Proposal for a Role-Based Multi-Agent LLM Architecture for Corporate Software Delivery

Alisson Luan de Lima Peloso
Federal University of Fronteira Sul
Chapecó, Brazil
Optidata
Chapecó, Brazil
alisson.peloso@estudante.uffs.edu.br

Fernando Bevilacqua
Optidata
Chapecó, Brazil
fernando@optidata.com

Abstract

Large language models (LLMs) have advanced from one-shot code generation to agents that iteratively interact with development environments. Their evaluation, however, remains detached from how software is delivered in companies: dominant benchmarks measure isolated, self-contained tasks, and most multi-agent frameworks operate in greenfield scenarios, disconnected from the repositories, review processes, staging environments and people that constitute a real delivery flow. This short paper proposes a role-based multi-agent architecture, with Product, Architect, Engineer, Reviewer and QA agents, that covers the flow end-to-end from requirement refinement to staging validation. It uses the version-control platform itself as the communication and traceability medium, advances through staged validation gates, and keeps humans reachable at every step through a team manifest. Our main contribution is an agentic software development life cycle grounded in a survey of LLM-based agent approaches relevant to corporate delivery (multi-agent frameworks, benchmarks, requirements refinement and quality assurance), together with an evaluation protocol based on real issues from production repositories and centered on acceptance-criteria coverage.

Keywords

LLM Agents, Multi-Agent Systems, Software Engineering, Human-in-the-Loop, Acceptance Criteria, Software Delivery Automation

1 Introduction

The use of Large Language Models (LLMs) in the Software Development Life Cycle (SDLC) is rapidly transforming with Agentic SDLC (ASDLC) featuring human-AI cooperation [17]. An LLM acts as an agent when it interacts with an environment, taking actions iteratively and receiving feedback through an interface to the environment (e.g. search, file navigation) to resolve real issues [26]. In that context, the use of multiple agents over a single one presents improved results. Approaches range from agents organized in a waterfall-style pipeline communicating in natural language to work division among specialized roles [14]. Cooperation between human and AI remains an important aspect throughout this process, with humans kept in the loop to supervise, steer and validate agent work, as surveys of LLM-based agents for software engineering consistently report [15].

Despite this progress, there is a persistent mismatch between what these systems are evaluated on and how SDLC operates in a corporate environment. Evaluation is typically dominated by isolated, self-contained tasks, and even benchmarks built from real repositories frame the solution as an isolated patch, a gap between benchmark performance and real-world software engineering workflows documented in recent surveys of LLM code generation [7]. Unlike a lab-like, controlled environment, corporate software delivery presents a complex cycle that contains a product and a strategy [21]. That is precisely the part these benchmark-centric evaluations lack. In that light, a feature is not a patch: it is part of a workflow that involves written requirements, refinements and user stories, acceptance criteria, planning and implementation against an existing codebase, followed by review and tests before a release. Humans are accountable at each step [1] due to the financial aspects of the outcome, as enterprise studies of AI coding assistants underscore the productivity, quality and security stakes at play [16].

This short paper presents a research proposal for a role-based multi-agent architecture that covers the delivery flow end-to-end, from requirement refinement to validation in the context of a corporation with a real product. Human-AI collaboration is conducted through a version-control platform, keeping humans accountable and in the loop. Our main contribution is the proposal of an ASDLC, to be evaluated in a corporate SDLC context, grounded in a survey of LLM-based agent approaches relevant to this goal. We present this literature evidence on LLM-based agents in the SDLC, highlighting the use of role-based, multi-agent approaches in a delivery contextualized not in a laboratory setup, but in the challenging environment of a corporation whose product has an audience, requirements, planning, implementation, and testing. Finally, we propose an evaluation of our approach.

2 LLM-Based Agents for Software Development

Using LLM agents to automate software maintenance and improvement is itself an active research direction [24]. The works presented in this section summarize different ASDLC approaches, how they are evaluated and how they diverge from, or fall short of, a corporate setup.

