Beyond Task Completion: Product and Human-Agent Process Quality in Agentic Software Engineering

Rubens Abraão da Silva Sousa
State University of Ceará
Fortaleza, Ceará, Brazil
abraao.sousa@aluno.uece.br

Gabriel Pinheiro Rezende de Lima
State University of Ceará
Fortaleza, Ceará, Brazil
gabriel.rezende@aluno.uece.br

Evellin Moura
State University of Ceará
Fortaleza, Ceará, Brazil
evellin.moura@aluno.uece.br

Sávio Freire
Federal Institute of Ceará
Morada Nova, Ceará, Brazil
State University of Ceará
Fortaleza, Ceará, Brazil
savio.freire@uece.br

Ismayle de Sousa Santos
State University of Ceará
Fortaleza, Ceará, Brazil
ismayle.santos@uece.br

Paulo Henrique Mendes Maia
State University of Ceará
Fortaleza, Ceará, Brazil
pauloh.maia@uece.br

Abstract

Large Language Model-based agents are increasingly used to plan, modify, test, and review software artifacts across the software development life cycle. Current evaluations, however, still emphasize task completion, benchmark scores, and final artifact correctness, leaving the human-agent process that produces those artifacts largely unobserved. This vision paper argues that agentic software engineering requires process-quality instruments capable of capturing how agentic decisions are generated, reviewed, trusted, and attributed in real development workflows. We propose four constructs for human-agent process quality: decision traceability, human review workload, trust calibration, and responsibility attribution. For each construct, we outline operational definitions, candidate metrics, possible measurement instruments, and empirical anchors from recent software engineering studies. We argue that these constructs are not independent: traceability conditions responsibility, while review workload and trust calibration may create tensions between efficiency and defect detection. By shifting the evaluation focus from task success alone to observable human-agent process quality, this paper outlines a research agenda for more robust, accountable, and governable agentic software engineering.

Keywords

Agentic Software Engineering, Human-Agent Collaboration, Software Development Life Cycle

1 Introduction

Large Language Model (LLM)-based agents have expanded the role of artificial intelligence (AI) in software development. The agents are able to combine perception, memory, planning, and action to operate on software artifacts [14], iterative tool use, and the capability to execute partially autonomous actions within development environments and artifacts. Liu et al. [5] organize the field from two complementary perspectives: i) the software engineering (SE) tasks addressed by agents and ii) the agentic capabilities that sustain their performance throughout the software development life cycle (SDLC).

These perspectives differentiate LLM-based agents from traditional coding assistants, describing SE agents as capable of using tools, executing commands, observing environment responses, and planning actions that may provide benefits throughout the SDLC [15]. This mode of operation shifts agents from a suggestive role to direct action in coding, testing, elicitation, and communication tasks within development tools. The performance of these agents is already evident in different phases of the SDLC. Takerngsaksiri et al. [12], for instance, present a human-in-the-loop framework in which agents support planning and refinement based on requirements specifications, reinforcing that agentic SE spans the entire SDLC.

Recent evaluations [6, 9] demonstrate that LLM-based agents succeed in solving development tasks, yet they also reveal limitations regarding quality: i) Semantically equivalent changes in the input description generate modifications in recommendations and impact the correctness of the generated code [6], and ii) Code produced with agent assistance was significantly less secure regarding security tasks [9]. Thus, it is established that the completion of a task by agents does not, by itself, guarantee the quality of what is being generated throughout the SDLC.

Software product quality remains a central axis for evaluating the effects on agentic SE. ISO/IEC 25010:2023 [4] defines a quality model composed of characteristics and sub-characteristics applicable to software and ICT (Information and Communication Technology). However, the model focuses predominantly on the evaluation of the software artifact and provides limited support for assessing emerging aspects of human-agent collaboration during software development. In agentic environments, factors such as human review workload, traceability of delegated decisions, and trust calibration become part of the quality construction process itself, making attributes such as reliability, maintainability, testability, and security dependent on decisions made during human-agent-tool interaction [2, 7, 10].

