Software Development Using a Multi-agent Approach

Juliana Gomes Ribeiro[1], **Joel Machado Pires**[1], **Jonas Oliveira Pereira**[1], **Frederico Araujo Durão**[1], **Célia Ghedini Ralha**[1]

[1]Computer Institute – University of Bahia (UFBA)
Campus Ondina, 40.170-110 Salvador, Brazil

{julianagr,joelpires,jonas.pereira,fdurao,celiaralha}@ufba.br

Abstract. *Software development involves complexities in requirements analysis, code generation, and continuous validation. This work presents a multi-agent system (MAS) designed to enhance the software engineering process by integrating Large Language Model (LLM) capabilities with human oversight. The MAS comprises five specialized agents that communicate through a shared message broker and maintain context in a persistent knowledge base. These agents autonomously handle tasks such as architectural planning, artifact generation, code review, and operational management, while human experts intervene to ensure accuracy and strategic alignment. The findings indicate that LLMs can impact software engineering, offering tools and techniques that automate tasks, enhance code generation, and potentially improve the development process.*

Keywords: Gemini AI Chatbot, LLM, multi-agent system, software engineer.

1. Introduction

Software development is inherently complex, involving multiple stages, from requirement gathering to coding, testing, and deployment, each with specific challenges. Challenges such as collaborative development, schedule estimation, and project management persist despite technological advances. Exists variation in development standards, discrepancies in technical skills, and high-quality software to meet real-world demands.

Focusing on human communication gaps, inconsistent development standards, and varying technical skills often leads to inefficiencies and errors. Problems like detecting term ambiguities in requirement documents, improving clarity and consistency in interdisciplinary projects, demand the ability to collaborate with humans and external tools to address complex tasks.

Recent research has explored Large Language Model (LLM) with Multi-Agent System (MAS) to automate and streamline these tasks [Cinkusz et al. 2025, Jin et al. 2024, Liu et al. 2024]. Multiple intelligent agents can work along with humans to achieve software project goals, automating the generation of artifacts such as requirement documents, offering intelligent code suggestions, and assisting with tasks like code refactoring and test generation.

This work presents an MAS to guide the entire software development process, from requirements gathering to coding. The system provides recommendations to human experts, who can accept or reject, steering the process in the correct direction. By combining the strengths of LLM with human expertise, this approach aims to enhance the efficiency, accuracy, and reliability of software development processes. By releasing the implementation openly,[1] we support reproducibility and promote further research.

The rest of this manuscript includes, in Section 2, related work, in Section 3, materials and methods, presenting five agent roles, the architecture, and the Tropos model to elucidate the agent-oriented software system. Section 4 presents the results of the MAS, while Section 5 includes a conclusion and future work.

2. Related Work

In the literature, some approaches faced issues with low-information-density outputs and vague requirements. [Jin et al. 2024] leverage transformer-based models, detecting term ambiguities in requirements engineering, while [Liu et al. 2024] demonstrate LLM agents' versatility in various software tasks. [Lin et al. 2024] explore LLM-driven process modeling for code generation and testing, yet struggled with correctness and modularization. [Qian et al. 2024a] integrate LLM into multi-agent frameworks with specialized roles using ChatDev.

In testing and validation, [Jorgensen et al. 2024] showed that prompt precision significantly impacts LLM-generated test cases, leading to inconsistent results. In project management, [Sami et al. 2024] applied ChatGPT to requirement prioritization via the Analytic Hierarchy Process (AHP) and Weighted Shortest Job First (WSJF) but omitted scheduling considerations. [Hong et al. 2024] proposes meta-programming for a multi-agent collaborative framework called MetaGPT, although innovative in collaborative development, still resource-intensive. [Cinkusz et al. 2025] found scalability limitations in cognitive agents powered by LLM-driven agile software development.

Platforms such as AutoGen [Wu et al. 2023] and SWE-agent [Yang et al. 2024] enable LLM agents to interact directly with file systems, compilers, and test environments, granting them IDE-like capabilities and leading to significant improvements in code quality and test success rates. Other tools, such as MacNet [Qian et al. 2024b], model thousands of agents as instruction-level components in a distributed computing architecture, achieving state-of-the-art performance on complex system design benchmarks by leveraging collaborative scaling effects.

