

A HUB-ORCHESTRATED, ROLE-AWARE AGENTIC AI FRAMEWORK FOR COMMUNICATION AND COORDINATION IN HYBRID AGILE SOFTWARE TEAMS: A CONCEPTUAL FRAMEWORK

Sumaira Hussain^{*1}, Dr. Muhammad Zulqarnain Siddiqui², Mubbashir Ahmed³

^{*1,2,3}Iqra University Department of Computer Science

¹sumaira.21385_n@iqra.edu.pk, ²zulqarnain@iqra.edu.pk, ³mubbashir.ahmed@iqra.edu.pk

DOI: <https://doi.org/10.5281/zenodo.21819609>

Keywords

Agile software development, hybrid teams, agentic AI, multi-agent systems, large language models, requirements communication, coordination frameworks

Article History

Received: 11 March 2026

Accepted: 15 April 2026

Published: 30 April 2026

Copyright @Author

Corresponding Author: *
Sumaira Hussain

Abstract

Hybrid Agile teams integrates co-located and remote members working in the same development process. These teams often lose shared situational awareness, because information that would normally pass through informal, in-person conversation reaches team members inconsistently. Existing collaboration tools Jira, Slack, Microsoft Teams record this communication, but they do not interpret it. Team members are still left to judge, prioritize, and reason across roles on their own.

Current AI-based approaches do not fill this gap. Most target a single software-engineering task, such as defect prediction, code assistance, or meeting summarization. LLM-based multi-agent systems go further, but they are built to produce software, not to manage team communication itself.

This paper proposes a Hub-Orchestrated, Role-Aware Agentic AI Framework to address this gap. The framework uses a layered architecture with four core steps. First, it collects multi-modal Agile communication from multiple sources. Second, it converts this communication into structured, traceable requirements. Third, six role-specific agents Product Owner, Developer, QA, UX/Design, DevOps, and Scrum Master analyze the requirements in parallel. Fourth, a central coordination engine resolves the agents' outputs: it detects misalignment, resolves dependencies, and generates role-specific recommendations.

We describe this architecture in enough technical detail for independent implementation. We also illustrate its intended behavior through a conceptual application to three representative communication breakdowns, drawn from real-world Agile issue-tracking data recorded in the public Apache Jira data set.

The contribution of this paper is architectural and conceptual. We report no working implementation or empirical evaluation here. The illustrative cases demonstrate the framework's applicability; they do not validate it. We close by outlining the prototype and evaluation plan needed to test the framework's claims empirically.

1. INTRODUCTION

1.1 Background

Communication is the foundation of Agile software development. Agile teams depend on continuous interaction among Developers, Scrum Masters, Product Owners, and stakeholders to keep project goals aligned. Practices such as daily stand-ups, sprint planning, sprint reviews, and retrospectives happens specially to support this continuous communication.

This architecture works well in a co-located team (All team members are physically present). It becomes harder in a hybrid (onsite and remote team members), where some members work remotely and others work in person. Hybrid teams face three recurring problems, documented through recent research:

- Remote members have less access to the informal, spontaneous discussions that co-located members share [1], [2], [3].
- Hybrid meetings often reduce engagement for remote participants and give them less influence over decisions [3], [5].
- Trust, communication transparency, and collaboration quality become harder to maintain as teams disperse [3], [6].

Agile collaboration tools such as Jira, Slack, and Microsoft Teams support information exchange, but they do not interpret it. Team members must still manually read, organize, and prioritize communication. As communication volume grows, this manual interpretation becomes a source of rework and delay in its own right [1], [6].

Hybrid Agile teams need a proper way to manage communication and coordination that goes beyond simply recording it. Existing collaboration tools do not detect ambiguity, do not reason across the different needs of different Agile roles, and do not reconcile conflicting information automatically. This gap converts more significant as hybrid work environments become more common these days.

Artificial intelligence has already been used extensively in software engineering, but mostly to single tasks that is detect prediction and software

analytics [22]–[25], natural language processing over requirements and issue text [24], meeting summarization [26], and AI-assisted coding tools [27]. These tools are effective within their own scope, but none of them reason jointly across multiple stakeholder roles and perspectives.