This vision paper argues that evaluations centered exclusively on task resolution or the final quality of the artifact are insufficient for agentic SE. The presence of agents in the SDLC introduces new sociotechnical dimensions related to human supervision, decision-making traceability, calibrated trust, and responsibility for changes

made to software artifacts. Therefore, agents should not be evaluated solely for resolving an issue, generating a patch, or passing existing tests, but also for the impact of their actions on sociotechnical aspects that are definitely affect the final artefact.

This implies recognising that part of the quality emerges not only from the produced artefact but also from the dynamics of coordination, supervision, and decision-making between humans and agents. Not only the final product is important, but also the process through which it is created. When agents propose plans, modify code, execute tests, or suggest fixes, quality emerges from the interaction between agents, human supervision mechanisms, and the criteria used for the decision-making of accepting, rejecting, or modifying their suggested contributions.

The remainder of this paper is organized as follows: Section 2 characterizes agentic SE and the performance of LLM-based agents in the SDLC. Section 3 discusses the limits of evaluations centered on task resolution. Section 4 presents a conceptual agenda for evaluating product quality and human-agent collaboration quality. Lastly, Section 5 synthesizes the defended position and presents implications for future research.

2 Agentic Software Engineering Across the Software Development Life Cycle

The incorporation of LLMs as the controlling core of autonomous agents represents a structural transformation in the way SE tasks are conducted. Unlike reactive LLMs, which operate statically on textual inputs, LLM-based agents expand their capabilities by integrating four fundamental components: perception, memory, planning, and action [5, 14]. These components enable agents to perceive the state of the development environment, decompose complex tasks into subtasks, keep records of previous decisions, and execute concrete actions [5]. This capacity for iterative interaction with dynamic environments distinguishes agents from reactive coding assistants and positions agentic SE as an operationally distinct paradigm [15].

From the SDLC, LLM-based agents have been applied to wide range of of activities, from requirements engineering to continuous software maintenance. In a review of 124 studies, Liu et al. [5] identified that research efforts are primary concentrated on code generation, testing, and static verification. Complementing this perspective, Wang et al. [14] characterize perception modules as foundations for agents' contextual understanding capacity across different development phases. Building on these advances, Terragni et al. [13] propose a view in which specialized agents act in an orchestrated manner across all phases of the SDLC, from the elicitation and validation of requirements to the generation and maintenance of test code, establishing a partnership between developers and AI systems.

The role of agents throughout the SDLC is not restricted to isolated tasks. Multi-agent systems have demonstrated the capacity to support complete software development and maintenance processes. Liu et al. [5] observe that, in multi-agent systems, each agent assumes specialized roles, coordinating through collaborative or competitive communication mechanisms to produce artifacts of greater complexity. In requirements tasks, multi-agent systems such as MARE and Elicitron coordinate elicitation, modeling, negotiation, and verification [5], while Xia et al. [15] show that even non-agentic approaches can achieve competitive performance on issue-resolution benchmarks — indicating that architectural complexity is neither sufficient nor necessary for effectiveness in individual SDLC tasks.

However, human-agent collaboration constitutes a transversal and structurally relevant dimension across all phases of the SDLC. Liu et al. [5] identified four predominant points of human integration in agentic systems: (i) the planning phase, in which developers can review and modify automatically generated plans; (ii) the requirements phase, in which ambiguities are resolved through clarifying questions directed to the user; (iii) the development phase, in which engineers can iteratively edit generated code; and (iv) finally, the evaluation phase, in which results are manually validated before being incorporated into the final artifact.

Terragni et al. [13] emphasize that, in the medium term, software engineers will remain indispensable for understanding, reviewing, combining, and validating the contributions of AI systems, acting as critical supervisors whose judgment capacity cannot be replicated by autonomous agents. It is from this conjunction between agentic autonomy and human supervision that central questions about quality in agentic software development emerge.

3 Why is Success in the Task not Enough?

The benchmarks that organize the evaluation of agents in SE reduce performance to a single measure: the fraction of completed tasks. Aleithan et al. [1] established this format with real GitHub issues, in which the agent generates a patch evaluated by tests that move from failure to success. Liu et al. [5], in turn, confirmed that code generation and validation concentrate the greatest volume of evaluation effort in the field, with *issue* resolution as the reference task. The measure is binary and falls entirely on the final artifact: whether or not it resolves the proposed issue.

