Specifying the Delegated-Autonomy Boundary: Requirements Engineering for Agentic AI

Chetan Arora
Faculty of IT, Monash University
Melbourne, Victoria, Australia
chetan.arora@monash.edu

Andreas Vogelsang
University of Duisburg-Essen
Essen, Germany
andreas.vogelsang@uni-due.de

Abbi Sharma
Faculty of IT, Monash University
Melbourne, Victoria, Australia
abbi.sharma@monash.edu

Abstract

Agentic AI systems do not just predict or recommend; they plan, maintain state, and act in external environments with varying degrees of autonomy. This changes the requirements engineering problem in a specific and under-addressed way: it introduces what we call the *delegated-autonomy boundary*—the set of decisions about what may be delegated to the system, under what graduated authority, with what oversight, and how control is returned. Current practices bury these decisions inside prompts, tool schemas, and runtime policies, even though they are requirements-level commitments. This paper proposes two complementary artifacts. First, an *Agency Justification Record* (AJR) helps teams decide *when* an agent is warranted over simpler alternatives. Second, an *Agentic Delegation Policy* (ADP) captures *what* must be specified for safe and effective development: purpose, authority, information, coordination, assurance, and evolution. Crucially, authority in the ADP is modelled as *graduated*, i.e., a tiered structure. We illustrate the framework with two contrasting examples: a safety-critical hospital discharge coordination agent and an automated code review agent.

Keywords

Requirements Engineering, Agentic AI, AI agents, Delegated Autonomy, Software Engineering.

1 Introduction

For most of software engineering (SE) history, specifying a system meant specifying its *behaviour*: given this input, produce this output, subject to these constraints. The rise of AI techniques—machine learning, deep learning, and large language models (LLMs)— has expanded RE to encompass data quality, model validation, fairness, and uncertainty [2, 6, 24]. Yet the underlying assumption held: the system *responded*; it did not *act* [7]. Agentic AI breaks this assumption entirely. An agent invokes tools, modifies external state, retains context across steps, and pursues goals through *iterative, self-directed* sequences of autonomous decisions [3, 15]. We scope this paper to such *high-autonomy, multi-step* agentic systems (with cross-system action) and distinguish them from single-step LLM inference embedded in a pipeline, which some practitioners also label "agents" but that introduces a different, narrower RE problem. Specifying an *agentic* system requires deciding not only what it should produce but *also what it may do*, *to what resources*, *under whose authority*, and *when control returns to a human*. We call the specification of these decisions the *delegated-autonomy boundary* and argue that it is the central missing object in current RE practice.

This gap arises for two recurring reasons. The first is *unjustified agency*: a system is built as an agent, even though a prompted model, deterministic workflow, or conventional software design would suffice. The second is *under-specified agency*: once an agentic architecture is chosen, the system's authority, memory scope, coordination rules, escalation conditions, and success criteria remain implicit—buried in prompts, tool schemas, or orchestration scripts. Together, these failure modes reveal that RE for agentic AI must address both *when* to use agents and *what* must be specified.

The urgency is real. Agents are already operating in healthcare coordination, software development, legal document processing, and enterprise automation [21, 26]. While robust frameworks for NL specification are emerging, overreliance on ad hoc prompt tinkering in high-risk operational contexts poses long-term governability challenges [9, 19, 20]. For safety-critical or heavily regulated deployments, unstructured engineering compromises auditability and makes regulatory compliance a moving target. The window for establishing good practice is open now, during the formative period of agentic deployment, and our paper presents a vision in that direction. The AJR and ADP structures were developed in consultation with two senior practitioners: one with experience in agentic enterprise systems, and one in healthcare, each with at least more than ten years of industry experience. Their feedback shaped both the criteria in Section 2 and the authority-tier structure in Section 3.

Contributions. We make the following contributions in this paper.

