A 3-Layer Agentic Model for Nonfunctional Requirements in Software Engineering

Ehsan Zabardast
Software Engineering Research Lab SERL
Blekinge Institute of Technology
Karlskrona, Sweden
and
Gaetir
Karlskrona, Sweden
ehsan.zabardast@bth.se

Tiago Vieira
Independent Researcher
Stockholm, Sweden
tiagovr@gmail.com

Tony Gorschek
Software Engineering Research Lab SERL
Blekinge Institute of Technology
Karlskrona, Sweden
and
fortiss GmbH
Munich, Germany
tony.gorschek@bth.se

***Abstract*—Modern software-intensive systems must address a wide range of nonfunctional requirements (NFRs)—such as security, compliance, and maintainability—that are critical for the long-term success of the system. With the rise of large-language-model-based agents, software engineering is entering an "agentic" era where AI components are not only tools but collaborators in development processes. However, leveraging these agents introduces dual challenges: ensuring that AI components themselves meet quality standards (e.g., compliance, security, maintainability), and harnessing AI effectively to support system-level NFR assurance. Our perspective explicitly spans both SE4AI, where AI components such as agents are engineered and subjected to quality assurance and AI4SE, where AI agents support the engineering of software-intensive systems. While these are conceptually distinct, our model addresses both in a unified way. This position paper introduces a conceptual, domain-agnostic three-layer model—comprising Data, Agent, and Perspective layers—for systematically embedding AI agents into NFR assurance across the software lifecycle. The model explicitly captures two complementary viewpoints: Quality for AI (ensuring AI agents are trustworthy and maintainable) and AI for Quality (using agents to support system NFRs). Through illustrative examples in compliance, security, and maintainability, the paper demonstrates how this model can guide researchers and practitioners in designing agent-based approaches to software quality. We argue that this model not only clarifies the dual roles of AI in software engineering but also provides a foundation for responsible, scalable, and effective integration of AI into NFR assurance.**

***Index Terms*—AI Agents in Software Engineering, Nonfunctional Requirements (NFRs), Software Quality Assurance.**

I. Introduction

Modern software-intensive systems must satisfy numerous nonfunctional requirements (NFRs)—such as security, regulatory compliance, and maintainability—which ensure the system's quality attributes beyond basic functionality. Meeting these NFRs is critical for trustworthy and robust software, yet it remains challenging due to their often cross-cutting, evolving, and context-dependent nature [1]. At the same time, a paradigm shift is underway in software engineering with the rise of AI agents powered by large language models (LLMs) and similar technologies [2]. These generative AI agents have begun to revolutionize development processes, outperforming traditional techniques on many tasks [3]. In practice, we are witnessing a growing "symbiotic partnership" between human developers and AI, where AI assistants boost productivity and support decision-making [2]. This trend is blurring the line between tool and collaborator, introducing what some call an "agentic" approach to software engineering.

However, leveraging AI agents for software engineering also raises new quality concerns. For example, naive use of a single LLM-based agent can lead to issues like hallucinated outputs or integration failures [3], undermining reliability. Moreover, when AI components are embedded into software, engineers must address novel questions: Is the AI itself secure, compliant, and maintainable? How do we verify that AI-generated artifacts (code, plans, test cases, etc.) uphold the system's required quality standards [4]? These dual concerns highlight a perspective gap in current practice—we must ensure quality of the AI components and use AI to ensure the quality of the overall system.

In this position paper, we propose a conceptual, domain-agnostic 3-layer model for developing and applying AI agents to support NFRs throughout the software lifecycle. The model consists of a *Data Layer*, an *Agent Layer*, and a *Perspective Layer*. The key idea is to introduce an explicit perspective layer to capture the two complementary viewpoints on quality:

1) **Quality Aspect for AI (SE4AI)** – ensuring that AI agents themselves meet required qualities (e.g., `Compliance4AI`, `Security4AI`), and;
2) **AI for System Quality (AI4SE)** – leveraging AI agents to help achieve system NFRs (e.g., `AI4Compliance`, `AI4Security`).