This single measure does not hold during the development process. Aleithan et al. [1] reexamined the benchmark and found solution leakage in the descriptions and weak test suites. Some of the solutions considered correct reproduced answers already present in the prompt, and insufficient tests approved defective *patches*. Under validation, the resolution rate dropped from 12.5% to 4%. If the count of resolved tasks is fragile with respect to the correctness of the artifact itself, it does not provide information about the process that produced it.

The traceability of decisions is invisible. In the benchmark, correctness is decided by automatic tests, without a developer who reviews, adjusts, or rejects the agent's contribution. Minh et al. [7] measured this effort based on 33,707 real *pull requests* authored by agents, rather than on a benchmark, finding that 20% of the contributions concentrate 69% of the review effort. The effort that determines the real cost of integrating the agent only appears when the evaluation leaves the controlled environment and observes human review.

Responsibility for the changes is not attributed to anyone. The benchmark credits the resolution to the evaluated system, without distinguishing what was proposed by the agent, reviewed by the human, and approved for integration. Stalnaker et al. [11] documented

Table 1: Human–agent process quality constructs: operational definition, candidate metric, instrument, and anchor study.

Construct	Operational Definition	Candidate Metric	Instrument	Anchor Study
Decision Traceability	The degree to which each artifact change can be linked to the agent's decision that originated it and to the recorded rationale for that decision.	Proportion of artifact changes with a corresponding and traceable decision node in the agent's execution trajectory (*decision coverage*).	Execution trajectory analysis; provenance graph linking prompt, plan, tool call, and *patch*.	Aleithan et al. [1] map trajectories of five agents and propose a taxonomy of decision paths. *Feasibility anchor; no consolidated coverage metric.*
Human Review Load	Effort that the developer expends to review, adjust, or reject the agent's contributions before integration.	Effort score per contribution (review and discussion volume); fraction of total effort concentrated in the tail of costly contributions.	Mining of *pull requests*; predictive effort model from structural signals (*patch* size, touched files).	Minh et al. [7] predict review effort with AUC 0.957; 20% of contributions concentrate 69% of the effort. *Direct evidence.*
Confidence Calibration	Correspondence between the confidence that the developer places in the agent's contribution and the effective reliability of that contribution.	Difference between objective reliability (defects, redundancy) and reviewer reaction (acceptance, sentiment) for the same contribution (*calibration gap*).	Paired analysis of code quality and reviewer sentiment on the same contributions.	Huang et al. [3] find agentic code more redundant with less negative reviewer sentiment. Sabouri et al. [10] define confidence as a state recalibrated during interaction. *Proxy via sentiment; calibration not measured directly.*
Change Accountability	Possibility of attributing each incorporated artifact change to a responsible human agent, based on an auditable record of authorship and approval.	Proportion of changes with declared and verifiable (human/agent) authorship and approver in the artifact history (*attribution completeness*).	Version history audit; human–agent authorship declaration protocol.	Stalnaker et al. [11] document developers' uncertainty regarding authorship and responsibility for AI-generated code. *Perception anchor; measurement instrument to be constructed.*

that developers do not know for certain to whom authorship and responsibility for AI-generated code should be attributed, an uncertainty that persists in real use and that resolution-based evaluation does not help to resolve, since it does not preserve the chain of authorship and approval.

Benchmarks offer a concrete advantage: once constructed, they can evaluate any agent at the same cost, reproducibly and without human intervention. Process-oriented evaluation does not provide the same economy, as it requires observing developers while they review, interpret, and take responsibility for agent-generated contributions.This efficiency, however, does not eliminate the need to study the process itself. Accurately assessing the correctness of an artifact provides little insight into the coordination mechanisms that produced it. In this regard, Mozannar et al. [8] demonstrated that gains in static benchmarks do not translate proportionally into developer productivity, and that assessing the human side required a closer study, centered on routine practice.

The objection that constructing a sufficiently adequate benchmark would be enough misinterprets the problem. A defining characteristic of benchmarks is that they provide a static snapshot established in advance, functioning as an automated oracle that replaces human judgment. As a result, the process itself is excluded from the measurement. One way to address this limitation is to define constructs that explicitly involve human supervision, decision-making, and accountability for contributions. A format that replaces human involvement with pre-defined tests cannot measure these constructs by design, not because of any engineering limitation. Making them observable requires a different class of instruments. Benchmarks measure outcomes and should continue to do so. The process, however, requires its own instruments, which are proposed in Section 4, where we define the relevant constructs and the means to make them observable.