Large language models have empowered a further class of system: agentic AI, which reasons, plans, and performs rather than only generating text [7]–[10], and which increasingly uses memory and external tools [11]–[14]. These agentic capabilities have been organized into multi-agent systems such as AutoGen, MetaGPT, CAMEL, AgentVerse, and ChatDev [16]–[20]. These systems exhibit that assigning specialized roles to agents improves collaboration and task quality. However, they are built for software production tasks i.e coding, testing, project planning but not for communication and coordination among a human Agile team, and they do not centrally orchestrate agents around the responsibilities of Product Owners, Developers, QA, UX, DevOps, and Scrum Masters [21].

None of the existing framework combines: (a) centralized orchestration for global coordination, (b) Role-aware agents aligned with real Agile responsibilities, (c) LLM-based reasoning applied to communication understanding, and (d) explicit support for the multi-platform, multi-modal communication environment of hybrid Agile teams. This is the gap addressed by the present research and the Research Questions are

- **RQ1.** What communication and coordination challenges do hybrid Agile teams face?
- **RQ2.** How can a Role-aware Agentic AI framework address communication misalignment among Agile team members?
- **RQ3.** What role does a hub-orchestrated coordination mechanism play in improving collaboration and alignment among Agile roles?
- **RQ4.** To what extent could the proposed framework improve communication and coordination compared with traditional Agile approaches?

1.2 Contribution

This paper proposes the Hub-Orchestrated Role-Aware Agentic AI Framework and defines its six architectural layers in ample detail to support independent implementation. The paper's contribution is conceptual: it presents the framework's design and illustrates its intended behavior through a case-based application to real communication data, rather than reporting a working implementation or an empirical evaluation.

2. Related Work

2.1 Agile Communication and Hybrid Teams

Distributed Agile teams normally face communication delays, information-sharing challenges, and coordination overhead, which can lead to misunderstandings, duplicated work, and reduced team cohesion [1]. Coordination complexity increases further as team size and project dependencies increases [4].

A related concept is shared situational awareness: the team's collective understanding of project status, ongoing activities, and emerging issues. When communication channels are fragmented, team members can lose visibility into project developments, which leads to coordination breakdowns and delayed decisions [4], [5].

Agile tools such as Jira, Slack, and Microsoft Teams support information exchange, but they do not guarantee effective coordination. Team members must still manually interpret and prioritize communication, which becomes a notable workload in complex projects [1], [6].

Hybrid Agile teams, combining co-located and remote members, introduce additional challenges. Remote members often lack access to the informal, spontaneous interactions available to co-located colleagues, so important contextual information is not always shared equally [2], [3]. Hybrid meetings continuously show reduced engagement among remote participants and challenges in maintaining equal participation [3]. Remote participants often have lower visibility and participation in decision-making than on-site colleagues [5]. Trust, collaboration quality, and

communication transparency become harder to sustain as teams become more distributed [3], [6]. Existing research on hybrid Agile teams largely identifies these challenges rather than proposing intelligent tools to address them [2].

2.2 Artificial Intelligence in Software Engineering

AI techniques are now used across the software development lifecycle, including requirements engineering, defect prediction, effort estimation, testing, project management, and developer support. Early machine-learning research in software engineering established the foundation for defect prediction, quality assessment, and project analytics [23], [25].

AI also supports analysis of large volumes of project data. Software analytics research shows that data-driven insights can improve decision-making, project visibility, and risk management [22], [24]. Natural language processing enables automated analysis of requirements documents, issue reports, user stories, and communication logs, creating opportunities to extract insights from Agile communication data [24]. Automated summarization can reduce information overload from meetings and discussions [26], and AI-assisted development tools can increase developer productivity [27].

Despite these advances, most AI applications in software engineering remain task-specific. They focus on individual activities – coding assistance, document analysis, effort estimation – rather than team-level communication and coordination.

2.3 Large Language Models and Agentic AI

Large language models such as GPT-3 and GPT-4 have broadened what AI can do in natural language understanding, generation, reasoning, and summarization [7], [8]. Beyond text generation, agentic AI systems combine reasoning, planning, memory, and action. The ReAct framework shows how language models can combine reasoning with task execution [9]. Reflexion introduces self-improvement through

reflective learning, allowing agents to adapt based on experience [10].