2.1 Multi-Agent Frameworks

Qian et al. [19] propose *ChatDev*, which simulates a software company in which agents communicate exclusively in natural language, following a waterfall model where one phase of the process happens at a time. The evaluation reveals recurring failures such as unimplemented methods and missing imports, which are resolved through repeated development, review and test interactions, defects that static analysis could catch before review. Hong et al. [9] propose *MetaGPT*, which takes the opposite stance on communication: instead of free-form dialogue, it encodes Standard Operating

Procedures, requiring each role to produce structured outputs (e.g., a Product Requirements Document) consumed by the next role through a shared message pool. It was evaluated on *HumanEval*, the *Mostly Basic Python Problems* (MBPP) and *SoftwareDev* datasets using executability, cost and the number of human interventions required.

Yang et al. [29] propose *SWE-agent*, a single-agent framework that contributes the key idea of the agent-computer interface: purpose-built commands for searching, reading and editing files designed to use the context window efficiently. It reports %Resolved (pass@1) and average dollar cost per instance, with a $4 budget limit per task, an early acknowledgment that cost is a first-class metric for agents.

Underneath such systems sit general-purpose orchestration frameworks, e.g. *AutoGen* [27], *CrewAI* [6] and *LangGraph* [13], which supply conversation patterns, role abstractions and graph-structured control flow. They are deliberately domain-agnostic: they define how agents talk and hand off, not which roles a delivery flow needs, which artifacts it produces or where a human must sign off. Our proposal is scoped to that missing layer and could be implemented on any of them.

2.2 Benchmarks and Their Limits

Chen et al. [4] established the dominant evaluation paradigm with *HumanEval*: 164 hand-written programming problems checked by unit tests, with correctness measured by pass@k, but they remain isolated functions. Jimenez et al. [10] propose *SWE-bench*, moving to 2,294 tasks extracted from real pull requests in 12 Python repositories: given a repository and an issue, the model must produce a patch that makes the original test suite pass. Performance is very low (GPT-4 resolved ∼1.7% of tasks with retrieval), showing that understanding large real codebases, localizing changes across files and reasoning about side effects are fundamentally harder than completing functions.

Xu et al. [28] propose *TheAgentCompany*, which evaluates agents on 175 professional tasks in a self-hosted simulated company with GitLab, an issue tracker, chat and simulated colleagues. Beyond binary success, it scores partial completion through weighted checkpoints, which localize *where* in a long-horizon flow the agent fails. The best agent fully completed 24.0% of tasks (34.4% partial score), with communication as the main bottleneck and a documented tendency to fabricate shortcuts when stuck. Han et al. [8] propose *SWE-Skills-Bench*, a requirement-oriented design in which each task carries explicit acceptance criteria, and each criterion is converted into a deterministic test, so progress is traceable to requirements without relying on an LLM judge. Its paired with/without-skill comparison found a mean gain of only +1.2% and no correlation between added context (token cost) and accuracy.

2.3 Requirements and Acceptance Criteria

Zhang et al. [30] apply two agents, a Product Owner and a Requirements Engineer, to improve user stories in a real industrial setting (Austrian Post, six agile teams). The agents improved clarity, completeness and business value on INVEST-based criteria, but several stories became too long for practical use, and human validation, particularly by the PO, remains necessary to avoid scope creep. Quattrocchi et al. [20] compared 10 LLMs against students and an expert ground truth on user-story generation: LLMs achieved better coverage of the central requirements but markedly lower diversity, and when given an explicit evaluation codebook they assessed story quality with high human agreement ($\kappa = 0.87$): LLMs are good first-draft generators, and explicit criteria make LLM-based assessment reliable.

For acceptance criteria specifically, Wang et al. [25] propose *RAGcceptance M2RE*, which uses multimodal retrieval-augmented generation, drawing textual and visual context from the actual project, to generate Gherkin-format criteria, raising accuracy by over 20 percentage points versus a no-RAG baseline in an industrial system. Rawson and Reddivari [22] address the same grounding problem without RAG: they inject multi-level codebase summaries and the relevant requirements into the prompt, observing that without such context the model invents criteria or targets the wrong use case. The consistent finding is that acceptance criteria are only as good as the project context provided, and that Gherkin gives them a direct bridge to testing.