4 Toward Quality-Aware Agentic Software Engineering

Evaluation centered on task resolution observes the integrated artifact and discards the process that produced it. Table 1 introduces four quality constructs in the human-agent process. For each one, an operational definition, a candidate metric, the corresponding measurement instrument, and a study anchoring its empirical feasibility are presented. The instrumentation available in the literature is uneven: review load already has validated metrics, whereas decision traceability and responsibility for changes remain the least measured. This asymmetry delimits the research frontier proposed by this research agenda.

The four constructs are not independent. **Decision traceability** is a condition for responsibility for changes; without linking the change to the decision that originated it, there is no basis for attributing responsibility for it. **Human review load** and **confidence calibration** form the second pair. Huang et al. [3] demonstrate that reviewers react with less rigor to agent contributions, which reduces the apparent review load while redundancy accumulates as technical debt. Reducing review load and calibrating trust are objectives in tension, not mutually reinforcing goals.

Change accountability differs from traceability: while traceability explains how a change was made, accountability identifies who is responsible for approving it and answering for its outcomes. In agentic software engineering, roles may be distributed, agents generate changes, developers revise them, and maintainers approve them, making provenance alone insufficient.

An accountable record should include the responsible actor, approver, scope of delegated authority, supporting evidence (e.g., rationale, tests, risks), and who handles failures. This motivates metrics

such as *accountability coverage* (changes with a designated human owner) and *orphan-change rate* (changes without clear ownership).

5 Research Agenda

The instrumentation of the four constructs is uneven, and this unevenness orders the research agenda. Human review load already has a validated predictive metric [7]. Trust calibration has evidence of misalignment, but no direct metric [3, 10]. Decision traceability has a taxonomy of trajectories, but no coverage metric [2]. Responsibility for changes has only perceptual evidence, with no instrument [11]. The last two define the frontier: they are named but not mechanically characterized.

The lack of these instruments imposes a cost in the present. During the development cycle, developers integrate agent contributions that they have not fully claimed: Huang et al. [3] show that review reacts with less rigor to agent code, while redundancy accumulates as technical debt. When a change fails, the record linking the failure to the decision that originated it is missing, as is the person to whom responsibility should be attributed. Stalnaker et al. [11] document this uncertainty in real use. The problem is not theoretical, it concerns the integration of origin and responsibility during the process.

We convert this gap (more details in Table 1) into the following measurable research questions (RQs):

- **RQ1:** How can decision coverage be operationalized as the proportion of eligible artifact changes linked to explicit agent decisions, adequate rationales, and supporting evidence?
- **RQ2:** What trade-off does model-guided review prioritization create between reviewer effort and severity-weighted defect-detection effectiveness?
- **RQ3:** To what extent is reviewers' explicitly stated confidence calibrated to the independently assessed correctness of agent-generated changes?
- **RQ4:** What provenance, approval, and control evidence is required to validly attribute human accountability for integrated agent-generated changes?

Answering these RQs encounters concrete obstacles. Measuring the process requires a human in the loop, which Section 3 demonstrated to be costly and slow compared with automatic benchmarks. The constructs are coupled: without traceability, there is no basis for attributing responsibility, so RQ4 depends on RQ1.

What remains unexplored is not each construct in isolation, but the tension between them. Reducing review load and calibrating trust may be opposed: filtering contributions to reduce review may allow precisely the code that breaks or remains defective in the application to pass through. This *trade-off* has not yet been observed. Characterizing it requires observing, in the same population, review effort, calibration, and the defect that escapes, a design that neither current benchmarks nor isolated mining studies accommodate.

The position of this vision paper defines a program, not a principle. Before evaluating agents by their process quality, the missing instruments for traceability and responsibility must be constructed and validated, and the tensions among the four constructs must then be measured in real development projects.