Several works also demonstrate the effectiveness of agent teams in debugging and quality assurance. In user-acceptance testing (UAT), XUAT-Copilot [Wang et al. 2024] orchestrates agent roles such as planner, executor, and verifier, automating tests at scale, achieving a pass rate nearly four times higher than single-agent baselines. MapCoder [Islam et al. 2024] exemplifies the power of modular agent roles by separating retrieval, planning, coding, and debugging stages, enabling a structured and efficient development pipeline. AgileCoder [Nguyen et al. 2024] adopts a multi-agent approach based on Scrum roles, showing that agent-driven implementations of Agile practices can further enhance testing and code generation outcomes.

Together, these studies illustrate that multi-agent coordination not only scales code generation and debugging but also emulates real-world development processes with increasing performance. Despite the advancements, existing solutions still fall short of addressing the full spectrum of software development challenges. LLM suffer from critical

[1] https://github.com/joel021/agents

limitations such as hallucinations, inconsistent contextual understanding, and difficulty autonomously managing the software development lifecycle [Rao et al. 2025]. These results suggest that LLM-powered systems should not replace human experts but rather work collaboratively with them.

3. Materials and Methods

This section presents the MAS with five essential roles, which are fundamental to any software development team: Project Manager, Software Engineer, Developer Specialist, Operation System Controller, and Research. These agents are incorporated in the architecture, collaborating to cover all aspects of the software development lifecycle:

- Project Manager - coordinates the entire development lifecycle, translating high-level user requirements into actionable tasks, interfaces with expert users to guide architecture and project management decisions, tracks progress, delegates responsibilities to other agents, and maintains project state beliefs. It interacts with the Developer Specialist and Software Engineer to assign and monitor tasks.
- Software Engineer - ensures software quality, compliance with development standards, validates code, initiates refactoring, and may generate corrective tasks based on beliefs about code quality. It reviews artifacts from the Developer Specialist and flags issues or approves deliverables with the Project Manager.
- Developer Specialist - implements software components based on specifications, generates and modifies code, responds to feedback, and manages commits. It collaborates with the Software Engineer for validation and integrates research findings provided by the Research Agent. For example, when building an authentication system, it applies best practices such as JSON Web Token (JWT) and hashing.
- Operation System Controller - executes environment-level tasks and manages resource allocation, validates and performs system commands requested by other agents, ensuring safe and stable execution. For example, it might prepare the development environment before the Developer Specialist begins implementation.
- Research - provides external technical knowledge by conducting targeted internet searches, responds to queries from other agents (e.g., for bug fixes or design options), and delivers validated findings, giving support to the Developer Specialist in aligning with current standards.

The system operates in a partially observable environment where agents have limited access to Internet resources and system states. The environment is non-deterministic, requiring robust error handling and replanning capabilities when agents generate incorrect artifacts or encounter execution failures. The system maintains stability through discrete state management and sequential processing, ensuring agents only deliberate when the environment is stable.

3.1. System's architecture

The MAS architecture enables agents to autonomously select recipients based on evolving task goals, rather than relying on predefined communication flows, such as when the Project Manager interacts with both the Software Engineer and the Developer Specialist to coordinate a new implementation.

Figure 1 presents the system's implementation harnesses Python[2] for seamless integration with machine learning components. Real-time communication among agents is achieved through Redis,[3] which uses in-memory storage and Pub/Sub functionality to

[2]https://aws.amazon.com/pt/what-is/python/
[3]https://www.ibm.com/br-pt/topics/redis

enable a shared channel for asynchronous message exchange. Redis allows communication between the user and agents using the same channel, functioning as a bridge for all participants.