By structuring agent-based solutions with these layers, we aim to integrate AI into NFR assurance in a systematic way. Throughout, we use compliance, security, and maintainability as running examples; however, the Data–Agent–Perspective (DAP) model is NFR-agnostic and readily extends to other

This research is supported through the KKS SERT Research Profile project (Ref. 2018010), Blekinge Institute of Technology.

qualities such as robustness, resilience, and Responsible AI concerns (e.g., fairness, transparency, accountability). This approach echoes broader shifts in software engineering: just as `AI4SE` and `SE4AI` are now seen as intertwined aspects of an AI-driven Software Engineering paradigm, our model treats AI as both subject and instrument of software quality.

We begin in Section II by outlining the challenges associated with nonfunctional requirements (NFRs) and the rise of LLM-based agent systems. Section III then introduces the proposed three-layer model illustrated through examples in compliance, security, and maintainability to highlight the dual perspectives of Quality for AI and AI for Quality. In Section IV, we discuss the broader implications of adopting this model for research and practice, including the development of agentic workflows. Finally, Section V presents directions for future work aimed at realizing AI-augmented NFR assurance in practice.

II. Background

To situate our proposed model within existing work, it is essential to review the dual challenges that motivate it: the difficulties of assuring nonfunctional requirements (NFRs) in software engineering, and the recent rise of LLM-based agents as collaborators in development processes. This background section therefore outlines the nature of NFRs and their associated quality challenges, traces the emergence of agentic AI systems in software engineering, and highlights the need for integrating these perspectives into a unified framework for NFR assurance.

A. Nonfunctional Requirements and Quality Challenges

Nonfunctional requirements define how a system should operate, covering qualities like performance, security, safety, reliability, compliance to regulations, usability, and maintainability. Ensuring these qualities is essential for user trust and system success [5]. Unlike functional requirements, NFRs are often difficult to specify quantitatively and are verified through extensive testing, reviews, or runtime monitoring. They are also notoriously easy to overlook or under-prioritize in fast-paced development [6]. In practice, achieving NFRs can be labor-intensive (e.g., manual security code reviews, compliance audits) and demands specialized expertise. With the introduction of AI and machine learning (ML) components into software, additional complexity arises—for example, how to test a non-deterministic AI component for safety or compliance, or how to maintain an evolving ML model over time. Traditional software engineering practices are being strained to address these new aspects of quality.

Nonfunctional requirements (NFRs) remain difficult to validate in practice, especially at scale and in industrial contexts. Recent empirical work highlights both the cost and scalability issues of validating NFRs. For example, Yu et al. [7] conducted a multi-case study across Nordic companies showing that while automation can increase the efficiency of NFR testing in continuous integration environments, it remains a challenging task to operationalize in practice. A recent systematic review of 84 empirical papers concludes that most evaluations are small-scale and that industry practice around quality requirements is often ad-hoc, with a persistent research–practice gap [8]. In large-scale distributed agile contexts, interview and case studies report that testing and validating quality requirements is particularly challenging due to multi-team coordination, long feedback loops, and unclear ownership [9]. Multi-company case research further documents recurring requirements-engineering challenges at scale, with limited support for NFRs in agile/DevOps workflows [10]. At the project level, surveys indicate that roughly one-third of NFRs are changed during development and many are defined late or remain non-measurable—factors that directly complicate validation and increase cost [11]. Architects also report that NFRs strongly shape architecture yet are validated with limited tooling and fragmented processes [12]. Collectively, this evidence motivates approaches that reduce the cost and scalability barriers of NFR validation, which is the focus of our model.