Artifact Availability

No artifacts were produced for release

Acknowledgments

This work was developed with support from the National Council for Scientific and Technological Development - CNPq (Process 312173/2025-3) and This study was financed in part by the Brazilian Federal Agency for Support and Evaluation of Graduate Education – CAPES – Finance Code 001

References

[1] Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. 2024. SWE-Bench+: Enhanced Coding Benchmark for LLMs. doi:10.48550/ARXIV.2410.06992 Version Number: 2.

[2] Ira Ceka, Hailie Mitchell, Saurabh Pujar, Luca Buratti, Shyam Ramji, Junfeng Yang, Gail Kaiser, and Baishakhi Ray. 2025. Understanding Automated Program Repair Agents Through the Lens of Traceability: An Empirical Study. doi:10.48550/ARXIV.2506.08311 Version Number: 2.

[3] Haoming Huang, Pongchai Jaisri, Shota Shimizu, Lingfeng Chen, Sota Nakashima, and Gema Rodríguez-Pérez. 2026. More Code, Less Reuse: Investigating Code Quality and Reviewer Sentiment towards AI-generated Pull Requests. doi:10.48550/ARXIV.2601.21276 Version Number: 1.

[4] ISO/IEC. 2011. Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models. 34 pages. Available at: https://www.iso.org/standard/35733.html.

[5] 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 Transactions on Software Engineering and Methodology* (3 2026). doi:10.1145/3796507

[6] Antonio Mastropaolo, Luca Pascarella, Emanuela Guglielmi, Matteo Ciniselli, Simone Scalabrino, Rocco Oliveto, and Gabriele Bavota. 2023. On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2149–2160. doi:10.1109/ICSE48619.2023.00181

[7] Dao Sy Duy Minh, Huynh Trung Kiet, Nguyen Lam Phu Quy, Pham Phu Hoa, Tran Chi Nguyen, Nguyen Dinh Ha Duong, and Truong Bao Tran. 2026. Early-Stage Prediction of Review Effort in AI-Generated Pull Requests. doi:10.48550/ARXIV.2601.00753 Version Number: 2.

[8] Hussein Mozannar, Valerie Chen, Mohammed Alsobay, Subhro Das, Sebastian Zhao, Dennis Wei, Manish Nagireddy, Prasanna Sattigeri, Ameet Talwalkar, and David Sontag. 2024. The RealHumanEval: Evaluating Large Language Models' Abilities to Support Programmers. arXiv:2404.02806 [cs.SE] https://arxiv.org/abs/2404.02806

[9] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2023. Do Users Write More Insecure Code with AI Assistants?. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA). ACM, 2785–2799. doi:10.1145/3576915.3623157

[10] Sadra Sabouri, Philipp Eibl, Xinyi Zhou, Morteza Ziyadi, Nenad Medvidovic, Lars Lindemann, and Souti Chattopadhyay. 2025. Trust Dynamics in AI-Assisted Development: Definitions, Factors, and Implications. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE, Ottawa, ON, Canada, 1678–1690. doi:10.1109/ICSE55347.2025.00199

[11] Trevor Stalnaker, Nathan Wintersgill, Oscar Chaparro, Laura A. Heymann, Massimiliano Di Penta, Daniel M. German, and Denys Poshyvanyk. 2026. Developer Perspectives on Licensing and Copyright Issues Arising from Generative AI for Software Development. *ACM Transactions on Software Engineering and Methodology* 35, 2 (Feb. 2026), 1–39. doi:10.1145/3743133

[12] 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. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 342–352. doi:10.1109/ICSE-SEIP66354.2025.00036

[13] Valerio Terragni, Annie Vella, Partha Roop, and Kelly Blincoe. 2025. The Future of AI-Driven Software Engineering. *ACM Transactions on Software Engineering and Methodology* 34 (5 2025). Issue 5. doi:10.1145/3715003

[14] Yanlin Wang, Wanjun Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2025. Agents in software engineering: survey, landscape, and vision. *Automated Software Engineering* 32 (11 2025), 70. Issue 2. doi:10.1007/s10515-025-00544-2

[15] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. Demystifying LLM-Based Software Engineering Agents. *Proceedings of the ACM on Software Engineering* 2 (6 2025), 801–824. Issue FSE. doi:10.1145/3715754