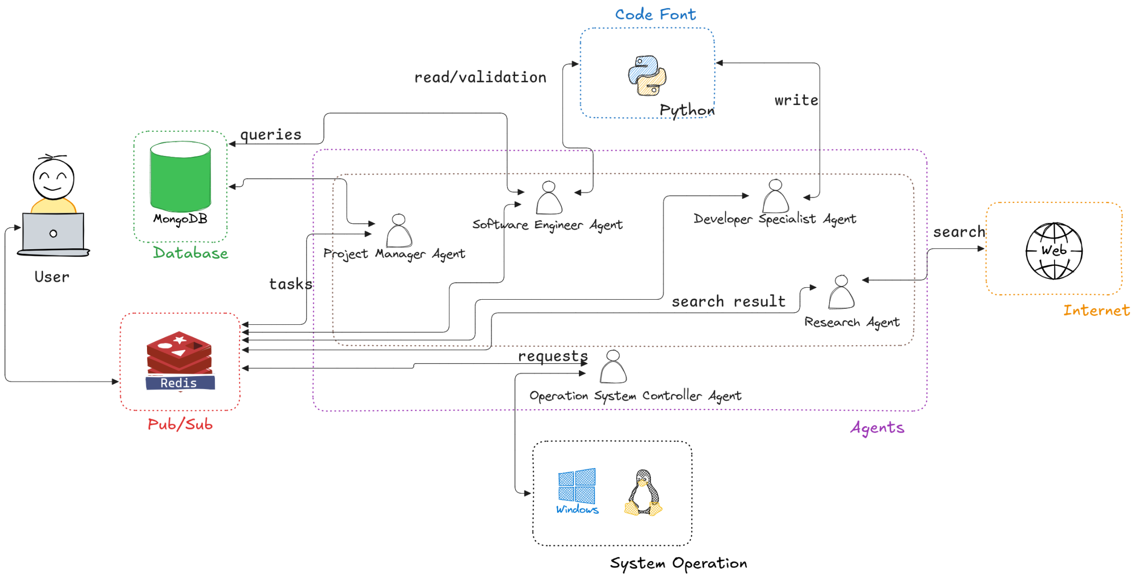

Figure 1. Software development architecture using an MAS approach.

For persistent storage and long-term context tracking, MongoDB[4] maintains all relevant project artifacts, including user commands, messages, epics, and tasks. Thus, agents can reason with historical data — for instance, retrieving the status of a task, reviewing previous decisions, or identifying unresolved requirements. The Project Manager may use this information to assess current progress and decide whether to create, initiate, or complete tasks or stories. The Software Engineer may consult the database to determine new dependencies or to refactor, and then guide the Developer Specialist and Operating System agents accordingly. Such structured memory plays a key role in reducing hallucination risks by anchoring LLM outputs to concrete, verifiable project data.

In addition, the architecture includes actuator interfaces for file system access and internet queries, used respectively for codebase operations and technical research. These interfaces (namely, Code Font and Internet) allow agents to interact directly with the operating system, executing command-line operations such as creating, updating, or deleting files, and to search external sources for technical documentation. By combining this modular architecture with persistent state management and multi-agent oversight, the system enforces checks and balances that help mitigate common LLM errors and hallucinations.

3.2. Agents' Reasoning Model

The system employs LLM as the cognitive engine for each agent, enabling sophisticated pattern recognition and reasoning capabilities while maintaining controlled interaction patterns to mitigate hallucination risks. Building upon these capabilities, a predicate-based reasoning model allows agents to represent and reason about the current state of the system, context-specific conditions, and actions using well-defined predicates:

[4]https://www.ibm.com/br-pt/topics/mongodb

- Objects — fundamental entities or events that agents interact with, such as messages, tasks, and project specifications.
- Beliefs (facts) - reflect the current state and context captured by predicates. For example, when a message is received or a task is identified, these beliefs help guide agent decision-making. They express conditions such as `message_received(m)` or `task_id_present(m)`
- Propositional rules - logical rules specifying how agents should act based on their beliefs, conditional statements that govern the agent's actions, such as executing tasks if certain conditions occur. For example, a rule might state $\texttt{message\_received}(m) \wedge \texttt{sender\_is\_pm}(m) \rightarrow \texttt{retrieve\_task}(t)$. This rule means that when a message is received and the sender is verified as the Project Manager, the system should retrieve the corresponding task.