B. Rise of LLM-Based Agents in SE

Recent advances in AI, especially large language models, have unlocked a new paradigm of AI-based development assistants and agents [13]. LLM-based tools like GitHub Copilot and ChatGPT have shown remarkable capability in coding, testing, and other tasks, effectively acting as intelligent partners in the development process [2]. Unlike earlier AI tools, modern LLM agents can autonomously perceive their environment, use external tools or knowledge sources, and execute complex sequences of actions towards a goal [13]. This extends their versatility far beyond single-step code suggestions. For example, multi-agent systems of LLMs have been proposed to collaboratively handle complex software tasks, improving problem-solving robustness and scalability [14]. Such agents could be applied across the software lifecycle, from requirements analysis and code generation to testing and maintenance [13]. From an applied perspective, Yu et al. [15] reported on industrial experiences with deploying LLM-based applications for information retrieval from proprietary enterprise data. Their findings highlight both the promise and the challenges of using LLMs in real-world software engineering settings, particularly around issues of data sensitivity, retrieval accuracy, and integration with existing processes. These insights complement our focus by grounding the opportunities and risks of agentic systems in empirical industrial cases.

Industry has begun adopting AI-driven solutions for software quality assurance, for example, AI-powered code analysis platforms now aim to automatically enforce coding standards and detect security or compliance issues in codebases. Empirical studies are still needed to demonstrate that generative AI assistants can measurably improve developer productivity and software quality, for example through automated bug fixes and test generation.

C. Integrating AI with NFR Assurance

While AI agents show promise in assisting with quality-related tasks (such as detecting vulnerabilities or optimizing performance), their integration raises two sides of a quality coin. On one side, we have quality for AI systems: the AI components themselves must be trustworthy, secure, and compliant before we can safely rely on them. On the other side, we face AI for software quality: AI agents can help monitor and improve attributes like reliability, performance, or compliance of the system. For example, an agent that scans requirements and code for regulatory compliance issues, or one that refactors code to improve maintainability.

Recent discussions highlight that the relationship between AI and software engineering is bidirectional—we must adapt engineering practices to incorporate AI (sometimes termed `SE4AI`), and simultaneously leverage AI to transform engineering practices (`AI4SE`). In the context of NFRs, this means treating quality assurance of AI and quality assurance by AI as equally important considerations. For example, deploying an AI agent for security testing not only requires training it to find vulnerabilities (`AI4Security`), but also ensuring the agent's recommendations do not introduce new risks and that the agent itself cannot be subverted (`Security4AI`).

Recent research has also investigated how the quality of generative AI systems themselves can be evaluated. Yu et al. [16] conducted a snowballing literature review and identified 28 metrics for assessing LLM outputs, mapping them to ISO/IEC 25023 quality characteristics such as usability, reliability, functional suitability, and maintainability. This work provides a bridge between traditional quality frameworks and the emerging domain of generative AI, and directly motivates our consideration of how NFRs can be addressed through measurable quality perspectives in our proposed model.

Finally, this duality is often missing in ad-hoc AI adoption. A lack of structured approach can lead to pitfalls, e.g., an LLM agent might produce an apparently optimized configuration that violates compliance policies, or it might hallucinate inaccurate results that mislead engineers [3]. To avoid such issues, there is a need for frameworks that explicitly embed quality considerations into the design and operation of AI agents. Researchers have begun moving in this direction: efforts are underway to define taxonomies and platforms for "AI agent communities" with governance, monitoring, and quality assessment mechanisms [3]. These emphasize that beyond raw functionality, aspects like output quality, resource usage, reliability, and ethics must be actively managed in agent-based systems. Building on these insights, we propose a three-layer model that makes the quality perspectives first-class elements of any AI agent deployment for software engineering.

