

Agentic Software Engineering: Measuring and Improving AI-Generated Code Quality

Antonino Coppola

University of Southern Denmark, Vejle, Denmark

Abstract

The increasing adoption of agentic AI systems in software engineering raises significant concerns about the structural quality and long-term maintainability, with a particular focus on testability as a key sub-characteristic, of the code they produce. While existing research has extensively examined functional correctness and developer productivity in LLM-based code generation, the evolution and long-term consequences of agent-generated code for software engineering practice remain largely unexplored. This PhD research addresses that gap by combining mining of software repositories, controlled experiments, and structural code analysis to empirically characterize agent-generated code and to develop techniques that improve its maintainability and testability. The work is structured around four research questions that progressively investigate: (i) the state of the art in agentic coding systems; (ii) their impact on structural quality metrics; (iii) the role of test-driven constraints in elevating output quality; (iv) and the identification of recurring anti-patterns in agent-generated code. Early outcomes include a Registered Report on the sustainability and evolution of agent-generated code (accepted at ESEM) and an exploratory study on coding agents' issue-localization accuracy (under review). The expected contributions include a multidimensional evaluation framework for agent-generated code quality, empirical evidence on the differences between human-written and agent-generated software, a taxonomy of structural anti-patterns, and practical guidelines for the responsible adoption of agentic AI in software engineering practice.

Keywords

Agentic AI, Software Engineering, AI-Generated Code, Code Quality, Testability, Software Maintenance

1. Introduction and Background

The emergence of Large Language Models (LLMs) such as ChatGPT, Claude, and Llama has profoundly transformed software development practices, particularly in the area of code generation. These models are increasingly capable of producing executable code, assisting developers during implementation, and automating several software engineering tasks. As a consequence, the role of developers is progressively shifting from direct code production toward higher-level supervision, validation, and orchestration of AI-generated artifacts [1].

This transformation has been further accelerated by the rise of agentic AI systems [2], including tools such as Devin, Claude Code, and AutoGPT, which extend traditional LLM capabilities with autonomous reasoning and execution mechanisms enabling them to plan tasks, invoke external tools, iteratively debug and refine solutions [3].

Despite their potential, significant concerns remain regarding the robustness, security, and trustworthiness of AI-generated software artifacts. LLMs are commonly trained on massive code repositories that may contain noisy, suboptimal, duplicated, or vulnerable code originating from human development practices [4]. Consequently, these models may learn and reproduce existing defects, insecure patterns, and architectural weaknesses embedded in their training data. Recent studies have also highlighted the tendency of LLMs to hallucinate code, generate logically inconsistent implementations, or produce syntactically correct but semantically flawed solutions, especially in complex development scenarios involving contextual dependencies and multi-step reasoning [5].

These limitations become even more critical in the context of agentic AI systems, where autonomous decision-making and iterative code generation can amplify quality degradation over time. While current

IDOESE 2026: International Doctoral Symposium on Empirical Software Engineering, October 07, 2026, Munich, DE

✉ antonino@imada.sdu.dk (A. Coppola)

🆔 0009-0006-0316-8433 (A. Coppola)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

research has extensively investigated the functional correctness and productivity impact of LLM-based code generation, comparatively limited attention has been devoted to understanding the structural quality, maintainability, and architectural implications of agent-generated software [6].

This PhD research investigates the quality of code produced by agentic AI systems and develops methods and tools to improve its maintainability and testability. Using empirical software engineering, repository mining, controlled experiments, and structural analysis, it systematically compares human-written, AI-assisted, and fully agent-generated code. It also identifies recurring structural patterns and anti-patterns in AI-generated software and designs validation mechanisms that steer agentic workflows toward higher-quality outputs. The work delivers both scientific evidence and practical tooling for evaluating and governing agentic AI in software engineering, fostering more maintainable and trustworthy AI-generated artifacts.

2. Research Design and Methodology

This PhD research proposes a novel framework and an empirical evaluation methodology to enhance the maintainability and testability of the code generated by agentic AI. By combining empirical investigation, controlled experimentation, and structural analysis, the work aims to produce both scientific evidence on the behavior of agentic systems and reusable methodological artifacts.