Surveys of agentic AI show increasing use of memory structures, planning, and tool integration in language agents [11], [12]. Generative Agents explain how memory and contextual awareness support autonomous behavior over extended interactions [13]. Toolformer extends these systems by allowing language models to use external tools [14].

Most Agentic AI systems are built for general-purpose problem solving instead of structured organizational coordination. Their application to Agile communication and team coordination remains undiscovered.

2.4 Multi-Agent Systems and Role-Aware Agents

Multi-agent systems allow multiple autonomous agents to collaborate toward shared goals while keeping specific responsibilities. This supports task decomposition, distributed problem solving, and coordinated decision-making [15].

Combining multi-agent architecture with large language models has produced capable collaborative systems. AutoGen presents structured communication among multiple language-model agents [16]. AgentVerse studies large-scale agent collaboration and emergent behavior [19].

Role specialization is an important theme in these systems. MetaGPT and CAMEL show that assigning specialized roles to agents improves collaboration efficiency, task quality, and communication effectiveness [17], [18]. ChatDev extends this idea into software engineering by organizing agents into role-based development teams [20].

These frameworks focus on software production tasks such as coding, testing, and project planning. Few of them address communication and coordination challenges within hybrid Agile teams directly, and existing frameworks typically lack a centralized orchestration mechanism that

maintains global awareness and alignment among agents [21].

2.5 Summary and Research Gap

The literature reviewed above shows progress in four areas: Agile communication and hybrid teams, AI in software engineering, agentic AI, and multi-agent systems. However, four limitations remain:

1. Research on Agile communication and hybrid teams mostly detects challenges rather than proposing intelligent coordination tools [1]–[6].

2. AI applications in software engineering mostly target individual tasks rather than team-level communication and coordination [22]–[27].

3. Agentic AI research delivers strong reasoning, planning, memory, and tool-use capabilities, but mostly for standalone agents rather than structured organizational settings [7]–[14].

4. Multi-agent frameworks demonstrate the value of role specialization and collaboration, but they primarily support software production, with limited support for communication governance and hybrid team coordination [15]–[21].

No existing framework combines centralized orchestration, role-aware agents aligned with Agile responsibilities, and LLM-based reasoning for communication understanding, specifically for hybrid Agile teams. This gap forms the basis for the framework proposed in this paper.

3. Proposed Framework

The proposed Hub-Orchestrated Role-Aware Agentic AI Framework combines role-aware AI agents, centralized coordination, communication analysis, and continuous feedback. It is organized into six layers, described in the subsections below, and is supported throughout by a Large Language Model, a Knowledge Base, and Security and Privacy mechanisms. Figure 1 shows the overall architecture.

