work to proceed in parallel with ongoing agent development. Another important layer addresses early value delivery. Since pursuing the best evaluation metrics is often a long and intensive task, a viable business path is to negotiate with business stakeholders the minimum acceptable threshold, marking the MVA LoD, at which the agent already produces an impactful, yet not perfect, business outcome, mirroring the role of the MVM.

Beyond these proposed entitled layers, LoD for agents should embed elements such as prompt and tool versioning, evaluation thresholds met, bias and safety review, guardrails in place, and instrumented observability. Teams can verify these milestones using measures suited to nondeterministic outputs, such as faithfulness checks based on Natural Language Inference (NLI), RAGAS scores (*a RAG evaluation framework*), or token-level log probabilities, which help manage hallucinations and enforce structured outputs. These measures bring the acceptance criteria closer to those of conventional software acceptance testing while preserving the experimental cycles inherent in nondeterministic components.

3.3 Roles

Support the PO with AI Engineering expertise. As for ML-enabled systems, the AI Engineering team should support the PO in writing Agent and Data Stories, much as software engineers already support backlog refinement for User Stories. This collaboration, however, does not relieve the PO of the need to be AI-literate. The aim is a productive middle ground: a PO is AI-literate enough to lead the conversation, particularly in negotiating autonomy levels and acceptance thresholds with business stakeholders, and is supported by engineers who provide the depth that full specification requires.

4 Concluding Remarks

FM-based agentic systems pose management challenges that remain unaddressed. In this vision paper, we present a first effort in this direction. Therefore, we draw on lessons from agile management of ML-enabled systems and extend them to the particularities of FM-based agent development, organizing our vision around practices, artifacts, and roles.

For practices, we suggest that teams refine agents through sprint-bound experimentation, guided by an initial requirements specification and an offset cadence between agent and conventional software work. For artifacts, we propose specifying agent-intensive requirements through Agent and Data Stories, with LoD marking maturity milestones such as Agent Demo APIs and MVAs. For roles, we advocate that the AI Engineering team support the PO in writing agent-intensive stories.

We position the contribution of our vision not as a prescriptive methodology but as a conceptual foundation with initial guidance that can serve as a basis for future empirical research.

Artifact Availability

No external software artifacts are associated with this work. Guidance and research directions are fully described in the paper.

Acknowledgements

We gratefully acknowledge the support of CAPES (Finance Code 001 and PROEX), CNPq (Grants 312275/2023-4 and 420191/2025-9), FAPERJ (Grant E-26/204.256/2024), and Instituto Kunumi for their generous support.

References

[1] Júlia Condé Araújo, Marina Condé Araújo, José MC Boaro, and Marcos Kalinowski. 2026. Towards an Approach for Specifying Intelligent Systems Involving Foundation Model Based Agents. In *5th International Conference on AI Engineering – CAIN'26* (Rio de Janeiro, Brazil).

[2] Rishi Bommasani et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[3] Martin Fowler, Jim Highsmith, et al. 2001. The agile manifesto. *Software development* 9, 8 (2001), 28–35.

[4] Christian Kästner. 2022. *Machine Learning in Production: From Models to Products.*

[5] Marco Kuhrmann, Paolo Tell, et al. 2021. What Makes Agile Software Development Agile? *IEEE Transactions on Software Engineering* 48 (2021).

[6] Mirko Perkusich, Danyllo Albuquerque, Allysson Allex Araújo, et al. 2026. Adoption of Large Language Models in Scrum Management: Insights from Brazilian Practitioners. In *Agile Processes in Software Engineering and Extreme Programming*. Springer Nature Switzerland.

[7] Lucas Romao, Hugo Villamizar, Romeu Oliveira, Silvio Alonso, and Marcos Kalinowski. 2026. Agile Management for Machine Learning: A Systematic Mapping Study. In *Software Engineering and Advanced Applications*, Davide Taibi and Darja Smite (Eds.). Springer Nature Switzerland.

[8] Lucas Romao, Luiz Xavier, Júlia Condé Araújo, Marina Condé Araújo, Ariane Rodrigues, and Marcos Kalinowski. 2026. Applying a Requirements-Focused Agile Management Approach for Machine Learning-Enabled Systems. In *5th International Conference on AI Engineering – CAIN* (Rio de Janeiro, Brazil).

[9] Davide Taibi, Henry Muccini, Karthik Vaidhyanathan, Marcos Kalinowski, et al. 2026. A research agenda on agents and software engineering: Outcomes from the rio A2SE seminar. *arXiv preprint arXiv:2605.11720* (2026).

[10] Karthik Vaidhyanathan, Anish Chandran, Henry Muccini, and Regi Roy. 2022. Agile4MLS—Leveraging Agile Practices for Developing Machine Learning-Enabled Systems: An Industrial Experience. *IEEE Software* 39 (2022).

[11] Lei Wang et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[12] Zheying Zhang et al. 2025. AI and Agile Software Development: A Research Roadmap from the XP2025 Workshop. *arXiv preprint arXiv:2508.20563* (2025).