This general aim is articulated through three specific objectives:

- **O.1** Characterize the current state of agentic AI tools in software development.
- **O.2** Identify the differences in testability and structural quality between human-only, AI-assisted, and agent-generated code across software engineering tasks (fixing, refactoring, and feature implementation), using an agent-based prototype .
- **O.3** Provide a profiling of the recurring patterns and anti-patterns in LLM-generated code.

2.1. Research Questions

We formulate the following research questions (RQs) aligned with the goal:

RQ₁: What capabilities and limitations of agentic AI systems for software maintenance and evolution have been identified by researchers and practitioners?

There has been several research contribution in the form of systematic literature reviews and surveys on LLM-based code generation, some of them [7] mapped research trends and datasets across 58 primary studies, identifying the lack of alignment with ISO 25010 as an open gap while others [8] provided a broad narrative survey of LLMs and autonomous AI agents across multiple application domain. However, none of these works consolidate empirical evidence on the quality of the generated code through a structured framework, nor incorporate grey literature from industry practice. Our proposed study aims to bridge this research gap. Following the approach established in prior work [9], our study will combine LLM-assisted automatic analysis with manual expert validation, ensuring systematic coverage of both the academic and grey literature sources from industry practice to produce a multivocal literature review (MLR). The outcome of this RQ will be a structured characterization of the state of the art, which will inform the design of subsequent studies (RQ2–RQ4).

RQ₂: What impact does the source of code generation (Human Only, AI Assisted, or Agent Generated) have on maintainability, branch coverage, mutation score, cohesion, coupling, and complexity of software?

A recent large-scale study confirms that AI-generated code introduces measurable technical debt that persists after deployment [10]. However, it does not distinguish AI-assisted from agent-generated

code, lacks a human-only baseline for direct comparison, and relies on issue frequency rather than structural metrics such as cohesion, coupling, and testability. To answer this question, we propose combining MSR methods with a controlled experiment. The two methods are complementary: MSR provides large-scale, ecologically valid evidence, while controlled experiments enable causal inference by controlling for confounders. The study would involve examining open-source code repositories in the wild and comparing code generated using three approaches (Human Only, AI-Assisted, and Agent-Generated). For the agent-generated condition, we will use the agent-based prototype developed from the MLR findings to generate code autonomously on the selected tasks. Repositories will be selected from the SQuAD dataset [11], applying filtering criteria to ensure data quality and comparability. For each approach, a suite of metrics will be computed, including testability metrics such as branch coverage, mutation score, and complexity, cohesion, and coupling measures. The metric trajectory will be examined at the commit and release levels to compare patterns of technical debt accumulation across the three groups.

RQ₃: Does the incorporation of test-driven constraints in the agentic code generation workflow improve testability and structural quality?

We hypothesize that the lack of quality constraints during the agentic code generation is one reason for the observed degradation in testability and structural quality of agent-generated code. Prior work has used interactively generated tests to disambiguate intent and improve functional accuracy [12], but in human-in-the-loop settings targeting correctness rather than structural quality. By test-driven rules, we mean specific constraints enforced during code generation, such as test-first generation (the agent must produce unit tests before implementation), minimum coverage thresholds (the agent must achieve a target branch coverage), or failing-test feedback (the agent uses test failures to guide iterative refinement). We believe that incorporating test-driven rules as constraint guards will help agentic systems generate testable and high-quality code. To test this hypothesis, we will use the agent-based prototype as the experimental platform, running the same maintenance tasks under two conditions: a baseline without test-driven constraints, and a treatment where such constraints are enforced. To account for variability across AI models, the experiment will be replicated using multiple LLMs, with model treated as a factor in the statistical analysis. The prototype’s outputs will be evaluated using the same metrics defined in RQ2, and compared against human-written baselines.

RQ₄: What structural patterns and anti-patterns are present in code generated by agentic AI systems?