(1) **A new RE problem formulation.** The *delegated-autonomy boundary* as a first-class RE concern, distinct from prior work on adaptive systems, goal modelling, or AI-specific RE.
(2) **The Agency Justification Record (AJR).** A lightweight artifact that disciplines teams to justify agentic architectures against simpler alternatives, operationalising the principle of *minimal justified autonomy*.
(3) **The Agentic Delegation Policy (ADP).** A structured specification artifact covering purpose, graduated authority, information and memory, coordination, assurance, and evolution.
(4) **A research agenda for RE for agentic AI.** Four open problems—notations, requirements-to-runtime compilation, requirements-driven evaluation, and accountability—framed as tractable empirical and design-science questions.

We do not claim that delegated autonomy exhausts the RE problem for agentic AI. Stakeholder goals, usability, ethics, and regulatory compliance remain essential and orthogonal concerns. In fact, ADP is defined to complement the software requirements specification (SRS). SRS captures stakeholder goals, functional requirements, and quality attributes; the ADP specialises in authority, memory, and coordination dimensions unique to agentic delegation. Hence, our claim is narrower: agentic systems introduce a *distinct* RE problem regarding the scope, graduation, and governance of delegated action that must now be addressed.

Table 1: AJR record with running examples.

Criterion	Discharge coord. agent	Code review agent
C1: Task structure	Cross-system, exception-heavy; steps and order cannot be predefined ✓	Largely enumerable; well-defined linting and diff parsing ✗
C2: Unstructured context	Clinical notes, payer letters, ambiguous histories require NL reasoning ✓	Diffs are structured; rules are explicit; ambiguity is low ✗
C3: Action surface	Orchestrator delegates actions across EHR, and payer systems to specialist sub-agents; each sub-agent operates in a scoped boundary ✓	Source repository; read-only diff access only ✗
C4: Evaluable progress	Medication reconciliation status, clinician sign-off, discharge completeness are observable ✓	Comment accuracy & scope violations are observable ✓
C5: Bounded risk	Clinician approval; no finalisation without sign-off; HIPAA audit log ✓	Developer check before posting; actions reversible ✓
C6: Advantage over baseline	Discharge delay and medication reconciliation error rate vs clinical audit baseline; threshold: ≥20% reduction over 90 patient episodes ✓	Comment turnaround vs existing pipeline >90 days; threshold but little projected gain to justify agentic overhead ✗
Verdict	**Agentic approach justified**	**Rejected: C1–C3 + C6 fail; prompted model**

Running examples. We ground the paper in two contrasting examples. The first is a hospital discharge coordination system: a multi-agent system designed to manage patient preparation for discharge from an acute care ward. A coordinating orchestrator decomposes the task across three specialist sub-agents: a medication agent that reads and reconciles data from an Electronic Health Record (EHR) and pharmacy records; a payer and scheduling agent that interfaces with insurance authorisation portals and outpatient booking systems; and a discharge agent that drafts patient-facing discharge instructions. The orchestrator detects unresolved conflicts between sub-agent outputs, triggers escalation, and hands off to clinical staff. The system operates in a regulated environment (e.g., HIPAA [1]), involves clinicians and patients as primary stakeholders, and carries direct patient-safety consequences if any sub-agent acts beyond its sanctioned scope. The second is an automated code review agent: a system proposed to analyse pull requests (PRs) in a software repository, apply configurable style rules, and post inline comments for developers. It operates in an internal development environment and carries no direct safety consequences.

2 When to Use Agents: the AJR

Not every AI-enabled feature should be implemented as an agent. RE has long recognised the principle of *minimal adequate specification*: a system should not be given more complexity or autonomy than its requirements demand [23]. Applied to agentic AI, this becomes a gate: before committing to an agentic architecture, a team must justify why simpler alternatives are insufficient. We operationalise this gate as the *Agency Justification Record* (AJR)—completed before architecture is chosen and retained as a living design rationale document, analogous to an Architecture Decision Record [12] but focused on the decision to delegate autonomy. Practitioner guidance from both OpenAI and Anthropic independently reaches the same conclusion: start with the simplest solution and reserve agents for genuinely open-ended tasks [3, 15]. The AJR turns this intuition into an explicit, reviewable RE commitment.