2.4 Quality Assurance by Agents

Chevrot et al. [5] investigate converting Autonomous Web Agents into Autonomous Test Agents (*ATA*), an important distinction: an end-to-end test must follow prescribed steps and evaluate assertions, while a generic web agent only needs to reach a goal. Their *PinATA* design splits the tester into orchestrator, actor and assertor sub-agents. Almeyda Alania et al. [2] orchestrate two LLMs to transform backlog user stories into executable tests, injecting the frozen database schema to anchor generation in real data. They report an 80% reduction in test-design effort and a 91.9% pass rate, but concede that this "functional correctness" measures whether scripts execute, not whether they detect faults. At a different level, Kohl et al. [11] adapt the test pyramid to agents through structural testing, using OpenTelemetry traces, mocked LLM responses and assertions over observed behavior, covering the "are we building the product right?" question that acceptance-level evaluations skip.

3 Human-in-the-Loop and Corporate Adoption

The previously mentioned works automate individual functions. A smaller set examines what happens when agents are deployed inside a company. Takerngsaksiri et al. [23] propose *HULA*, deployed at Atlassian to resolve Jira work items, keeping engineers in control of planning and coding stages and reviews of every output, resolving 31% of *SWE-bench* Verified issues as a guardrail metric. Evaluating such systems is itself a challenge directly relevant to our design: unit-test-based correctness is binary, expensive and hard to scale, while LLM-as-judge fluctuates enough to mask small improvements, leaving a methodological gap between the two. Bandara et al. [3] propose *Agentsway*, a development methodology designed for agent-based teams (planning, prompting, coding, testing and fine-tuning agents with the human as orchestrator), including privacy-preserving fine-tuning with local LLMs after each development cycle. Its evaluation, however, covers only the planning and prompting agents via Likert-scale ratings. On the adaptation question, Pornprasit and Tantithamthavorn [18] study LLM-based code review automation and find that fine-tuning is the strongest

strategy when enough data exists, while few-shot prompting is the best cold-start option.

Taken together, the surveyed works automate isolated roles, evaluate on synthetic or simulated environments, and only partially address human collaboration. We are not alone in targeting the corporate setting: *HULA* [23] keeps humans in control of an end-to-end Jira-to-pull-request flow, *Agentsway* [3] redesigns the development methodology around agents, and Konda [12] proposes *Konda*, a role-based multi-agent architecture spanning requirements, testing and deployment. What distinguishes our proposal is the combination of three choices that none of them make together: it uses the version-control platform itself as the communication and traceability medium rather than a dedicated bus, it covers the full span from requirement refinement to staging validation under one set of roles, and it makes human escalation explicit through a *team manifest*. *Konda*, the closest work, is evaluated on benchmark datasets and synthetic microservices and frames deployment as release and rollback operations, whereas we ground every step in real issues from production repositories and center the flow on code review and pull-request semantics.

4 Proposal for a Role-Based Multi-Agent Delivery Flow

We propose a multi-agent architecture that mirrors a real development team, illustrated in Fig. 1. The pipeline is composed of five specialized agents, i.e. **Product**, **Architect**, **Engineer**, **Reviewer** and **QA**, which operate on the company's existing infrastructure. The five roles are not arbitrary: each covers one function that the surveyed literature automates in isolation, i.e. requirements refinement (Section 2.3), planning and code understanding (Section 2.2), implementation (Section 2.1), review (Section 3) and quality assurance (Section 2.4), and each mirrors a function a corporate delivery team already staffs. The decomposition therefore follows the role division already adopted by agent teams [3, 12, 14] and by industry practice, rather than an invented taxonomy. Throughout the stages, agents conduct the work with human cooperation when needed, guided by the following profiles:

(1) **Product** refines a raw issue into a user story with acceptance criteria in Gherkin format, grounded in project context (Section 2.3 shows that ungrounded criteria are generic or wrong, and that explicit criteria enable reliable downstream evaluation).
(2) **Architect** researches before coding: it explores the repository, related code and existing external solutions to produce an implementation plan, addressing the code-understanding bottleneck that *SWE-bench* exposes (Section 2.2).
(3) **Engineer** implements the plan and validates locally in stages: static analysis and linting first, then execution in a development environment. The static-analysis gate is a direct response to the failure modes *ChatDev* resolves expensively through review rounds (Section 2.1): deterministic checks run before any LLM-based review, saving tokens and iterations.
(4) **Reviewer** examines the pull request against the implementation plan and the acceptance criteria, posting change requests as review comments on the pull request itself. Because deterministic checks already ran in the **Engineer's** stage, the **Reviewer** is freed to focus on semantic and design issues (plan conformance, side effects, project conventions) rather than the mechanical defects that consume review rounds in *ChatDev* (Section 2.1). Following the code-review findings (Section 3), it uses few-shot prompting at cold start. Final merge approval remains with a human, consistent with *HULA*'s human-control principle (Section 3).
(5) **QA** converts each acceptance criterion into a deterministic test, including browser-automated end-to-end tests executed against a staging deployment, following the requirement-to-test traceability of *SWE-Skills-Bench* and the assertion-centric *ATA* design (Sections 2.2, 2.4).

Communication and traceability between agents and humans are based on GitHub, a version-control platform, via issues, pull requests, comments and code or text files. It allows humans and agents to interact in the same context, which seamlessly incorporates the addition or removal of members (human or AI). The handoff artifacts produced by each stage in the pipeline, such as user stories, acceptance criteria, implementation plans, pull requests and review comments, are versioned, structured and human-readable objects. For instance, acceptance criteria are written in Gherkin in an issue body, and implementation plans are markdown files in the repository `docs` folder. That approach ensures native traceability and evolution of the artifacts produced, preserving the discipline of structured handoffs mentioned by previous work instead of free-form content, and lets humans participate at every stage by editing the artifacts, commenting or judging the work.

Finally, the communication among team members in our proposed role-based architecture is routed via a *team manifest* that is always available to all. It contains one entry per member, i.e. human or agent, detailing their name, GitHub handle and specialization. For instance, entries can be tech lead(s), project owner(s) and QA engineer(s) who can be mentioned in specific situations. During development, for instance, an agent can reach a tech lead when an architectural decision is unclear, or the project owner when a requirement is ambiguous. Since artifacts and collaborators are all mentionable, social handoffs become explicit and traceable across every stage, addressing a failure point identified in previous work [28]. Additionally, agent executions are instrumented with traces (Section 2.4) so failures and progress can be localized within the workflow.

The validation of each phase is staged, so each phase only advances when the previous one is satisfied. The flow admits return cycles when a condition is not properly met. For instance, a change request from the **Reviewer** or a defect found by **QA** during testing reopens the **Engineer's** implementation step.

4.1 Evaluation Plan

We deliberately avoid external synthetic benchmarks, since the object of study is the delivery pipeline itself. The architecture will be evaluated on real issues from at least two production repositories of distinct domains, for example, calendar and file-manager modules of a commercial SaaS. Evaluation will be performed under two protocols. The *retrospective* protocol evaluates re-submission

