Agile Agents: Evaluating the Performance of LLM Agents Across Software Development Frameworks — Multi-Agents Track (4 Units)

Joergen Jore, Aditya Biswal, Priyan Jindal, Julian Strietzel , Dev Patel

University of California, Berkeley
Fall 2024
Team Space

Abstract

Autonomous agents for software engineering have traditionally operated as isolated entities, each performing narrowly defined tasks with limited interaction. While recent advances in large language models (LLMs) have opened the door to more complex, intelligent agents, a critical question remains: how can these agents be effectively orchestrated to collaborate within real-world software development environments? This paper investigates the feasibility and efficacy of different software engineering paradigms—specifically Agile vs a Single-Agent approach—as frameworks for multi-agent collaboration. Our approach introduces a custom framework built upon SWE-Agent, extending it with multi-agent interaction capabilities. This framework enables the creation and management of interacting agents, each with distinct roles, facilitating collaborative work on shared tasks within a simulated software development environment.

We evaluate these paradigms using the SWE-bench, a repository of real-world software engineering tasks derived from GitHub issues. Our results demonstrate that the Agile agent achieved a 34.78% completion rate (percentage of issues the agent believed it solved) and a 25% success rate (percentage of issues actually solved), compared to the single agent's 64.28% completion rate and 11% success rate. Our findings highlight the significant performance gains achieved by multi-agent frameworks, with Agile outperforming the Single-Agent approach, demonstrating the potential of collaborative agent teams for enhanced software development efficiency. This suggests that not only is the Agile approach more effective in solving problems, but it also demonstrates a more accurate self-assessment of its performance, highlighting the benefits of structured collaboration. By providing empirical evidence that structured, role-based teamwork among LLM-based agents enhances software engineering outcomes, this work lays the foundation for more effective, scalable, and adaptive multi-agent frameworks in the future.

1 INTRODUCTION

Large language model (LLM)-based agents have demonstrated significant potential in tasks such as code completion, bug detection, and documentation generation. These capabilities have positioned LLM agents as transformative tools in software engineering. However, their utility has largely been constrained to isolated tasks, lacking the capacity for effective collaboration in more sophisti- cated software engineering domains. This limitation is underscored in the survey by Liu et al. [1], which highlights the urgent need for research into collaborative mechanisms that could extend the applicability of LLMs.

The recent evolution of LLMs has catalyzed the development of frameworks to enable multi-agent collaboration. Frameworks like ChatDev exemplify how multiple AI agents can assume specialized roles such as design, coding, and testing and engage in structured communication to produce functional software autonomously. These advancements paved the way for a paradigm shift in software development, where AI agents could operate as cohesive teams rather than isolated entities. This research builds upon these innovations by exploring how principles of human team dynamics can be adapted to improve the collaborative efficiency of AI agents. Drawing insights from studies on human leadership and organizational behavior, we aim to design systems that enhance the ability of AI agents to coordinate, share knowledge, and solve problems collectively. By emulating human team structures, these systems promise to produce efficient but also robust and innovative software solutions.

The implications of such advancements are profound. Collaborative LLM agents could revolutionize software engineering by accelerating development cycles, improving code quality, and reducing costs. Moreover, integrating AI-driven collaboration aligns with the industry's increasing reliance on distributed and asynchronous development practices. Our research seeks to lay the foundation for a new era of AI-augmented software engineering by addressing the gap between isolated task performance and team-based problem-solving.

The Value of Agent Collaboration

A critical question arises when considering the application of LLM agents to team-based software development: Is agent collaboration truly valuable, given that agents, unlike humans, do not possess inherent capacity or knowledge limitations?

It is true that individual agents, powered by vast LLMs, theoretically possess access to a vast knowledge base and computational power that far exceeds human capabilities. However, several arguments suggest that agent collaboration might still offer significant advantages: **Specialization and Efficiency:** While a single agent can handle various tasks, specialized agents focused on specific domains (e.g., front-end, back-end, testing) could potentially perform their roles more efficiently. This echoes the division of labor observed in human teams, where specialization leads to increased productivity. OpenHands, for instance, supports the creation of specialized agents through its AgentHub, enabling the development of agents tailored for specific tasks. **Diverse Perspectives**

and Problem-Solving: Collaboration between agents with different strengths and weaknesses could lead to more creative and robust solutions. A team of agents might be able to overcome individual biases or blind spots by leveraging their collective intelligence. This is analogous to human teams, where diverse perspectives often lead to more innovative outcomes. **Scalability and Parallel Processing:** Agent teams can potentially handle larger and more complex projects by distributing tasks and working in parallel. This could significantly accelerate development cycles, enabling the completion of projects that would be infeasible for a single agent to handle within a reasonable timeframe.