III. The Data–Agent–Perspective (DAP) Model

Our proposed model for AI-agent-driven NFR support consists of three layers (see Figure 1): (1) Data, (2) Agent, (3) Perspective. This conceptual model is domain-agnostic, meaning it can be applied to any software domain (finance, healthcare, etc.) by instantiating domain-specific data and quality criteria. The layers separate concerns of information, intelligence, and intent, respectively:

A. Data Layer

This bottom layer encompasses the information sources and environment that the AI agent operates on. It includes all relevant data required to evaluate and improve NFRs. Examples of data inputs are: software artifacts (requirements documents, design models, source code, test cases), operational data (log files, performance metrics, usage analytics), knowledge bases (compliance checklists, security vulnerability databases, coding standards), domain-specific regulations or policies, and memories generated by AI Agents. By structuring the relevant data here, we provide the agent with the necessary context and facts to reason about quality. As an example, for a compliance scenario, the data layer might include regulatory guidelines or privacy policies that the system must adhere to. For security, it could contain known vulnerability patterns or past incident reports. Importantly, the data layer may also include feedback loops from the running system—e.g., monitoring alerts or user feedback—allowing agents to perceive the system's quality status in real time. In summary, this layer supplies the raw materials (both static and dynamic data) that inform the agent's analysis and actions on NFRs.

An important concern at the Data Layer is the handling of intellectual property rights (IPR) and sensitive data. When organizations rely on external, cloud-hosted LLMs for agentic support, there is often limited control over how engineering data flows beyond organizational boundaries. This creates a trade-off: external models may provide superior, up-to-date performance at lower cost, but they raise risks of data exposure and vendor lock-in; local models, while offering greater control, are resource-intensive and may lag behind in capability. This tension highlights a critical need for new trust and assurance mechanisms between model vendors and users, ensuring that `Security4AI` and `Compliance4AI` concerns are met while retaining the benefits of advanced external agents.

1) Data Management and Retrieval: Beyond enumerating sources, the Data layer must operationalize *how* information is prepared and delivered to agents. In practice this requires: (i) **ingestion and indexing** (schema/metadata design; semantically coherent chunking of artifacts; redaction of PII/secret material; versioning and incremental re-indexing); (ii) **retrieval** (hybrid lexical+vector search, reranking, and query expansion to maximize recall of relevant artifacts); (iii) **memory organization** (short-term "working" context for the task, longer-term episodic logs of agent–system interactions, and durable semantic memories distilled from past runs); and (iv) **governance** (access control, audit trails, and data retention policies aligned with `Compliance4AI` and `Security4AI` perspectives).

B. Agent Layer

The middle layer is the AI agent (or agents) itself, which embodies the reasoning and actuation capabilities. The agent in our model is typically an LLM-based autonomous agent, potentially composed of multiple specialized sub-agents working in concert [3], [13]. It can interpret the software artifacts and

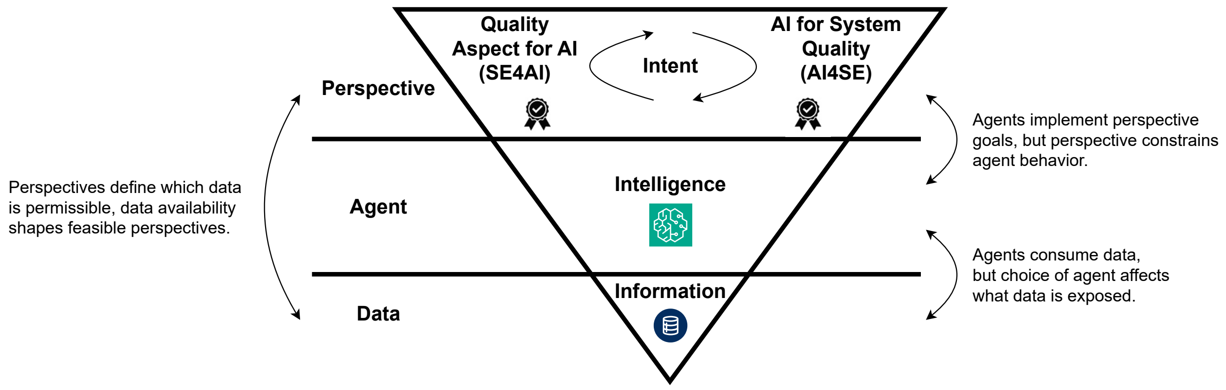
Quality Aspect for AI (SE4AI)
Intent
AI for System Quality (AI4SE)
Perspective
Agent
Data
Intelligence
Information
Perspectives define which data is permissible, data availability shapes feasible perspectives.
Agents implement perspective goals, but perspective constrains agent behavior.
Agents consume data, but choice of agent affects what data is exposed.