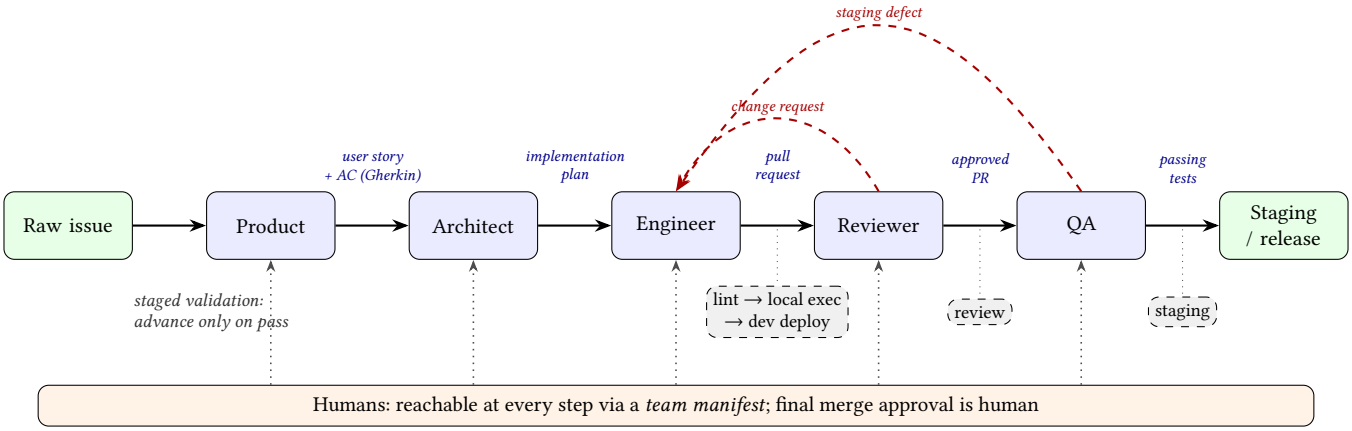

Figure 1: Role-based multi-agent delivery flow. Five agents (blue) hand off versioned artifacts (issues, pull requests and review comments) through the version-control platform itself, giving native traceability. Work advances through staged validation gates (dashed) only when the previous one passes, and dashed red cycles return it to the Engineer on a review change request or a staging defect. Humans are reachable at every step through a *team manifest*, and the final merge approval stays with a human.

of already-resolved issues compared against the recorded human implementation, while the *prospective* protocol follows current, never-before-resolved issues end-to-end. The retrospective protocol can be run at scale, since the issues are already resolved.

Due to the complexity of a corporate environment, we do not use coarse success or failure labels. Instead, we measure each run along several axes. They include the number of times the work loops through the stages, review and test cycle, how much of the acceptance criteria it satisfies, the token cost, the wall-clock time, and a final human rating of how well the delivered work fulfills the task. The central, fine-grained metric is *acceptance-criteria coverage*: each criterion produced by the **Product** agent becomes a deterministic test, and coverage is reported in bands, e.g. 90–100%, 70–89%, 50–69% and below 50%, so progress is legible without relying on binary test suites (success/failure) or unstable LLM judges. To keep this metric from rewarding criteria the system itself made too easy, the criteria will be first validated by a human technical reviewer who will score them from 0 to 10. Only criteria that pass this human check count toward coverage, and in the retrospective protocol the recorded human resolution provides a further reference for what they should have required.

4.2 Expected Obstacles

Real issues in a corporate environment tend to be noisier than benchmark tasks, because descriptions are incomplete and acceptance criteria may be wrong. As a consequence, part of the flow's quality depends on the **Product** agent and the associated human review. A cost and cycle limit per issue must be enforced, as agent loops can consume unbounded tokens (Sections 2.1, 2.2). We therefore bound any return cycle between two agents at ten exchanges per issue: a change request that is still unsatisfied at that point is escalated to the human role named for it in the team manifest instead of looping silently, and the issue is recorded with the coverage it achieved rather than discarded, since its interactions still provide investigation material. Running on the version-control platform also exposes the flow to API rate limits, since agents read and write far more often than humans do. We mitigate this by batching calls, backing off exponentially on throttling responses and caching repository reads within a run, and we count API calls as a measured cost alongside tokens. Finally, results on our internal repositories may not generalize across companies due to size, team dynamics, business market and so on. The contribution we aim for is the architecture and the results of the evaluation protocol, with our deployment as evidence of feasibility. A fuller treatment of these limitations is left to the future full paper.