Therefore, despite the lack of inherent limitations in individual agents, agent collaboration might still offer valuable benefits in terms of efficiency, problem-solving, and scalability. This research aims to explore this potential by investigating how different team structures and collaboration mechanisms impact the performance of LLM agents in software development.

2 Related Work

ChatDev: A Framework for Multi-Agent Development. ChatDev [2] pioneered the concept of structured multi-agent collaboration in software development. By introducing specialized roles like designers, programmers, and testers, ChatDev demonstrated how multiple AI agents could work together to produce functional software. The framework's approach to role-based task assignment and inter-agent communication protocols provides valuable insights for developing more sophisticated collaborative systems. However, ChatDev's implementation primarily focuses on code generation rather than comprehensive software engineering tasks.

SWE-bench: Evaluating Real-World Performance. The introduction of SWE-bench [3] marked a significant advancement in evaluating LLM-based agents' capabilities. By providing a repository of actual GitHub issues, SWE-bench offers a realistic benchmark for assessing how agents handle real-world software engineering tasks. This benchmark has been particularly valuable in understanding agents' abilities to navigate codebases, implement changes, and work within existing project constraints. The diverse range of issues in SWE-bench, from bug fixes to feature implementations, provides a comprehensive framework for evaluating agent performance.

SWE-agent: Bridging AI and Development Environments. SWE-agent [4] addresses a critical gap in LLM-agent implementation by providing an Agent-Computer Interface (ACI). This interface enables agents to interact with development environments in ways that mirror human developer workflows, from repository navigation to code editing. The ACI's ability to translate high-level agent decisions into concrete development actions represents a significant step toward practical AI-assisted software development. However, the current implementation focuses primarily on single-agent scenarios, leaving open questions about multi-agent coordination.

Integration Challenges and Future Directions. While these frameworks each address crucial aspects of AI-assisted software development, significant challenges remain in combining their

strengths for effective multi-agent collaboration. Recent work [5, 6] has begun exploring various approaches to multi-agent collaboration in software development, but there are still open questions about the most effective ways to organize and coordinate multiple agents. Our work specifically builds upon these foundations by integrating ChatDev's role-based collaboration model with SWE-agent's practical development capabilities, while using SWE-bench for evaluation. This integration aims to address the current limitations in organizing and managing multiple LLM agents to function effectively as a cohesive development team.

3 Implementation

This section provides a technical overview of the implementation of the multi-agent collaborative framework for software development. The implementation is based on a forked version of the **SWE-agent** repository, modified to support multi-agent collaboration by incorporating concepts from the **ChatDev** framework.

3.1 Core Design and Modifications

At the core of the system is an **organization agent** (`OrgAgent`), implemented as a Python class. This agent acts as a central coordinator, orchestrating the actions of specialized sub-agents to collaboratively solve tasks within a software development project. Key modifications to SWE-agent include:

- **Introduction of the Organization Agent:** The `OrgAgent` divides a high-level task into manageable subtasks and delegates them to specialized sub-agents. It will take reports from the `SubAgent`s to guide the progress of the team and further divide work.
- **Definition of Specialized Sub-Agents:** Roles such as *coder*, *tester*, *reviewer*, and *planner* are introduced, each sub-agent equipped with unique capabilities tailored to its role.
- **Dynamic Prompt Generation:** The `OrgAgent` dynamically generates prompts based on the task description and sub-agent capabilities, guiding each sub-agent's action towards task completion.
- **Integration of ChatDev Concepts:** Role-based task assignment and structured communication protocols from ChatDev are adapted to enhance coordination and decision-making.
- **Management of Shared History:** The reporting of agents work towards the `OrgAgent` ensures that their work will get handled appropriately and essential information will be forwarded to the next agent included in the process.
- **Agent Communication and Coordination:** A structured message-passing system facilitates effective communication and task coordination among agents.