Fig. 1. The Data–Agent–Perspective model for NFRs. In addition to the hierarchical structure, the arrows highlight mutual dependencies across layers: data availability and governance constrain both agent capabilities and perspective goals; agents influence what data is generated or consumed; and perspectives regulate how data and agents may be used. This interdependence reflects the reality that AI4SE and SE4AI considerations must be coordinated across all layers.

context, make decisions or recommendations, and interact with tools or humans. Key components of the agent include capabilities for planning (breaking down complex quality tasks, e.g., multi-step security analysis), memory (maintaining context about the system or past findings), perception (interpreting data inputs, such as parsing code or requirements text), and action (executing changes or producing reports/tests) [13]. For example, an agent could autonomously perform a compliance audit by reading requirement specifications, checking code comments for required legal disclaimers, querying a policy database, and then generating a compliance report or even opening issues for non-compliant code sections. Another agent might focus on maintainability by analyzing code complexity and suggesting refactoring, or by autonomously generating documentation for poorly understood modules. The agent layer can also leverage external tools—e.g., calling a static analysis tool for deeper code inspection or using a search engine to retrieve the latest regulatory updates—thereby extending the agent's knowledge beyond the base LLM [13]. If multiple agents are used, they could be orchestrated to handle different NFRs or different aspects of a complex goal, communicating among themselves. Regardless of internal structure, the agent layer is responsible for bridging between raw data and high-level quality goals: it interprets data layer inputs through the lens of the perspective layer's objectives, and produces outputs (decisions, artifacts, or direct system modifications) aimed at improving or validating NFR compliance.

While the Agent Layer holds promise, it also faces practical limitations. A key challenge is scalability: real-world engineering artifacts are often vast—thousands of pages of documentation or millions of lines of code. Current LLM-based agents struggle with such scale, typically requiring chunking strategies (splitting data into smaller segments). However, chunking introduces its own challenges: ensuring continuity of reasoning across segments, avoiding loss of global context, and preventing contradictions between partial analyses. Early work, e.g., [13], [14], report both promise and persistent difficulties with these approaches. Addressing such scalability issues is therefore a priority for future work, where advances in retrieval-augmented agents, hierarchical reasoning, or hybrid human-in-the-loop approaches may provide more robust solutions.

C. Perspective Layer

The top layer encodes the quality objectives, constraints, and viewpoints that guide the agent's behavior. This is where our model explicitly distinguishes between the two critical perspectives: **Quality Aspect for AI**, ensuring the agent itself and any AI components in the system meet certain qualities and **AI for System Quality**, using the agent to ensure the system meets its NFRs.

1) Quality Aspect for AI (SE4AI perspective): This viewpoint treats the AI agent as an object of quality assurance. It includes constraints and requirements to which the agent (and underlying ML, deep learning models or data) must adhere. For example, `Compliance4AI` means the agent must operate in compliance with relevant standards—if handling user data, it should follow privacy laws (no unauthorized use of personal data) and organizational policies. Compliance, as discussed here, is only one illustrative perspective. However, it is important to note that being compliant does not necessarily mean being secure, ethical, or beneficial for the user organization. Compliance depends on what standard or regulation is applied: one standard may address IPR protection, while another may govern privacy, safety, or accessibility of the final system. Thus, a compliant AI may still pose risks if the compliance scope is narrow or misaligned with stakeholder concerns. Beyond compliance, organizations may equally emphasize perspectives such as `Security4AI`, `Privacy4AI`, `Maintainability4AI`, or `Ethics4AI`, depending on their context. Our model therefore encourages defining multiple perspectives to reflect the diverse and sometimes conflicting priorities of stakeholders.