5 Conclusion and Future Work

This short paper proposes a role-based multi-agent architecture (**Product**, **Architect**, **Engineer**, **Reviewer** and **QA**) for corporate software delivery, grounded in a survey of LLM-based agent approaches, that communicates through the version-control platform, validates in stages and keeps humans reachable at every step. Our proposed evaluation, based on real issues and acceptance-criteria coverage, will be conducted in a corporate scenario rather than the lab-like setups where previous building blocks were tested, aiming for a traceable, human-supervised delivery flow with integrated human–AI collaboration on real infrastructure. As future work, we plan to run a pilot with the **Engineer** and **QA** pair on a single repository, operationalize the handoff artifacts (user story, plan, pull request, test report) and instrument the internal benchmark, before extending the flow to the full five-role architecture.

Artifact Availability

This paper contributes an architectural proposal and its evaluation design, with no implementation yet, so no research artifact accompanies it. The agent prompts, the team-manifest schema and the evaluation package will be released with the full paper that reports the results of the protocol described in Section 4.

References

[1] Adam Alami, Victor Jensen, and Neil Ernst. 2025. Accountability in Code Review: The Role of Intrinsic Drivers and the Impact of LLMs. *ACM Trans. Softw. Eng. Methodol.* 34, 8, Article 233 (Oct. 2025), 44 pages. doi:10.1145/3721127

[2] Fredy Antonio Almeyda Alania, Alfredo Barrientos Padilla, and Ronald Gustavo Siancas Garay. 2025. Engineering Prompt-Orchestrated LLM Workflows for Automated Test Case Generation in Agile Environments. *International Journal of Advanced Computer Science and Applications* 16, 12 (2025). doi:10.14569/IJACSA.2025.0161272

[3] Eranga Bandara, Ross Gore, Xueping Liang, Sachini Rajapakse, Isurunima Kularathne, Pramoda Karunarathna, Peter Foytik, Sachin Shetty, Ravi Mukkamala, Abdul Rahman, Amin Hass, Ng Wee Keong, Kasun De Zoysa, Aruna Withanage, and Nilaan Loganathan. 2025. Agentsway – Software Development Methodology for AI Agents-based Teams. arXiv:2510.23664 [cs.SE] https://arxiv.org/abs/2510.23664

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG] https://arxiv.org/abs/2107.03374

[5] Antoine Chevrot, Alexandre Vernotte, Jean-Rémy Falleri, Xavier Blanc, and Bruno Legeard. 2025. Are Autonomous Web Agents Good Testers? arXiv:2504.01495 [cs.SE] https://arxiv.org/abs/2504.01495

[6] CrewAI, Inc. [n. d.]. CrewAI: Framework for Orchestrating Role-Playing, Autonomous AI Agents. https://github.com/crewAIInc/crewAI.

[7] Burak Gülmez. 2026. Code generation with large language models: a survey from neural program synthesis to autonomous software development. *Applied Intelligence* 56, 6 (2026), 200. doi:10.1007/s10489-026-07230-0

[8] Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. 2026. SWE-Skills-Bench: Do Agent Skills Actually Help in Real-World Software Engineering? arXiv:2603.15401 [cs.SE] https://arxiv.org/abs/2603.15401

[9] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352 [cs.AI] https://arxiv.org/abs/2308.00352

[10] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770 [cs.CL] https://arxiv.org/abs/2310.06770

[11] Jens Kohl, Otto Kruse, Youssef Mostafa, Andre Luckow, Karsten Schroer, Thomas Riedl, Ryan French, David Katz, Manuel P. Luitz, Tanrajbir Takher, Ken E. Friedl, and Céline Laurent-Winter. 2026. Automated structural testing of LLM-based agents: methods, framework, and case studies. arXiv:2601.18827 [cs.SE] https://arxiv.org/abs/2601.18827

[12] Ravikanth Konda. 2025. Agentic AI for Software Development: Autonomous Agents in Requirements Engineering, Testing, and Deployment. *International Journal of Emerging Research in Engineering and Technology* (Nov. 2025), 139–148. doi:10.63282/3050-922X.AECTIC-118

[13] LangChain, Inc. [n. d.]. LangGraph: Low-Level Orchestration Framework for Building Stateful Agents. https://github.com/langchain-ai/langgraph.