3.2 Techniques for History Management and Coordination

Several techniques were implemented to address challenges in managing shared history, facilitating agent communication, and maintaining team-level coordination:

- **Centralized History Storage:** The `OrgAgent` maintains a comprehensive repository of all task-related reports, observations, and results, ensuring shared context and consistency across all agents. It will provide sub-agents with the relevant information out of its repository for challenging their specific task.
- **History Summarization:** At the end of each iteration, the sub-agent summarizes task progress and updates the shared history, enabling agents to operate with concise and relevant information.
- **Definition of Done Criteria:** Each subtask includes a clear definition of done, empowering sub-agents to work independently while ensuring task objectives are met.
- **Subtask Decomposition and Delegation:** The `OrgAgent` dynamically decomposes the main task into subtasks and allocates them to sub-agents based on their specialized roles, previous reports, and current progress.

3.3 Implementation Challenges and Solutions

The implementation posed several challenges, mostly concerning facilitating efficient agent work, as well as overcoming common LLM struggles. Additionally, working with the extensive and resource-intensive SWE-agent framework presented its own set of difficulties. One significant challenge was formatting commands so that agents could properly interpret and work with them. For example, the `edit` command, tailored by the SWE-agent framework as ACI, still posed challenges, as agents would sometimes get stuck in endless loops, repeatedly trying the same operation. This also complicated debugging cycles, as agents would fail to recognize that the code base could change according to their edits between iterations, assuming the original reference instead. As a result, agents struggled to properly use the ACI, necessitating several minor improvements and adjustments throughout the project. Furthermore, the SWE-agent framework itself proved to be extensive and difficult to navigate, as it makes extensive use of hooks and contains files that span thousands of lines. This complexity, combined with the undeterministic nature of LLM output, further hindered debugging and made it challenging to ensure consistent behavior.

3.4 Changes to the Agent Class for History Handling

Several changes were made to the `Agent` class to improve history management and facilitate smoother interaction within the organizational agent framework. These modifications focus on them working appropriatly in the organizational agent framework, improving task summaries, and ensuring clear delegation and reporting between agents.

The `Agent` class was modified to support summarization at the end of each iteration. After completing a subtask, the `Agent` generates a summary that includes important details such as the

outcome of the subtask, the current progress on the overall task, and any insights or challenges encountered during the iteration. These summaries are reported to the `OrgAgent`, making them available for future interactions by other agents. This ensures that each agent working on the task has access to a clear record of prior actions — as summarized by the organizer — and can build on them effectively.

Another key update to the `Agent` class involves the integration of observations from executed subtasks. These observations are formatted and added to the shared history, which helps maintain the context for subsequent sub-agent interactions. This change ensures that agents are always working with the most up-to-date information, reducing the risk of repeating previous steps or missing new context that could affect task completion.

Finally, to differentiate the role of `Agent` from that of the `OrgAgent`, the `Agent` class was adjusted to ensure that only the `OrgAgent` can issue the "submit" command. Sub-agents, represented by the `Agent` class, now only signal task completion by using the "DONE" keyword, which is a critical distinction within the organizational framework. This change prevents sub-agents from prematurely finalizing tasks, ensuring that the `OrgAgent` retains control over when tasks are considered complete.

3.5 Sub-Agent Implementation and Dynamic Task Assignment

The `OrgAgent` interacts with a variety of sub-agents, each of which is prompted to perform specific tasks as part of a larger workflow. Rather than having role-specific implementations or functions, the differentiation between sub-agents lies in the task definitions and prompts provided by the `OrgAgent`. The `OrgAgent` dynamically generates and assigns tasks to these sub-agents based on the specific needs of the task at hand, as well as the context provided by prior interactions. While all of them have the same capability, this is thought as a guidance for task delegation, which could be further extended by fine-tuned models or command access in the future.

For instance, the `OrgAgent` may assign a "coding" task to a sub-agent by providing a task-specific prompt along with any necessary file references. This task is executed by the sub-agent, which performs the task autonomously within the scope of the given prompt. Similarly, the `OrgAgent` might prompt a sub-agent with a task related to testing, code review, or planning, and the sub-agent will carry out the necessary work according to the provided instructions.

4 Discussion

Benefits of Organizational Framework. Our evaluation demonstrates both the potential and limitations of the organizational agent approach. The key advantage observed is the introduction of a meta-level coordinator that provides strategic oversight. This organizational structure proved particularly valuable in three areas: (1) deadlock resolution, where the OrgAgent could redirect stuck sub-agents to more promising approaches, (2) context maintenance, ensuring consistent knowledge sharing across the team, and (3) workload optimization through dynamic task allocation.