By defining inference rules, the decision-making process is mapped into formal, testable logical expressions. The predicates are implemented across the five agents to ensure reliable and traceable decision-making:

- Project Manager - uses this model to process user requirements and coordinate development activities with beliefs about the project state and rules for task delegation.
- Software Engineer - applies the model to code validation, using beliefs about code quality and propositions that determine when to request revisions or approve deployments.
- Developer Specialist - focuses on code generation, with beliefs about requirements and rules governing when to incorporate feedback or submit completed artifacts.
- Operation System Controller - uses beliefs about the system state and propositions to ensure the safe execution of requested actions.
- Research - employs beliefs about information needs and propositions for validating and delivering relevant findings to other agents.

3.3. Tropos Methodology

The Tropos agent-oriented software development methodology was used to model the MAS using the piStar online tool (version 2.1.0) [Pimentel and Castro 2018]. Figure 2 presents the MAS architectural design. The model represents the agents collaborating to achieve system goals, while maintaining clear boundaries of responsibility and explicit dependency relationships. In the model, there are actors (red circles), goals (orange rounded rectangles), tasks (beige hexagons), and resources (pink rectangles). Dependencies between elements are represented by 'D' links, while task decomposition and means-end relationships are shown through connecting lines. Dashed boundaries group related elements that belong to a single actor's scope of responsibility [Bresciani et al. 2004].

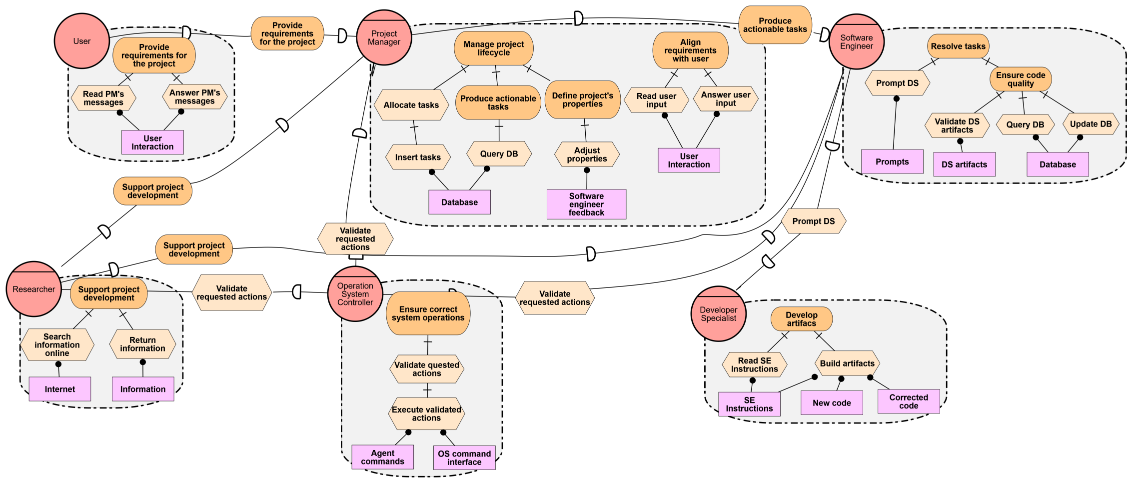

Figure 2. The Tropos model representing the interaction of agents in the MAS.

The human-in-the-loop contribution is represented through bidirectional interaction patterns between the User and Project Manager, with dedicated user interaction nodes enabling continuous feedback incorporation. The feedback loop extends through the Software Engineer's oversight of Developer Specialist outputs, creating multiple validation points.

End-to-end process automation is achieved through a chain of dependencies from initial user requirements to final artifact delivery. The Project Manager's central role ensures systematic task allocation and progress tracking. The Operation System Controller validates and executes all automated actions, creating a controlled development pipeline with clear handoff points.

Hallucination mitigation is embedded in the model's multi-layered validation structure, where each agent contributes to reliability: the Research validates information sources, the Operation System Controller ensures command validity, and the Software Engineer verifies the Developer Specialist's outputs. The database integration provides a persistent truth reference, while user interaction nodes enable expert validation.