Anti-patterns are quick solutions that have negative consequences or cause significant risks in software development, leading to low maintainability, refactoring difficulty, and high technical debt. The recurring nature of anti-patterns means they are not isolated incidents but recurring patterns developers encounter across different projects and contexts. Recent work has demonstrated the feasibility of using LLMs to detect anti-patterns in software code from a predefined, curated taxonomy through retrieval-augmented prompting pipelines [13]. This research question aims to empirically derive a taxonomy of structural patterns and anti-patterns directly from agent-generated artifacts. The corpus will be collected from the RQ2 and RQ3 experiments, ensuring that code generated by multiple agents and models is represented, so that the resulting taxonomy is not biased toward a single system.

2.2. Research Plan and Expected Contributions

The research is organized into three phases: systematic characterization, implementation and evaluation, and pattern extraction with benchmark development, as explained in Table 1. This PhD research aims to contribute to both the scientific understanding and the practical governance of agentic AI systems in software engineering. The expected contributions of the research are the following:

- A systematic characterization of the capabilities, limitations, and quality concerns of agentic AI

	RQ ₁	RQ ₂	RQ ₃	RQ ₄
O.1	●			
O.2		●	●	
O.3				●

Figure 1: Alignment matrix between research objectives and research questions.

systems for software engineering tasks through a MLR and empirical investigation.

- A multidimensional evaluation framework for assessing the structural quality, maintainability, and testability of AI-generated code, combining static, dynamic, and process-related metrics.
- Empirical evidence on the differences between human-written, AI-assisted, and fully agent-generated software artifacts across maintenance and evolution tasks such as bug fixing, refactoring, and feature implementation.
- A taxonomy of recurring structural patterns and anti-patterns in AI-generated code, supporting the identification of quality degradation phenomena and technical debt accumulation.
- An extensible prototype integrating agentic AI workflows with automated validation and quality assessment mechanisms to improve the maintainability and testability of generated code.
- Public datasets, experimental pipelines, and benchmarking artifacts to support reproducibility and future research on agentic software engineering.
- Practical guidelines and recommendations for researchers and practitioners on the evaluation, governance, and safe adoption of agentic AI systems in software development.

3. Challenges and Limitations

This PhD research addresses a rapidly evolving domain characterized by methodological, technical, and practical challenges. The following limitations are anticipated, together with corresponding mitigation strategies. These challenges concern aspects such as data representativeness, reproducibility of LLM-based experiments, evaluation of software quality, measurement of agentic behavior, computational scalability, prototype generalizability, and ethical implications. Table 2 summarizes the main anticipated challenges together with the corresponding mitigation strategies adopted throughout the PhD research.

4. Current Status

Since the start of this PhD research, several activities have been undertaken in alignment with the research plan outlined in Section 2, spanning the early stages of Phase 1. While most of these activities are still ongoing, they have already produced initial research outputs.

The first activity initiated in direct response to RQ₁ was a MLR, currently underway, aimed at consolidating existing evidence on the capabilities and limitations of agentic AI systems in software

Table 1
Research Plan Overview

Activities	Deliverables
Phase 1: Systematic Characterization (Year 1)	
- Conduct a multi-vocal literature review combining academic papers and grey literature - Perform opinion mining on Stack Overflow, GitHub Issues, and developer forums - Identify key technical gaps: precision, context management, controllability, and reliability - Define representative software engineering tasks - Design and develop an initial LLM/Agent-based prototype with context-aware mechanisms	- Structured state-of-the-art and gap analysis - Dataset of developer perceptions and issues - Initial prototype integrating LLMs and agentic workflows - First publication (literature review or vision paper)
Phase 2: Impact on Testability and Code Quality (Year 2)	
- Design controlled experiments comparing: - Human-only development - AI-assisted development - Fully agent-generated code - Define and operationalize metrics for testability and code quality - Build an automated validation pipeline - Conduct empirical studies on bug fixing, refactoring, and feature implementation tasks - Extend the prototype with an automated testability assessment module	- Reproducible experimental dataset and validation pipeline - Quantitative evidence on testability and structural quality - Extended prototype with assessment capabilities - Publications (empirical study and tool paper)
Phase 3: Patterns, Anti-patterns, and Benchmarking (Year 3)	
- Apply mining software repositories (MSR) using AST and graph-based representations - Identify recurring patterns and anti-patterns in AI-generated code - Develop computational rules to automatically detect structural issues - Build a labeled dataset of AI-generated code annotated with quality metrics - Design and release a public benchmark framework for evaluating AI agents	- Catalog of patterns and anti-patterns in AI-generated code - Public datasets and benchmarking framework - Final integrated prototype and validated guidelines - Final publications and PhD thesis