2.1 Decision criteria

A task is a strong candidate for an agent when it meets most of the following criteria.

(1) **Open-ended task structure.** The number or order of steps cannot be reliably predefined; the agent must interpret context, revise plans, or recover from unforeseen exceptions.
(2) **Unstructured context.** Success depends on reasoning over NL, semi-structured artifacts, or ambiguous observations that make rule-based approaches brittle. However, a team-level decision might be necessary to allow an agentic approach if the context is deemed neither entirely "unstructured" nor semi-structured.
(3) **Cross-system actions.** The task must inspect, update, or orchestrate external systems rather than only generate content.
(4) **Evaluable progress.** There are explicit success criteria or intermediate checks that allow the agent or a supervising human to detect poor trajectories before harm occurs.
(5) **Bounded risk.** Harm can be constrained through sandboxes, reversible actions, rate limits, approval gates, or human takeover, and the team can specify these constraints before deployment.
(6) **Advantage over baselines.** The team can specify, in advance, a measurable threshold at which the added complexity of an agent is justified and a plan to verify. For example: *"the agent must reduce discharge delays by at least 20% over the current manual process, over 90 episodes"*. Without such a pre-committed threshold, the decision to use an agent cannot be falsified.

Criterion 6 cannot be verified before the agent is built, but AJR requires the team to state *how* the advantage will be measured and *what threshold* would justify the complexity. This forward commitment prevents post-hoc rationalisation. The AJR must also document *anti-criteria* that disqualify an agentic architecture regardless of how many positive criteria hold: a fixed workflow already performs adequately; outcomes cannot be verified; actions are irreversible or high-stakes; or multiple agents are proposed for conceptual neatness rather than demonstrated gains.

AJR examples. Table 1 shows AJR records for both running examples. The discharge agent clears all six and proceeds to ADP specification; the code review scenario fails at criteria 1–3 + 6, and the AJR rejects the agent in favour of a simpler prompted-model pipeline. The rejection path is as important as the acceptance path: a rejected AJR records the rationale for a simpler architecture, preventing the system from accruing unjustified agentic complexity. We note that the rejection path does not mean "no AI": a prompted-model pipeline suffices and remains within the scope of conventional RE for AI, even though some practitioners label such pipelines "agents", they fall outside the multi-step, cross-system planning that warrants the delegated-autonomy machinery of an ADP.