Performance Analysis. The significant gap between completion rates (34.78%) and success

Table 1: Comparison of Hierarchical vs. Non-Hierarchical Multi-Agent Configurations

Configuration	Total	Submitted	Completed	Resolved	Completion Rate	Success Rate
Hierarchical	23	8	8	2	34.78%	25%
Non-Hierarchical	23	18	2	0	64.28%	11%

rates (25%) in the Agile framework, compared to the single agent's metrics (64.28% and 11% respectively), reveals interesting insights about agent self-assessment. The Agile framework's closer alignment between perceived and actual success suggests that collaborative validation helps agents develop more accurate assessments of their work quality.

Scalability and Resource Considerations. While the organizational framework shows promise, it introduces significant computational and complexity overhead. Each additional agent increases not only the direct computational load but also the coordination complexity exponentially. Our experiments suggest that the optimal team size varies based on task complexity - smaller teams (2-3 agents) performed better on focused, single-domain tasks, while larger teams showed advantages on complex, multi-domain problems requiring diverse expertise.

Limitations and Future Work. Several key limitations emerged during our evaluation. First, the communication overhead between agents sometimes led to decision latency. Second, while the OrgAgent effectively manages task distribution, it occasionally struggles with resolving conflicting suggestions from sub-agents. Future work should focus on:

- Developing more sophisticated conflict resolution mechanisms for the OrgAgent
- Optimizing the communication protocols to reduce coordination overhead
- Investigating dynamic team sizing based on task characteristics
- Implementing learning mechanisms to improve agent role allocation over time

Implications for Software Engineering. The success of the Agile framework in achieving higher quality outcomes, despite lower raw completion rates, suggests that structured collaboration among LLM agents might be particularly valuable for complex software engineering tasks where accuracy and reliability are paramount. The framework's improved self-assessment capabilities also indicate potential for more reliable autonomous development processes, though careful consideration must be given to the trade-off between quality and development speed.

5 Conclusion

This paper investigated the efficacy of different software engineering paradigms for LLM-based agents, comparing an Agile multi-agent approach against a single-agent framework. Our implementation, built upon SWE-agent and inspired by ChatDev, demonstrates the potential of structured collaboration between specialized agents in software development tasks.

Evaluated using SWE-bench, our results showed that while the Agile framework achieved a lower completion rate (34.78% vs 64.28%), it demonstrated substantially higher success rates (25% vs 11%) compared to the single-agent approach. This suggests that collaborative validation and specialized roles contribute to more reliable software development outcomes, despite increased coordination overhead.

The implemented multi-agent system introduces key innovations including a centralized coordinator for task management, specialized agent roles, and a structured history management system. These components enable sophisticated problem-solving capabilities while maintaining consistent context across the development process. While challenges remain in optimizing agent communication and resource management, this work demonstrates that principles from human team organization can effectively enhance the performance of LLM-based agents in software engineering tasks.

References

1. Liu J, Wang K, Chen Y, et al. Large Language Model-Based Agents for Software Engineering: A Survey. 2024. arXiv: 2401.00755 [cs.SE].
2. Qian C, Liu W, Liu H, et al. ChatDev: Communicative Agents for Software Development. 2023. URL: https://arxiv.org/abs/2307.07924v5 (visited on 10/01/2024).
3. Jimenez CE, Yang J, Wettig A, et al. SWE-BENCH: CAN LANGUAGE MODELS RESOLVE REAL-WORLD GITHUB ISSUES? 2024.
4. Yang J, Jimenez CE, Wettig A, et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793. 2024. DOI: 10.48550/arXiv.2405.15793. URL: http://arxiv.org/abs/2405.15793 (visited on 10/10/2024).
5. Nguyen MH, Chau TP, Nguyen PX, and Bui NDQ. AgileCoder: Dynamic Collaborative Agents for Software Development based on Agile Methodology. arXiv:2406.11912 [cs]. 2024. URL: http://arxiv.org/abs/2406.11912 (visited on 10/16/2024).
6. Lin F, Kim DJ, Tse-Husn, and Chen. When LLM-based Code Generation Meets the Software Development Process. arXiv:2403.15852 [cs]. 2024. URL: http://arxiv.org/abs/2403.15852 (visited on 10/16/2024).