engineering. The MLR combines systematic analysis of academic literature with grey literature from industry sources. The study aims to identify open gaps regarding the structural quality and maintainability of AI-generated code, providing the empirical grounding for the subsequent investigations planned in RQ2-RQ4.

In parallel, a Registered Report titled *AI Writes Code, Humans Pay the Debt. An Empirical Study on the Sustainability and Evolution of Agent-Generated Code* was designed and submitted to ESEM, where it was accepted. The study, which is currently in its data collection and analysis phase, directly addresses RQ2 and RQ3 and will conduct a large-scale MSR study using the SQuAD dataset [11]. Agent-based implementations will be generated and compared against actual developer commits using static analysis metrics, at both commit and release levels, across multiple LLMs. The study is expected to provide empirical evidence on the trade-offs introduced by agent-based development, including differences in localization accuracy and variations in technical debt introduction.

In addition, an Emerging Results paper titled *On Coding Agent Issue Localization Accuracy. An*

Challenges	Mitigation Strategy
Model Instability & Reproducibility	• Use versioned models and datasets whenever possible • Log prompts, configurations, and outputs for traceability • Design experiments with multiple runs to account for variability • Define standardized evaluation pipelines • Prefer open-weight models from repositories such as HuggingFace • Favor lightweight models (7–40B parameters) when possible to improve reproducibility on commodity hardware
Testability & Code Quality Evaluation	• Define a multi-dimensional evaluation framework including: – Static metrics (complexity, coupling, cohesion) – Dynamic metrics (coverage, mutation score) – Process metrics (reproducibility, edit distance) • Complement automated metrics with qualitative analysis where necessary
Measurement of Agentic Behavior	• Provide operational definitions of agentic behavior (autonomy, goal completion, adaptability) • Develop evaluation criteria covering functional and non-functional aspects • Analyze agent behavior across multiple tasks and contexts • Incorporate explainability mechanisms where possible
Ethical & Reliability Concerns	• Integrate validation mechanisms (compiler checks, static analysis, test-based validation) • Promote human-in-the-loop strategies • Ensure transparency of experimental design and results • Consider ethical implications within the evaluation framework • Share anonymized aggregated data while protecting developers' identities
Limitations	Mitigation Strategy
Data Collection & Representativeness	• Select diverse repositories across domains, sizes, and maturity levels • Apply filtering criteria to ensure data quality and consistency • Complement repository mining with grey literature and developer discussions • Explore industry collaborations to validate findings in real-world settings
Computational Resources & Scalability	• Leverage cloud-based infrastructures for scalable experimentation • Use smaller or optimized models where appropriate • Apply sampling strategies to reduce computational overhead • Design efficient automated evaluation pipelines
Generalizability of the Prototype	• Clearly define the scope and assumptions of the prototype • Evaluate the prototype across multiple tasks and datasets • Emphasize modularity and extensibility in the system design • Provide adaptation guidelines for different contexts

Table 2

Main challenges and limitations of the proposed PhD research together with mitigation strategies.

Exploratory Study was prepared and submitted to ESEM. The study analyzed 10 large-scale Java projects and evaluated three agents (each combining the OpenCode¹ harness with a different open-weight LLM) on 2,441 issues from the SQuAD dataset [11]. Agent-modified entities were compared against human-implemented fixes at the package, class, and method levels using Accuracy, Precision, Recall, F1-score, and MCC. Results indicate that agents partially identified the software entities requiring modification, but localization performance strongly depended on granularity, with the strongest results at the package level and progressive degradation at finer levels. The paper has been submitted and the outcome is currently awaited.