`Security4AI` means the agent should be secure in its operation—for example, robust against prompt injections or model manipulation, not exposing sensitive system information, and maintaining confidentiality/integrity of the data layer inputs. `Maintainability4AI` might involve ensuring the agent's knowledge or prompts are up-to-date and that the AI components can be efficiently retrained or updated as standards evolve. In practice, this layer might impose rules like "the agent must explain its recommendations" (an interpretability requirement) or "the agent should not execute code in production without human approval" (a safety/oversight requirement). By codifying *Quality4AI* concerns, the perspective layer guards against the introduction of new risks by the very tools meant to help – aligning with emerging principles of responsible and trustworthy AI in software engineering [14].

2) AI for System Quality (AI4SE perspective): This viewpoint positions the AI agent as an instrument to achieve system NFRs. It defines the specific quality objectives the agent is tasked to monitor or improve in the target system. For example, `AI4Compliance` means the agent's goal is to ensure the software system complies with requirements or regulations—it could proactively check that requirements specifications include necessary legal clauses [2], or verify that code meets accessibility standards. `AI4Security` would focus the agent on identifying security weaknesses, such as scanning code for common vulnerabilities or analyzing runtime logs for intrusion patterns. `AI4Maintainability` might have the agent track code quality metrics (e.g., cyclomatic complexity, code smells) and suggest or even apply refactorings to keep the design clean and adaptable. The perspective layer provides the acceptance criteria or definitions of done for these tasks—for example, what counts as a security vulnerability, or which compliance rules must be satisfied—possibly linking to explicit artifacts in the data layer (e.g., a regulatory checklist). In essence, this part of the perspective layer translates stakeholder quality requirements into an agent mission.

Crucially, these two sub-perspectives are not isolated; they interplay. If an organization deploys an AI agent to enforce compliance (`AI4Compliance`), it must also ensure the agent's operation is itself compliant (`Compliance4AI`)—otherwise, the effort is self-defeating. Our model's Perspective layer encourages engineers to define both aspects in tandem. In practical terms, this could mean that for every NFR-focused agent introduced, one must articulate any new risks that agent brings and how to mitigate them. The agent can even be designed to self-monitor certain aspects of its behavior. For example, a security agent could have a secondary routine to detect if it has been given malicious instructions, thereby self-applying `Security4AI` principles.

Although we introduced these perspectives at the top layer of the model, it is important to recognize that `AI4SE` and `SE4AI` considerations permeate all layers. For example, decisions about what data to expose (Data Layer), or how to orchestrate multi-agent reasoning (Agent Layer), directly shape the feasibility and trustworthiness of Perspective Layer objectives. This interdependence underscores the need for holistic assurance across layers rather than treating them in isolation.

By separating the Perspective layer, we achieve a clear conceptual model: The Data layer provides facts, the Agent layer provides the reasoning and automation, and the Perspective layer provides the goals and guardrails. This separation is domain-agnostic—whether the system is a medical device with strict safety requirements or a fintech application with heavy compliance needs, the model applies by plugging in the appropriate data sources and perspective rules. For compliance, the data layer might include laws/regulations and system audit logs; the agent might perform continuous compliance checks; the perspective layer would encode both `Compliance4AI` (e.g., the agent respects privacy constraints) and `AI4Compliance` (e.g., the agent's goal is to enforce GDPR requirements in system features). Similar mappings can be drawn for security (with threat databases and security scanners as data, a penetration-testing agent, `Security4AI` constraints on the agent itself, and `AI4Security` objectives for the system) and for maintainability (with code metrics and documentation as data, a refactoring agent, rules to keep the agent updated with coding guidelines, and goals to maintain code simplicity and consistency).