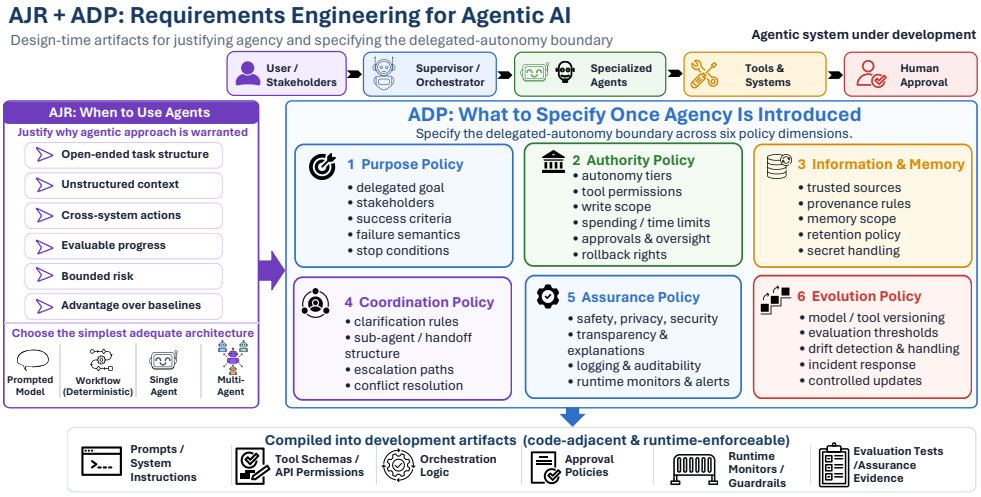

Figure 1: RE for Agentic AI Overview

3 What Must Be Specified: the ADP

If AJR justifies an agent, we extend conventional RE artifacts with an *Agentic Delegation Policy* (ADP). We favour *"policy"* over *contract* because requirements alone cannot enforce mutual obligations; a policy defines delegation terms while keeping accountability an organisational concern. The ADP is distributed across prompts, schemas, policy engines, and monitors (Figure 1 & Table 2). Although populated in Table 2 for illustration in second example, a rejected AJR would not, in practice, proceed to an ADP. Crucially, ADP adheres to the 'minimal adequate specification' principle by scaling with risk: safety-critical domains (e.g., healthcare) demand rigorous assurance, whereas low-risk settings (e.g., developer tooling) require only basic operational boundaries.

1. Purpose policy. The specification must define the delegated goal in operational terms: intended stakeholder(s), task scope, required outputs, success criteria, and failure semantics. Critically, it must include explicit *stop conditions*: max. attempts, uncertainty thresholds, deadlines, and conditions that force the system to relinquish control. In agentic systems, "keep trying" is itself a requirements decision, and its omission is a common source of runaway behaviour.

2. Authority policy and graduated autonomy. ADP further must specify *what the agent is permitted to do* and, critically, *at what level of oversight*. We model it in three-tiers that align with OpenAI's guidance that agent risk depends on write access, irreversibility, permissions, and monetary impact [15].

(A) **Autonomous tier.** Actions the agent may execute without human review, e.g., read-only ops and low-cost computations.
(B) **Advisory tier.** Actions the agent may propose but not execute until a human approves, e.g., orders with discrepancies flagged.
(C) **Prohibited tier.** Actions the agent may never take regardless of instructions, e.g., irreversible high-stakes modifications, and actions outside the delegated domain.

3. Information and memory policy. Agents mix retrieved context, intermediate notes, and persistent memory, so the teams must specify trusted sources, source precedence, retention duration, deletion rules, and confidential information handling. Session memory must be distinguished from durable memory, with requirements stating what may be remembered, for whom, and for how long. In first example, the agent is forbidden from retaining patient-identifiable information beyond the episode of care unless authorised. In second example, the agent may retain anonymised patterns to improve suggestions but not developer-identifiable commit histories.

4. Coordination policy. Requirements must define (often based on business processes) when an agent must ask clarifying questions, hand off execution, resolve contradictory outputs, or escalate to a human. This is vital for multi-agent designs, where decomposition introduces coordination overhead and novel failure modes [3, 26]. In the discharge system, the orchestrator must reconcile conflicting outputs from the medication and payer sub-agents; the coordination policy mandates that such conflicts be escalated to a clinician rather than resolved autonomously, and forbids sub-agents from initiating out-of-scope inter-agent communication. Furthermore, the coordination policy must govern dynamic, emergent tool-chaining. Rather than restricting individual tools, it forbids conflicting tool combinations and enforces mandatory human or deterministic verification gates before downstream, state-changing actions.

5. Assurance policy. Trustworthy-agent guidance emphasises meaningful human control, security, transparency, and privacy [4, 10]. NIST frames these as validity, safety, security, explainability, and accountability [13, 14]. For agentic systems, these concerns must be mapped to monitors, logs and explanation duties, including prompt-injection resilience for tool-using agents [16]. In practice, however the assurance remains largely implicit, as 25/30 deployed agents disclose no internal safety evaluations and 23/30 have no third-party testing [19]. The ADP assurance policy makes this explicit by enumerating runtime checks, red-team scenarios, and audit obligations that operationalise the assurance claims. Also, it requires that these be stated before deployment rather than after an incident.