Finally, a five-week research visit at the University of Oulu, Finland (25th May – 30th June 2026) was undertaken as part of an ongoing international collaboration on Generative AI for long-term system evolution and maintenance. Research activities, still in progress at the time of writing, include the joint design of AI-driven approaches, the analysis of software evolution data, and the empirical validation of proposed methods addressing challenges related to architectural degradation and technical debt.

Declaration on the use of generative AI The author used ChatGPT for suggestions on improving textual clarity. The entire manuscript was written by the author

References

- [1] V. Acharya, Generative ai and the transformation of software development practices, 2025. URL: <https://arxiv.org/abs/2510.10819>. arXiv: 2510.10819.
- [2] OpenAI, Practices for Governing Agentic AI Systems, Technical Report, OpenAI, 2023. URL: <https://openai.com>, white Paper.
- [3] H. Wang, J. Gong, H. Zhang, J. Xu, Z. Wang, Ai agentic programming: A survey of techniques, challenges, and opportunities, 2025. URL: <https://arxiv.org/abs/2508.11126>. arXiv: 2508.11126.
- [4] D. Kocetkov, R. Li, L. B. Allal, J. Li, C. Mou, C. M. Ferrandis, Y. Jernite, M. Mitchell, S. Hughes, T. Wolf, D. Bahdanau, L. von Werra, H. de Vries, The stack: 3 tb of permissively licensed source code, 2022. URL: <https://arxiv.org/abs/2211.15533>. arXiv: 2211.15533.
- [5] Z. Zhang, C. Wang, Y. Wang, E. Shi, Y. Ma, W. Zhong, J. Chen, M. Mao, Z. Zheng, Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation, Proc. ACM Softw. Eng. 2 (2025). URL: <https://doi.org/10.1145/3728894>. doi:10.1145/3728894.
- [6] M. Kharma, S. Choi, M. AlKhanafseh, D. Mohaisen, Security and quality in llm-generated code: A multi-language, multi-model analysis, 2026. URL: <https://arxiv.org/abs/2502.01853>. arXiv: 2502.01853.
- [7] Umama, K. Usman Danyaro, M. Nasser, A. Zakari, S. Abdullahi, A. Khanzada, M. Muntasir Yakubu, S. Shoaib, Llm-based code generation: A systematic literature review with technical and demographic insights, IEEE Access 13 (2025) 194915–194939. doi:10.1109/ACCESS.2025.3631952.
- [8] M. A. Ferrag, N. Tihanyi, M. Debbah, From llm reasoning to autonomous ai agents: A comprehensive review, 2026. URL: <https://arxiv.org/abs/2504.19678>. arXiv: 2504.19678.
- [9] R. Su, N. Ahmad, M. Esposito, A. Janes, D. Taibi, V. Lenarduzzi, Emerging trends in software architecture from the practitioner’s perspective: A five-year review, Journal of Systems and Software 236 (2026) 112820. doi:<https://doi.org/10.1016/j.jss.2026.112820>.
- [10] Y. Liu, R. Widyasari, Y. Zhao, I. C. Irsan, J. Chen, D. Lo, Debt behind the ai boom: A large-scale empirical study of ai-generated code in the wild, 2026. URL: <https://arxiv.org/abs/2603.28592>. arXiv: 2603.28592.
- [11] M. Robredo, M. Esposito, D. Taibi, R. Peñaloza, V. Lenarduzzi, Squad: The software quality dataset, 2025. URL: <https://arxiv.org/abs/2511.11265>. arXiv: 2511.11265.
- [12] S. Fakhoury, A. Naik, G. Sakkas, S. Chakraborty, M. Musuvathi, S. Lahiri, Exploring the effectiveness of llm based test-driven interactive code generation: User study and empirical evaluation, in: Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering:

¹<https://opencode.ai/>

Companion Proceedings, ICSE-Companion '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 390–391. URL: <https://doi.org/10.1145/3639478.3643525>. doi:10.1145/3639478.3643525.

- [13] M. T. Kanse, E. Fallon, T. B. Rodrigues, Comparison of large language models for software anti-patterns detection, in: 2026 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA), 2026, pp. 1–6. doi:10.1109/ACDSA67686.2026.11467800.