IV. Discussion

Adopting an agentic approach to NFRs raises numerous research questions. First, the effectiveness of AI agents in fulfilling system quality tasks needs thorough evaluation. Early studies and tools suggest improvements in productivity and defect detection, but ensuring that AI-generated solutions truly meet NFR standards (e.g., performance budgets or security hardening) remains an open challenge. Validating that an AI's output satisfies a given nonfunctional requirement is non-trivial [4]. Researchers must devise methods to formally verify or systematically test NFR compliance of AI-generated artifacts [4]. This may involve developing new benchmarks and quality metrics for AI in roles like "compliance auditor" or "risk assessor".

Second, there is a rich space for research into agent collaboration and specialization for NFRs: perhaps a team of diverse agents (one focusing on performance tuning, another on security analysis, etc.) can collaborate to balance trade-offs between competing NFRs (e.g., performance vs. security). How these agents communicate and resolve conflicts (with or without human oversight) is an open question. Early visions of Software Engineering 2.0 imagine autonomous, scalable, and trustworthy agent communities tackling such complex trade-offs [14].

Another research implication is the reconceptualization of software engineering knowledge areas in light of AI integration. Our position aligns with the notion that `AI4SE` and `SE4AI` are not separate silos but a fundamental transformation of Software Engineering. This means research should not treat NFR-supporting AI as a niche; instead, every sub-discipline (requirements, design, testing, maintenance, etc.)

should evolve to incorporate agentic assistance and constraints. For example, requirements engineering research might explore new ways to elicit and represent NFRs that are directly consumable by AI agents (making the perspective layer machine-interpretable). Architecture research could investigate reference architectures that embed our 3-layer model – possibly adding a coordination layer for multiple agents or a feedback layer for continuous learning from production data. The fast pace of GenAI advancement offers fertile ground to re-imagine classical SE problems with a fresh, AI-augmented perspective.

A further implication of adopting agentic approaches is the competitive advantage that suppliers of LLMs and agent platforms may derive from collected usage and engineering data. For example, companies specializing in security-oriented AI agents may continuously improve their models using data obtained from customers' development workflows. Similarly, integration of AI assistants into environments such as Visual Studio raises questions about how solution and engineering data are harvested and leveraged by providers. These dynamics could influence not only the evolution of AI-enabled software engineering tools, but also issues of vendor lock-in, data governance, and fairness in access to advanced capabilities. Addressing such implications will be essential for ensuring that agentic workflows support long-term trust and equitable value distribution.

For software practitioners, applying AI agents to NFRs promises benefits but also demands careful changes in workflow ("agentic workflows"). On the positive side, AI agents can automate tedious quality assurance tasks and provide expertise on demand. Teams might employ a "compliance agent" to continuously review code and configs for regulatory violations, or a "maintenance bot" to create refactoring pull requests. This has the potential to catch issues early and reduce technical debt, ultimately improving productivity and product quality.

Yet, to realize these benefits, organizations must instill new practices and competencies. Developers will need to learn how to effectively collaborate with AI agents – treating them not just as static tools, but as semi-autonomous teammates. This involves skills like prompt engineering, interpreting AI explanations, and supervising AI decisions. Indeed, using AI as a "co-worker" transforms how engineers do their job, requiring understanding of human-AI teaming, oversight, and legal/accountability aspects.

V. Conclusion and Future Work

We presented a position on integrating AI agents into the assurance of nonfunctional requirements via a three-layer model capturing Data, Agent, and Perspective considerations. This model highlights the necessity of addressing quality in two directions: using AI to achieve system qualities and ensuring the AI itself meets quality standards. Our examples in compliance, security, and maintainability illustrate how an agentic approach can be operationalized while keeping these dual aspects in focus.

While we use compliance, security, and maintainability as running examples, the Data–Agent–Perspective (DAP) model is NFR-agnostic. We leave systematic instantiations of these qualities as future work, but the mechanism is identical. The proposed model is deliberately conceptual and domain-agnostic, intended as a thinking tool for researchers and practitioners to design AI enhancements to their quality processes without losing sight of governance and context.