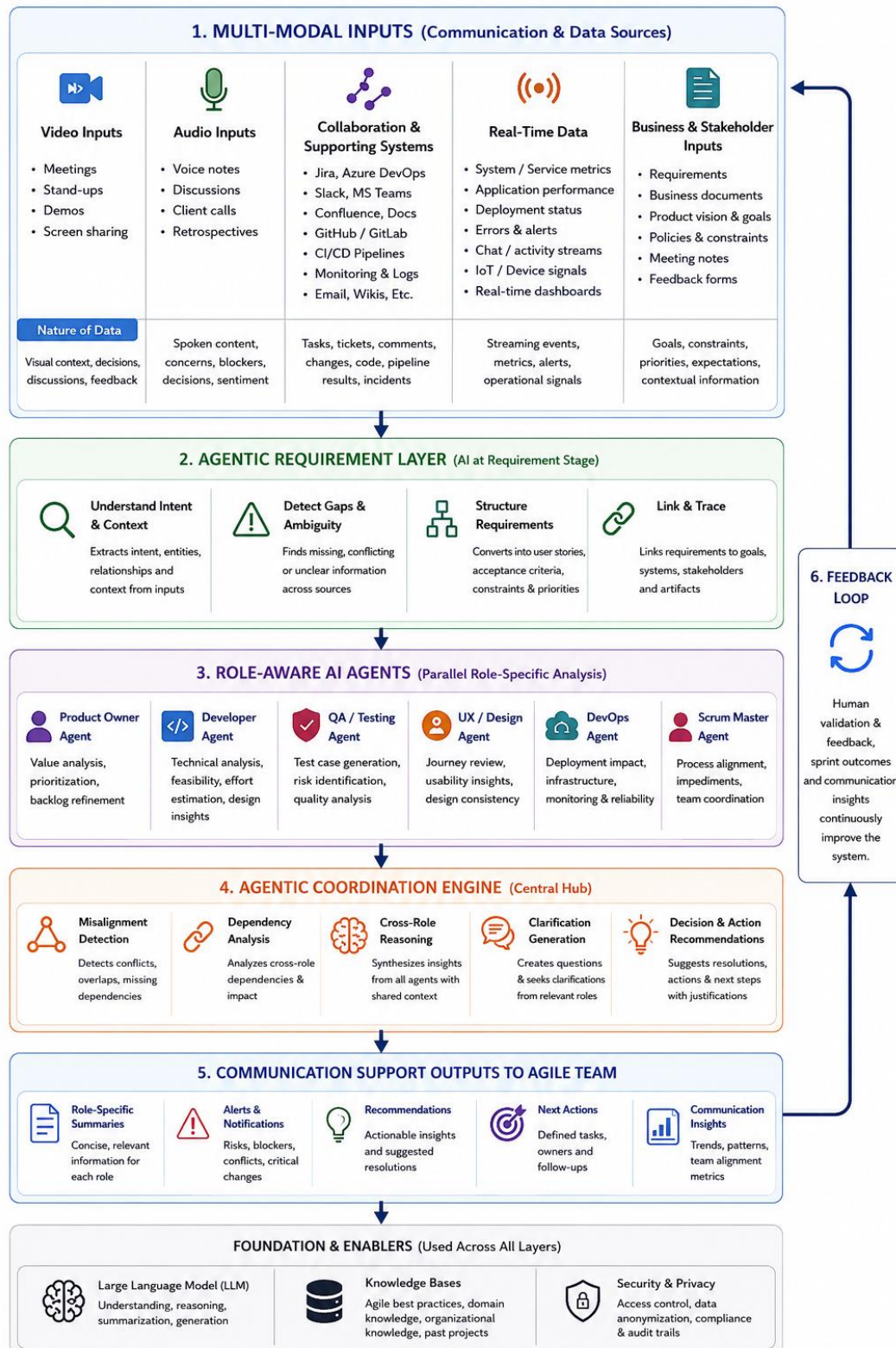


Figure 1. Overall architecture of the proposed Hub-Orchestrated Role-Aware Agentic AI Framework.

Communication artifacts coming from meetings, collaboration platforms, issue-tracking systems, and stakeholder interactions are first collected through the Multi-Modal Inputs Layer. These inputs are then processed by the Agentic Requirement Layer, which identifies communication intent, detects ambiguities, structures requirements, and maintains traceability. The extracted information is analyzed by the Role-Aware AI Agents, while the Agentic Coordination Engine acts as a central hub that integrates agent outputs, resolves conflicts, and maintains overall communication awareness. The framework produces role-specific summaries,

recommendations, alerts, and communication insights, supported by a continuous feedback loop and by the Large Language Model, Knowledge Base, and Security and Privacy components shown at the bottom of Figure 1.

3.1 Multi-Modal Inputs Layer

This layer is the entry point of the framework. Its purpose is to collect, integrate, and organize information from the multiple communication and collaboration channels used in hybrid Agile teams. Table 1 summarizes the input categories the framework supports.

Table 1. Multi-Modal Inputs Supported by the Framework

Input Category	Examples	Purpose
Video Inputs	Sprint reviews, planning meetings, stand-up meetings, demonstrations	Capture visual communication and discussion context
Audio Inputs	Voice discussions, retrospectives, recorded meetings	Capture spoken information and team interactions
Collaboration Systems	Jira, Slack, Microsoft Teams, GitHub, Azure DevOps	Collect project-related communication, issues, and collaboration data
Real-Time Sources	Project dashboards, alerts, notifications, operational metrics	Provide current project status and activity information
Business Inputs	Requirements documents, stakeholder feedback, business objectives	Capture strategic and organizational context

By integrating information from these sources, the Multi-Modal Inputs Layer establishes a comprehensive communication groundwork for the rest of the framework.