Code quality and modularization are enforced through dedicated quality control mechanisms in the Software Engineer's domain. Code artifacts undergo validation before database updates, with explicit dependencies ensuring quality standards are met before task completion. This separation between task resolution and quality assurance maintains high development standards.

4. Results

The MAS using the Gemini AI Chatbot (gemini-2.0-flash) is illustrated in Listing 1.

Listing 1. Agents interaction using LLM.

```
Developer Specialist Agent, Software Engineer Agent, Research Agent, Project Manager Agent Started.
Service started
Send a message: Desciption: Backend service built using Python and Flask that provides the following functionalities:
User Authrentication: User registration (name, email, password), User login with JWT-based authentication, Token
refresh and logout. User Management (CRUD): Create, Read, Update, and Delte users, Fetch a single user by ID, Fetch a
list of users with pagination. Security Considerations: Password hashing (bcript or similar), Role-based access con-
trol (admin/user roles), Secure endpoints with authentication middleware. Tech Stack & Dependencies: Flash (or Flask-
RESTful), Flask-JWT-Extended for authentication, SQLAlchemy (PostgreSQL preferred), Marshmallow for request valida-
tion, Docker support (optional).API documentation.The projet location is at: /home/joel/Documents/agents/work_dir/crud
Location is at:/home/joel/Documents/agents/work_dir/crud
2025-02-12 20:42:52 Projeto Manager: @Software Engineer
Software Engineer, please start working on the User Authentication story. Begin by setting up the develop environment,
including installing necessary packages.
2025-02-12 20:42:52 Software Engineer: @Operation System Controller
Operation System Controller, please set up a new project directory at Documents/agents/work_dir/crud and crate a vir-
tual environment within it. Install the following packages: Flask, Flask-JWT-Extended, SQLAlchemy, Marshmallow, and
bcrypt. Let me know when it’s ready.
2025-02-12 20:42:52 Project Manager: @Developer Specialist
Developer Specialist, please start the development of the User Authentication API endpoints based on the tasks created
by the Project Manager.Ensure to implement security measures such as password hashing and JWT authentication.
2025-02-12 20:42:52 Developer Specialist: @Research
Research, please gather information on best practices for implementing JWT-based authentication in Flask, including
token refresh and secure storage of secret keys. Provide a summary.
2025-02-12 20:42:52 Software Engineer: @Project Manager
I could not attend to the message: Software Engineer, please start working on the User Authentication story. Begin by
setting up the development environment, including installing necessary packages..
```

In this work, the implemented MAS validates four key contributions:

- Human-in-the-loop - the system accepts high-level user requirements through a command-line interface, which triggers the initialization of specialized agents. The Project Manager translates high-level inputs into actionable tasks, ensuring continuous expert oversight.
- End-to-end process automation with validation mechanisms - the system automates workflow execution through a chain of coordinated actions. When tasked with implementing user authentication, the Project Manager automatically delegates environment setup to the Operation System agent, development tasks to the Developer Specialist, and research queries to the Research agent. This structure enables traceable execution paths and verifiable task completion.

- Hallucination mitigation and error handling - safeguards address LLM hallucinations and errors. When issues like missing directories or invalid dependencies occur, agents promptly report failures, enabling the Software Engineer to initiate corrective actions. Transparent error reporting and cross-agent validation form a robust framework for detecting and correcting inconsistencies.
- Modularization and code quality enforcement - the results illustrate systematic enforcement of development standards through structured agent interactions. For example, in implementing authentication features, the Developer Specialist incorporates security best practices such as JWT authentication and password hashing, while the Research agent ensures alignment with current industry standards. The system maintains a modular organization with components for agents, utilities, and configurations, reflecting commitment to maintainable high-quality code.

5. Conclusion

The MAS demonstrates semi-autonomous software development capabilities through continuous interaction with software engineering experts. It employs an agile methodology to handle epics, user stories, and tasks dynamically while autonomously managing subtask decomposition and prioritization. A notable innovation is the system's ability to propose contextually appropriate architectural solutions, advancing the state of semi-autonomous software development.