Moving forward, we envision several avenues of work to refine and validate this model. On the research side, prototyping the model in specific domains (e.g., an AI agent for privacy compliance in healthcare software) would provide concrete insights – such case studies can evaluate the efficacy of the Data–Agent–Perspective separation and identify any gaps. Additionally, developing standardized schemas for Perspective layer elements (such as machine-readable compliance rules or security policies for agents) could facilitate broader adoption and tool interoperability.

In conclusion, AI agents offer a promising new arsenal for tackling the perpetual challenges of nonfunctional requirements in software engineering. Our position is that a structured model, accounting for both the AI's and the system's qualities, is essential to harness this promise responsibly.

References

[1] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Springer Science & Business Media, 2012, vol. 5.

[2] V. Terragni, A. Vella, P. Roop, and K. Blincoe, "The future of ai-driven software engineering," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–20, 2025.

[3] C. Di Sipio, M. C. S. De Oliveira, D. Di Ruscio, P. T. Nguyen, and R. Rubei, "Agentware in software engineering: A taxonomy for leveraging llms-based multi-agent systems," *Available at SSRN 5273078*.

[4] A. Nguyen-Duc, B. Cabrero-Daniel, A. Przybylek, C. Arora, D. Khanna, T. Herda, U. Rafiq, J. Melegati, E. Guerra, K.-K. Kemell *et al.*, "Generative artificial intelligence for software engineering—a research agenda," *Software: Practice and Experience*, 2025.

[5] H. Washizaki, Ed., *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide)*, version 4.0 ed. IEEE Computer Society, 2024. [Online]. Available: https://www.swebok.org

[6] J. Horkoff, "Non-functional requirements for machine learning: Challenges and new directions," in *2019 IEEE 27th international requirements engineering conference (RE)*. IEEE, 2019, pp. 386–391.

[7] L. Yu, E. Alégroth, P. Chatzipetrou, and T. Gorschek, "Automated nfr testing in continuous integration environments: a multi-case study of nordic companies," *Empirical Software Engineering*, vol. 28, no. 6, p. 144, 2023.

[8] T. Olsson, S. Sentilles, and E. Papatheocharous, "A systematic literature review of empirical research on quality requirements," *Requirements Engineering*, vol. 27, no. 2, pp. 249–271, 2022.

[9] W. Alsaqaf, M. Daneva, and R. Wieringa, "Quality requirements challenges in the context of large-scale distributed agile: An empirical study," *Information and software technology*, vol. 110, pp. 39–55, 2019.

[10] R. Kasauli, E. Knauss, J. Horkoff, G. Liebel, and F. G. de Oliveira Neto, "Requirements engineering challenges and practices in large-scale agile system development," *Journal of Systems and Software*, vol. 172, p. 110851, 2021.

[11] L. Viviani, E. Guerra, J. Melegati, and X. Wang, "An empirical study about the instability and uncertainty of non-functional requirements," in *International Conference on Agile Software Development*. Springer Nature Switzerland Cham, 2023, pp. 77–93.

[12] D. Ameller, C. Ayala, J. Cabot, and X. Franch, "How do software architects consider non-functional requirements: An exploratory study," in *2012 20th IEEE international requirements engineering conference (RE)*. IEEE, 2012, pp. 41–50.

[13] J. Liu, K. Wang, Y. Chen, X. Peng, Z. Chen, L. Zhang, and Y. Lou, "Large language model-based agents for software engineering: A survey," *arXiv preprint arXiv:2409.02977*, 2024.

[14] J. He, C. Treude, and D. Lo, “Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead,” *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–30, 2025.

[15] L. Yu, E. Alégroth, P. Chatzipetrou, and T. Gorschek, “Experience with large language model applications for information retrieval from enterprise proprietary data,” in *International Conference on Product-Focused Software Process Improvement*. Springer, 2024, pp. 92–107.

[16] ——, “Measuring the quality of generative ai systems: Mapping metrics to quality characteristics-snowballing literature review,” *Information and Software Technology*, p. 107802, 2025.