6. Evolution policy. An agentic system is not stable if its model, tools, prompts, or policies change underneath the specification. The ADP must define versioning rules, acceptance thresholds, regression tests, rollback strategies, and incident response. Part of the requirements is specifying *how the requirements remain valid over time*. In both examples, model upgrades require regression across tier boundaries and escalation conditions, not just accuracy metrics.

Table 2: Illustrative ADP specification across all six dimensions for both running examples.

Dimension	Tier	Discharge coordination agent	Code review agent
Purpose	Success	All medication discrepancies resolved or escalated; discharge complete; clinician signed off.	Draft comments generated for ≥90% of flagged lines; review turnaround reduced vs. baseline
	Stop	Missing chart data unresolved in ≥ 2h; system unavailable; clinician override.	PR closed or merged; developer dismisses all suggestions.
Authority	Auto.	Read patient chart; compare medication lists; retrieve scheduling.	Read diffs; run static analysis; retrieve style guide.
	Advise.	Medicinal sub-agent flags discrepancy to orchestrator, which drafts consolidated discharge instructions for clinician;	Post draft inline comments pending developer acknowledgment
	Prohib.	Sign discharge orders; transmit instructions to patient; modify standing medication orders	Merge pull requests; modify protected branches; alter CI/CD configuration
Memory	Session	Full episode context including chart, notes; deleted at episode close	Diff context for current PR; cleared after merge or close
	Durable	Anonymised discharge-pathway patterns (no PII); authorised and logged	Anonymised code-pattern library across repositories; no developer-identifiable commit history
Coord.	Escalate	Orchestrator escalates to clinician with unresolved conflicts after N attempts;	
	Handoff	Orchestrator transfers context to on-call clinician if time limit exceeded;	No sub-agent handoff; single-agent pipeline only
Assurance	Monitor	HIPAA audit log on every data access; escalation trigger logged with evidence bundle	All comment drafts logged with line reference and rule ID; developer action (accept/dismiss) recorded
	Red-team	Prompt injection via adversarial discharge notes; privilege escalation via tool chaining	Out-of-scope branch modification; duplicate comment injection; rule-conflict resolution bypass
Evolution	Govern.	Model upgrade requires regression over all tier-boundary and escalation tests; clinical sign-off	Model upgrade requires re-validation of comment accuracy and scope-boundary tests over 30-day sample

4 Research Directions

R1: Notations and tools for graduated delegated autonomy. Existing goal models (i* [28], KAOS [22]) and uncertainty-aware languages (RELAX [27]) lack constructs for tiered permissions, memory scope, approvals, handoffs, and reversibility. The Agentic Problem Frames framework [17] structures agent boundaries but omits graduated authority and memory governance. Extending these notations to represent autonomy tiers and delegation boundaries is a crucial next step; matching tool support can then be empirically evaluated to see if it improves specification quality and defect detection in controlled inspection studies.

R2: Requirements-to-runtime compilation. We intend to develop methods that transform AJR/ADP artifacts into executable harnesses (tool constraints, monitors and evaluation plans). Current orchestration infrastructure [3, 15] can directly encode some ADP components (tool permission lists, memory retention windows, approval gate triggers); security-oriented work on design patterns for prompt-injection resilience [5] shows that some agent constraints are already partially compilable, though purpose semantics and provenance obligations require richer notation yet to be developed.

R3: Requirements-driven evaluation and testing. System-level testing must be guided by requirements [25]; accordingly, agent evaluation should be systematically derived from the ADP rather than improvised post-hoc. This formalisation directly addresses a significant transparency gap highlighted by the 2025 AI Agent Index [19], which found that most deployed frontier agents lack publicly disclosed safety evaluations. Rather than implying a complete absence of internal testing, this widespread lack of disclosure underscores an industry-wide absence of verifiable, requirements-level safety commitments. For our running examples, ADP-derived test scenarios provide this missing verifiability by targeting critical constraints: tier-boundary violations, missing provenance, and escalation pathways for medical discharge, as well as out-of-scope modifications and rule-conflict bypasses for code review. Automatically translating these ADP fields directly into concrete test oracles remains an open design-science challenge.