Several directions for future work have been identified. These include expanding agent actuator capabilities, implementing persistent memory across system restarts, supporting multiple concurrent projects, developing a web-based user interface, optimizing the single-threaded message handling architecture, and refining LLM interaction prompts. Addressing these limitations would enhance the system's robustness and practical applicability in complex software development environments.

References

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.

Cinkusz, K., Chudziak, J. A., and Niewiadomska-Szynkiewicz, E. (2025). Cognitive agents powered by large language models for agile software project management. *Electronics*, 14(1).

Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., and Schmidhuber, J. (2024). MetaGPT: Meta programming for a multi-agent collaborative framework. arXiv:2308.00352, `https://arxiv.org/abs/2308.00352`.

Islam, M. A., Ali, M. E., and Parvez, M. R. (2024). MapCoder: Multi-agent code generation for competitive problem solving. arXiv:2405.11403, `https://doi.org/10.48550/arXiv.2405.11403`.

Jin, H., Huang, L., Cai, H., Yan, J., Li, B., and Chen, H. (2024). From LLMs to LLM-based agents for software engineering: A survey of current, challenges and future. arXiv:2408.02479, `https://arxiv.org/abs/2408.02479`.

Jorgensen, S., Nadizar, G., Pietropolli, G., Manzoni, L., Medvet, E., O'Reilly, U.-M., and Hemberg, E. (2024). Large language model-based test case generation for gp agents. In *Genetic and Evolutionary Computation Conference (GECCO'24)*, page 914–923, New York, NY, USA. ACM.

Lin, F., Kim, D. J., et al. (2024). When LLM-based code generation meets the software development process. arXiv:2403.15852, `https://arxiv.org/html/2403.15852v1`.

Liu, J., Wang, K., Chen, Y., Peng, X., Chen, Z., Zhang, L., and Lou, Y. (2024). Large language model-based agents for software engineering: A survey. arXiv:2409.02977, `https://arxiv.org/abs/2409.02977`.

Nguyen, M. H., Chau, T. P., Nguyen, P. X., and Bui, N. D. Q. (2024). AgileCoder: Dynamic collaborative agents for software development based on agile methodology. arXiv:2406.11912, `https://doi.org/10.48550/arXiv.2406.11912`.

Pimentel, J. and Castro, J. (2018). piStar Tool – a pluggable online tool for goal modeling. In *IEEE 26th Int. Requirements Engineering Conference (RE)*, pages 498–499.

Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., and Sun, M. (2024a). ChatDev: Communicative agents for software development. In *the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, Bangkok, Thailand.

Qian, C., Xie, Z., Wang, Y., Liu, W., Zhu, K., Xia, H., Dang, Y., Du, Z., Chen, W., Yang, C., Liu, Z., and Sun, M. (2024b). Scaling large language model-based multi-agent collaboration. arXiv:2406.07155, `https://doi.org/10.48550/arXiv.2406.07155`. Accepted to ICLR 2025.

Rao, H., Zhao, Y., Hou, X., Wang, S., and Wang, H. (2025). Software engineering for large language models: Research status, challenges and the road ahead. arXiv:2506.23762v1, `https://arxiv.org/html/2506.23762v1`.

Sami, M. A., Waseem, M., Rasheed, Z., Saari, M., Systä, K., and Abrahamsson, P. (2024). Experimenting with multi-agent software development: Towards a unified platform. arXiv:2406.05381, `https://doi.org/10.48550/arXiv.2406.05381`.

Wang, Z., Wang, W., Li, Z., Wang, L., Yi, C., Xu, X., Cao, L., Su, H., Chen, S., and Zhou, J. (2024). Xuat-copilot: Multi-agent collaborative system for automated user acceptance testing with large language model. arXiv:2401.02705, `https://doi.org/10.48550/arXiv.2401.02705`.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. (2023). Autogen: Enabling next-gen llm applications via multi-agent conversation. arXiv:2308.08155, `https://doi.org/10.48550/arXiv.2308.08155`.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. (2024). SWE-agent: Agent-computer interfaces enable automated software engineering. arXiv:2405.15793, `https://doi.org/10.48550/arXiv.2405.15793`.