3.2 Agentic Requirement Layer

Communication artifacts collected by the Multi-Modal Inputs Layer are often incomplete,

ambiguous, or unstructured. The Agentic Requirement Layer transforms these raw inputs into structured, analyzable, and traceable information for use by the role-aware agents and the coordination engine. Table 2 summarizes its main functions.

Table 2. Functions of the Agentic Requirement Layer

Function	Description
Understand Intent and Context	Identifies stakeholder objectives, discussion context, and communication purpose
Detect Gaps and Ambiguity	Identifies missing details, unclear statements, conflicting information, and incomplete requirements
Structure Requirements	Organizes extracted information into a structured format suitable for analysis, traceability, and decision support

Function	Description
Link and Trace Information	Maintains relationships among requirements, discussions, issues, stakeholders, and project artifacts

By changing unstructured communication into structured requirements, this layer improves information quality and supports more effective coordination among Agile roles.

3.3 Role-Aware AI Agents Layer

Communication and coordination challenges affect different Agile roles differently.

Information important to a Product Owner may vary from what a Developer, QA Engineer, DevOps Engineer, or Scrum Master needs. The framework therefore includes six role-aware agents, summarized in Table 3.

Table 3. Role-Aware AI Agents and Responsibilities

Agent	Primary Responsibility
Product Owner Agent	Evaluates business objectives, stakeholder expectations, requirement priorities, and value delivery
Developer Agent	Examines technical feasibility, implementation concerns, dependencies, and development risks
QA/Testing Agent	Identifies quality concerns, testing requirements, validation needs, and potential defects
UX/Design Agent	Evaluates usability, user experience, interface consistency, and design concerns
DevOps Agent	Analyzes deployment requirements, infrastructure dependencies, operational considerations, and release readiness
Scrum Master Agent	Monitors team coordination, communication effectiveness, process alignment, and workflow impediments

The role-aware agents operate in parallel, analyzing the structured information from the Agentic Requirement Layer. Each agent generates observations and recommendations based on its own responsibilities. This distributed analysis captures multiple perspectives at the same time and reduces the risk of overlooking important communication details.

3.4 Agentic Coordination Engine

After the role-aware agents complete their individual analyses, the Agentic Coordination Engine consolidates their outputs, identifies relationships among role-specific observations, and generates coordinated insights. Its purpose is to turn multiple role-specific perspectives into one unified understanding of a communication situation. Table 4 summarizes its main functions.

Table 4. Functions of the Agentic Coordination Engine

Function	Purpose
Misalignment Detection	Identifies inconsistencies, conflicting viewpoints, and differences in stakeholder understanding
Dependency Analysis	Examines relationships and dependencies among requirements, tasks, roles, and project activities
Cross-Role Reasoning	Integrates insights from different role-aware agents into a comprehensive understanding of the situation

Function	Purpose
Clarification Generation	Produces clarification requests and recommendations when communication gaps or ambiguities are detected
Decision Support Recommendations	Generates coordinated suggestions to help stakeholders address communication and coordination challenges

The coordination engine does not replace the role-aware agents. It acts as an intermediary that synthesizes their findings into meaningful communication support outputs, reducing information fragmentation and improving visibility across Agile roles.

3.5 Communication Support Outputs Layer

After the Agentic Coordination Engine produces its consolidated insights, this layer transforms them into outputs that stakeholders can act on directly, rather than requiring team members to manually interpret large volumes of raw communication data. Table 5 summarizes these outputs.