R4: Human agency, trust, and accountability. Delegating autonomy changes who decides, who approves, and who is responsible. Ronanki [18] and Holzinger *et al.* [8] highlight that human oversight in autonomous systems is often superficial (present in design but absent in deployed behaviour) and that accountability erodes as autonomy increases. Emerging regulatory frameworks compound this urgency [11]. RE should make socio-technical commitments explicit: specifying not only what the agent may do, but the oversight role each stakeholder plays and the recourse available when the agent errs. Empirical work on how developers, clinicians, and auditors engage with ADP specifications, e.g., Do they update them after incidents? is needed to assess practical governability.

Limitations. The AJR and ADP are RE artifacts grounded in practitioner consultation and contrasting worked examples; we present them as a starting point for the community and practitioners to evaluate, refine, and build upon, with empirical validation on real-world agentic systems as the immediate next step.

5 Conclusion

Agentic AI is not simply "AI with tools"; it is software to which humans delegate bounded, graduated autonomy. That delegation boundary (what may be done, level of oversight, governance level) is the missing object in current RE practice. We have proposed the AJR to discipline teams to justify agent adoption against simpler alternatives, and the ADP to specify the terms of delegation as a graduated policy. We have illustrated both artifacts with two contrasting examples spanning safety-critical regulated systems and lower-stakes development tooling. Making delegated autonomy explicit, reviewable, and testable is the central RE challenge of the agentic AI era; this paper establishes the vocabulary and artifacts that make that challenge tractable.

Data Availability Statement

There is no additional data to report for this paper.

References

[1] 1996. Health Insurance Portability and Accountability Act of 1996. Public Law 104-191, 110 Stat. 1936. https://hhs.gov

[2] Khlood Ahmad, Mohamed Abdelrazek, Chetan Arora, Muneera Bano, and John Grundy. 2023. Requirements engineering for artificial intelligence systems: A systematic mapping study. *Information and Software Technology* 158 (2023), 107176. https://doi.org/10.1016/j.infsof.2023.107176

[3] Anthropic. 2024. Building effective agents. Anthropic Research. https://www.anthropic.com/research/building-effective-agents Accessed: 2026-04-27.

[4] Anthropic. 2025. Our framework for developing safe and trustworthy agents. Anthropic News. https://www.anthropic.com/news/our-framework-for-developing-safe-and-trustworthy-agents Accessed: 2026-04-27.

[5] Luca Beurer-Kellner, Beat Buesser, Ana-Maria Creţu, Edoardo Debenedetti, Daniel Dobos, Daniel Fabian, Marc Fischer, David Froelicher, Kathrin Grosse, Daniel Naeff, et al. 2025. Design patterns for securing llm agents against prompt injections. *arXiv preprint arXiv:2506.08837* (2025).

[6] Umm e Habiba, Markus Haug, Justus Bogner, and Stefan Wagner. 2024. How mature is requirements engineering for AI-based systems? A systematic mapping study on practices, challenges, and future research directions. *Requirements Engineering* 29, 4 (2024), 567–600. https://doi.org/10.1007/s00766-024-00432-3

[7] Ahmed E Hassan, Hao Li, Dayi Lin, Bram Adams, Tse-Hsun Chen, Yutaro Kashiwa, and Dong Qiu. 2025. Agentic software engineering: Foundational pillars and a research roadmap. *arXiv preprint arXiv:2509.06216* (2025).

[8] Andreas Holzinger, Kurt Zatloukal, and Heimo Müller. 2025. Is human oversight to AI systems still possible? , 59–62 pages.

[9] Kaicheng Huang, Fanyu Wang, Yutan Huang, and Chetan Arora. 2025. Prompt engineering for requirements engineering: A literature review and roadmap. In *2025 IEEE 33rd International Requirements Engineering Conference Workshops (REW)*. IEEE, 548–557.