[14] Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. 2024. A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth* 1, 1 (2024), 9. doi:10.1007/s44336-024-00009-2

[15] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. 2026. Large Language Model-Based Agents for Software Engineering: A Survey. *ACM Trans. Softw. Eng. Methodol.* (March 2026). doi:10.1145/3796507 Just Accepted.

[16] Michele Merler, Rangeet Pan, Rahul Krishna, Tin Kam Ho, Raju Pavuluri, and Maja Vukovic. 2026. Usage, Effects and Requirements for AI Coding Assistants in the Enterprise: An Empirical Study. In *Proceedings of the 3rd International Workshop on Large Language Models For Code (LLM4Code '26)*. ACM, New York, NY, USA, 97–104. doi:10.1145/3786181.3788727

[17] Nesreen Otoum and Nuha Elkhalili. 2026. Methods and Techniques of Agentic Software Engineering: A Systematic Literature Review. *IEEE Access* 14 (2026), 7443–7465.

[18] Chanathip Pornprasit and Chakkrit Tantithamthavorn. 2024. Fine-tuning and prompt engineering for large language models-based code review automation. *Information and Software Technology* 175 (2024), 107523. doi:10.1016/j.infsof.2024.107523

[19] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative Agents for Software Development. arXiv:2307.07924 [cs.SE] https://arxiv.org/abs/2307.07924

[20] Giovanni Quattrocchi, Liliana Pasquale, Paola Spoletini, and Luciano Baresi. 2025. Can LLMs Generate User Stories and Assess Their Quality? arXiv:2507.15157 [cs.SE] https://arxiv.org/abs/2507.15157

[21] Thanigaivel Rangasamy. 2026. Agentic AI-Driven Product Development Life Cycle. *Journal of Computational Analysis & Applications* 35, 1 (2026), 832.

[22] Jessica Rawson and Sandeep Reddivari. 2025. A ChatGPT-powered Prompt Engineering Framework for Generating Software Acceptance Criteria. In *Proceedings of the 2025 ACM Southeast Conference* (Southeast Missouri State University, Cape Girardeau, MO, USA) *(ACMSE 2025)*. Association for Computing Machinery, New York, NY, USA, 282–288. doi:10.1145/3696673.3723078

[23] Wannita Takerngsaksiri, Jirat Pasuksmit, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Ruixiong Zhang, Fan Jiang, Jing Li, Evan Cook, Kun Chen, and Ming Wu. 2025. Human-In-the-Loop Software Development Agents. arXiv:2411.12924 [cs.SE] https://arxiv.org/abs/2411.12924

[24] Fernando Vallecillos Ruiz. 2024. Agent-Driven Automatic Software Improvement. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024)*. ACM, New York, NY, USA, 470–475. doi:10.1145/3661167.3661171

[25] Fanyu Wang, Chetan Arora, Yonghui Liu, Kaicheng Huang, Chakkrit Tantithamthavorn, Aldeida Aleti, Dishan Sambathkumar, and David Lo. 2025. Multi-Modal Requirements Data-based Acceptance Criteria Generation using LLMs. arXiv:2508.06888 [cs.SE] https://arxiv.org/abs/2508.06888

[26] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345. doi:10.1007/s11704-024-40231-1

[27] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155 [cs.AI] https://arxiv.org/abs/2308.08155

[28] Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. 2025. TheAgentCompany: Benchmarking LLM Agents on Consequential Real World Tasks. arXiv:2412.14161 [cs.CL] https://arxiv.org/abs/2412.14161

[29] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793 [cs.SE] https://arxiv.org/abs/2405.15793

[30] Zheying Zhang, Maruf Rayhan, Tomas Herda, Manuel Goisauf, and Pekka Abrahamsson. 2024. LLM-Based Agents for Automating the Enhancement of User Story Quality: An Early Report. 117–126 pages. doi:10.1007/978-3-031-61154-4_8