Table 5. Communication Support Outputs

Output Category	Purpose
Role-Specific Summaries	Provide tailored summaries relevant to the responsibilities and information needs of individual Agile roles
Alerts and Notifications	Highlight communication risks, blockers, missing information, unresolved dependencies, and emerging issues
Recommendations	Suggest actions that may improve communication clarity, coordination effectiveness, and stakeholder alignment
Next Actions	Identify follow-up activities and responsibilities required to address outstanding concerns
Communication Insights	Present consolidated observations regarding communication patterns and coordination challenges

3.6 Supporting Components and Feedback Loop

Three supporting components function across all layers of the framework. The Large Language Model provides natural language understanding, reasoning, summarization, and communication analysis. The Knowledge Base keeps project knowledge, organizational context, historical communication records, and domain-specific references. Security and Privacy mechanisms support access control, confidentiality, and responsible handling of communication data.

The framework also includes a feedback loop, in which human validation, sprint outcomes, and communication insights are collected and used to refine future analyses. This feedback loop is intended to support continuous improvement of the framework over time; the present paper

describes it at the level of design intent rather than a fully specified mechanism.

4. Evaluation

The following cases are a conceptual application of the framework, not an empirical evaluation. No implementation of the framework was made or run against this data. The reasoning attributed to each agent below describes how the framework in Section 3 is intended to behave, not measured output from a working system. All three cases are drawn from real, publicly available Agile project communication recorded in the Apache Jira issue-tracking dataset, and were chosen to represent three distinct communication challenges observed in that real-world data.

4.1 Case 1: Communication Ambiguity and Missing Information (BEEHIVE-145)

A stakeholder reports being unable to reproduce a reported problem from the information given. Under the proposed framework, the Agentic Requirement Layer would detect missing reproduction steps and environment details as an information gap. The Developer and QA agents would each identify that the available information is unsatisfactory. The Coordination Engine would generate a clarification request asking the reporter for detailed reproduction steps and system configuration.

4.2 Case 2: Technical Blocker and Collaborative Problem Resolution (COCOON-911)

A comment reports that a bug has just been encountered and that a fix will be attempted. Here, the Developer agent would identify implementation concerns, the QA agent would evaluate testing implications, and the DevOps agent would consider operational impact. The Coordination Engine would consolidate these three technical perspectives into a single recommendation: evaluate the proposed solution

and verify technical feasibility before implementation, then perform validation testing.

4.3 Case 3: Requirement Clarification and Stakeholder Alignment (MNG-309)

A comment asks under what conditions an email notification should be triggered, a business rule that was not clearly defined. The Product Owner agent would identify the missing business requirement, the Developer agent would identify the resulting implementation uncertainty, and the Scrum Master agent would identify the need for stakeholder coordination. The Coordination Engine would generate a clarification request directed at the Product Owner or relevant business stakeholder, along with a recommendation to document the notification rule before implementation continues.

4.4 Comparison with Traditional Agile Communication Practices

Table 6 compares how these types of communication challenges are typically handled in traditional Agile environments with how the proposed framework is intended to handle them.

Table 6. Comparison of Traditional Agile Communication and the Proposed Framework

Evaluation Criterion	Traditional Agile Approach	Proposed Framework
Communication Clarity	Ambiguities are identified manually through stakeholder discussions and reviews	Communication gaps and ambiguities are detected through the Agentic Requirement Layer
Requirement Understanding	Understanding depends on individual stakeholder interpretation and experience	Requirements are analyzed, structured, and linked through contextual and role-aware processing
Role Awareness	Stakeholders evaluate issues primarily from their own perspective	Role-aware agents analyze communication from multiple Agile role perspectives at the same time
Coordination Support	Coordination relies on meetings, follow-ups, and manual communication	The Coordination Engine is intended to support stakeholder alignment and coordination directly
Clarification Support	Clarifications are requested manually after communication issues are discovered	Clarification requests are intended to be generated automatically when missing or conflicting information is detected
Decision Support	Decisions are primarily based on stakeholder judgment and available documentation	Recommendations are intended to be generated through cross-role reasoning and coordinated analysis

Evaluation Criterion	Traditional Agile Approach	Proposed Framework
Dependency Awareness	Dependencies may be identified late, through project discussions or issue tracking	Dependencies are intended to be identified through centralized coordination

In traditional Agile environments, identifying communication gaps and coordination issues depends on manual effort and stakeholder awareness, so misunderstandings and delays can persist until team members notice and address them. The proposed framework is designed to detect these issues earlier and more consistently, though whether it does so in practice has not yet been tested.