[10] Yutan Huang, Chetan Arora, Wen Cheng Huong, Tanjila Kanij, Anuradha Madugalla, and John Grundy. 2026. Ethical concerns of generative AI and mitigation strategies: A systematic mapping study. *Applied Soft Computing* (2026), 114789.

[11] Luca Nannini, Adam Leon Smith, Michele Joshua Maggini, Enrico Panai, Sandra Feliciano, Aleksandr Tiulkanov, Elena Maran, James Gealy, and Piercosma Bisconti. 2026. AI agents under EU law. *arXiv preprint arXiv:2604.04604* (2026).

[12] Michael T. Nygard. 2011. Documenting Architecture Decisions. https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions.

[13] National Institute of Standards and Technology. 2023. *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. Technical Report. U.S. Department of Commerce. https://doi.org/10.6028/NIST.AI.100-1

[14] National Institute of Standards and Technology. 2024. *Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile*. Technical Report. U.S. Department of Commerce. https://doi.org/10.6028/NIST.AI.600-1

[15] OpenAI. 2025. A practical guide to building agents. OpenAI Developers. https://platform.openai.com/docs/guides/agents Accessed: 2026-04-27.

[16] OWASP Foundation. 2025. LLM Prompt Injection Prevention Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/LLM_Prompt_Injection_Prevention_Cheat_Sheet.html Accessed: 2026-04-27.

[17] Chanjin Park. 2026. Agentic Problem Frames: A Systematic Approach to Engineering Reliable Domain Agents. *arXiv preprint arXiv:2602.19065* (2026).

[18] Krishna Ronanki. 2025. Facilitating Trustworthy Human-Agent Collaboration in LLM-based Multi-Agent System oriented Software Engineering. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. 1333–1337.

[19] Leon Staufer, Kevin Feng, Kevin Wei, Luke Bailey, Yawen Duan, Mick Yang, A Pinar Ozisik, Stephen Casper, and Noam Kolt. 2026. The 2025 AI Agent Index: Documenting Technical and Safety Features of Deployed Agentic AI Systems. *arXiv preprint arXiv:2602.17753* (2026).

[20] Yongjian Tang and Thomas Runkler. 2026. LLM-Based Agentic Systems for Software Engineering: Challenges and Opportunities. *arXiv preprint arXiv:2601.09822* (2026).

[21] Karthik Vaidhyanathan and Henry Muccini. 2025. Software Architecture in the Age of Agentic AI. In *Software Architecture*. Lecture Notes in Computer Science, Vol. 15982. Springer, 57–71. https://doi.org/10.1007/978-3-032-04403-7_5

[22] A. van Lamsweerde. 2001. Goal-oriented requirements engineering: a guided tour. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering*. 249–262. https://doi.org/10.1109/ISRE.2001.948567

[23] Axel Van Lamsweerde. 2009. *Requirements engineering: From system goals to UML models to software*. Vol. 10. Chichester, UK: John Wiley & Sons.

[24] Andreas Vogelsang and Markus Borg. 2019. Requirements engineering for machine learning: Perspectives from data scientists. In *2019 IEEE 27th international requirements engineering conference workshops (REW)*. IEEE, 245–251.

[25] Fanyu Wang, Chetan Arora, Chakkrit Tantithamthavorn, Kaicheng Huang, and Aldeida Aleti. 2026. Requirements-driven automated software testing: A systematic review. *ACM Transactions on Software Engineering and Methodology* (2026).

[26] 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 (2025), 70. https://doi.org/10.1007/s10515-025-00544-2

[27] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Jean-Michel Bruel. 2010. RELAX: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering* 15, 2 (2010), 177–196. https://doi.org/10.1007/s00766-010-0101-0

[28] Eric SK Yu. 1997. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of ISRE'97: 3rd IEEE International Symposium on Requirements Engineering*. IEEE, 226–235.