5. Discussion

The framework's design supports the research questions stated in Section 1.4. Its role-aware structure (RQ2) allows communication to be analyzed from multiple stakeholder perspectives at once, rather than from a single point of view. Its centralized coordination mechanism (RQ3) is designed to unite these perspectives, identify gaps, and generate clarification requests, which is intended to support alignment and reduce communication fragmentation across Agile roles. The comparison in Table 6 (RQ4) directs where the framework is designed to add capabilities like automated ambiguity detection, role-aware reasoning, clarification generation, coordination assistance that traditional Agile communication practices do not provide by default.

Several design strengths follow from the architecture in Section 3. First, the framework provides a structured approach for analyzing communication from different sources. Second, the role-aware structure allows multiple stakeholder perspectives to be considered at the same time. Third, the centralized coordination mechanism is designed to support cross-role reasoning and stakeholder alignment. Fourth, the framework is designed to generate actionable outputs recommendations, clarification requests, and communication insights intended to help Agile teams address communication challenges.

Several limitations should be acknowledged. First, this paper presents a conceptual application of the framework rather than a working software prototype; the illustrative cases in Section 4 demonstrate intended reasoning, not measured system behavior. Second, the illustrative cases are drawn from a single dataset (Apache Jira issue comments) and a small number of examples, so they show applicability to particular challenge types rather than general effectiveness across all Agile communication. Third, the feedback loop described in Section 3.6 is specified at the level of design intent, not a fully worked-out mechanism. Fourth, the framework has not been deployed or tested in an industrial hybrid Agile environment, so its practical performance under real communication volume remains unknown.

6. Conclusion and Future Work

This paper proposed a Hub-Orchestrated Role-Aware Agentic AI Framework to support communication and coordination in hybrid Agile teams. The framework combines a Multi-Modal Inputs Layer, an Agentic Requirement Layer, a Role-Aware AI Agents Layer, an Agentic Coordination Engine, a Communication Support Outputs Layer, and a feedback loop, supported by a Large Language Model, a Knowledge Base, and Security and Privacy mechanisms. This design addresses a gap identified across research on hybrid Agile communication, AI in software engineering, agentic AI, and multi-agent systems: no existing approach combines centralized orchestration, Agile-role-aware agents, and LLM-based reasoning specifically for communication and coordination in hybrid Agile teams.

The framework was illustrated through a conceptual application to three representative communication challenges that is communication ambiguity and missing

information, a technical blocker requiring collaborative resolution, and requirement clarification and stakeholder alignment each taken from real Agile project communication in the Apache Jira dataset, and compared against traditional Agile communication practices. This illustrative application demonstrates the framework's intended applicability; it does not constitute an empirical evaluation.

Future work should proceed in the following order. First, a working prototype should implement the six layers described in Section 3, tested initially on the same illustrative cases used in this paper, so that intended behavior can be compared against actual system output. Second, the framework should be evaluated on a larger and more diverse set of Agile communication data, ideally with human-rated ground truth for what a correct clarification or recommendation looks like. Third, the framework should be integrated with live Agile collaboration platforms rather than static datasets, to test it under real communication volume. Fourth, the feedback loop described in Section 3.6 needs a concrete mechanism, for example logging human acceptance or rejection of generated recommendations and using this feedback to refine the framework's future analyses. Only after these steps would it be appropriate to describe the framework's objectives as achieved rather than addressed at a conceptual level.

REFERENCES

- [1] L. Haslinger, H. Sikora, and R. Riedl, "Problem Bereiche verteilter agiler Teams: Literaturanalyse und Praxisimplikationen," *HMD Praxis der Wirtschaftsinformatik*, vol. 57, no. 5, pp. 1014-1032, 2020.
- [2] D. Khanna, E. L. Christensen, S. Gosu, X. Wang, and M. Paasivaara, "Hybrid Work meets Agile Software Development: A Systematic Mapping Study," in *Proc. IEEE/ACM Int. Conf. Cooperative and Human Aspects of Software Engineering (CHASE)*, 2024.
- [3] E. L. Christensen, M. Paasivaara, and I. Salman, "Hybrid Work in Agile Software Development: Recurring Meetings," in *Proc. IEEE/ACM Int. Conf. Cooperative and Human Aspects of Software Engineering (CHASE)*, 2025.
- [4] M. Mikalsen, N. B. Moe, S. I. Wong, and V. Stray, "Agile Information System Development Organizations Transforming to Large-Scale Collaboration," arXiv preprint arXiv:2111.06193, 2021.
- [5] V. Stray, "Hybrid Meetings in Agile Software Development," in *Agile Processes in Software Engineering and Extreme Programming Workshops (XP Workshops)*, 2024.
- [6] F. Broomandi, M. Paasivaara, E. L. Christensen, S. Hyrynsalmi, and D. Khanna, "Hybrid Work in Agile Software Engineering: Current Research and Future Directions," in *Agile Processes in Software Engineering and Extreme Programming Workshops, Lecture Notes in Business Information Processing*, vol. 561, pp. 53-67, 2025.
- [7] T. B. Brown et al., "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877-1901, 2020.
- [8] OpenAI, "GPT-4 Technical Report," arXiv:2303.08774, 2023.
- [9] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2023.
- [10] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning," arXiv:2303.11366, 2023.
- [11] L. Wang et al., "A Survey on Large Language Model Based Autonomous Agents," *Frontiers of Computer Science*, vol. 19, no. 1, Art. no. 191601, 2025.
- [12] Z. Xi et al., "The Rise and Potential of Large Language Model Based Agents: A Survey," *ACM Computing Surveys*, 2024.

- [13] J. S. Park, J. O'Brien, C. Cai, M. Ringel Morris, P. Liang, and M. S. Bernstein, "Generative Agents: Interactive Simulacra of Human Behavior," in *Proc. ACM Symposium on User Interface Software and Technology (UIST)*, pp. 1–22, 2023.
- [14] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," in *Advances in Neural Information Processing Systems*, vol. 36, 2023.
- [15] M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed. Hoboken, NJ, USA: John Wiley & Sons, 2020.
- [16] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," in *Proc. Conference on Language Modeling (COLM)*, 2024.
- [17] S. Hong et al., "MetaGPT: Meta Programming for Multi-Agent Collaborative Frameworks," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2024.
- [18] G. Li, H. Hammoud, H. Itani, B. Khizbullin, and B. Ghanem, "CAMEL: Communicative Agents for Mind Exploration of Large Language Model Society," in *NeurIPS Workshop*, 2023.
- [19] Z. Chen, K. Hu, J. Hu, Z. Xu, Z. Chen, Y. Feng, and M. Zhang, "AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors," arXiv:2308.10848, 2023.
- [20] C. Qian, W. Liu, H. Liu, Z. Chen, Y. Liu, and D. Yu, "ChatDev: Communicative Agents for Software Development," in *Proc. Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.
- [21] Y. Guo et al., "Large Language Model-Based Multi-Agent Systems: A Survey of Progress and Challenges," arXiv, 2024.
- [22] T. Menzies and T. Zimmermann, "Software Analytics: So What?," *IEEE Software*, vol. 30, no. 4, pp. 31–37, 2013.
- [23] M. A. Babar and H. Zhang, "Systematic Literature Review of Machine Learning Techniques in Software Engineering," *Information and Software Technology*, 2009.
- [24] L. Liu et al., "Natural Language Processing for Software Engineering: A Systematic Literature Review," 2023.
- [25] T. Zimmermann, N. Nagappan, H. Gall, and B. Murphy, "Cross-Project Defect Prediction: A Large Scale Experiment on Data vs. Domain vs. Process," in *Proc. European Software Engineering Conference and Foundations of Software Engineering (ESEC/FSE)*, pp. 91–100, 2009.
- [26] Y. Liu, A. Stolcke, and E. Shriberg, "Automatic Meeting Summarization: A Survey," *ACM Computing Surveys*.
- [27] P. Fan et al., "An Empirical Evaluation of GitHub Copilot's Impact on Developer Productivity," 2024.